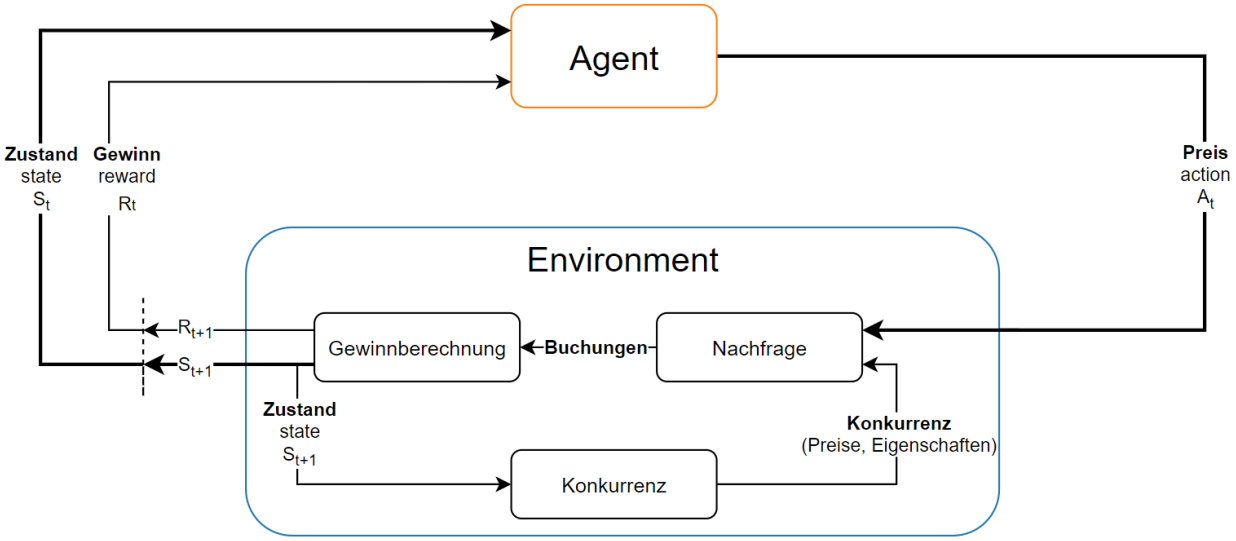
7. JUNI 2022

DYNAMIC PRICING VON HOTELPREISEN MITTELS REINFORCEMENT LEARNING



Bachelorarbeit – HSLU Informatik

Auftraggeber: Hochschule Luzern (HSLU)
Betreuer: Daniel Pfäffli
Experte: Tobias Baltensperger

Verfasser

Tobias Heller

Abstract

Die vorliegende Arbeit «Dynamic Pricing mit Reinforcement Learning in der Hotellerie» befasst sich damit, wie mit Offline Reinforcement Learning eine datenbasierte Preisstrategie entwickelt werden kann, welche den Gewinn eines Hotels maximiert und dabei die Erklärbarkeit sicherstellt.

Dynamic Pricing befasst sich mit dem Themengebiet, wie durch Analyse des Kunden- und Konkurrenzverhaltens der Preis eines Produktes festgelegt werden kann. Reinforcement Learning bietet einen Ansatz, wie mit der Simulation mit historischen Daten der Gewinn maximiert werden kann.

Diese Problemstellung wird im Rahmen dieser Arbeit in einem iterativ-inkrementellen Prozess bearbeitet. Dabei werden sowohl die Konkurrenz als auch die Kunden des zu simulierenden Hotels modelliert. Das zu simulierende Hotel wird dabei als Reinforcement Learning Algorithmus implementiert und auf den historischen Buchungsdaten trainiert.

Es kann gezeigt werden, dass mit Hilfe von historischen Preisdaten ein Konkurrenzmodell auf Basis von XGBoost trainiert werden kann, welches auf den drei Testdatensätzen ein Bestimmtheitsmass R^2 von 0.62, 0.718 und 0.81 erreicht. Weiter wird ersichtlich, dass mit den vorhandenen Daten und Modellen kein Nachfragemodell entwickelt werden kann, welches die Konkurrenz signifikant einbezieht.

Schlussendlich wird eine parametrisierbare Reinforcement Learning Umgebung entwickelt und validiert. Diese erlaubt durch eine standardisierte Schnittstelle eine simple Integration von neuen Reinforcement Learning Algorithmen, Nachfrage- und Konkurrenzmodellen. In einem Experiment kann gezeigt werden, dass es möglich ist, für ein einfaches Nachfragemodell mit einem Conservative Q-Learning Algorithmus eine Preisstrategie zu entwickeln, welches die historischen Daten im Mittel um 82.49 Reward (Gewinn) pro Tag übertrifft.

Inhaltsverzeichnis

Problemstellung

Früher haben Hotelbetriebe ihre Zimmer für einen fixen Preis pro Nacht angeboten, was heute immer weniger der Fall ist. Die Hotels passen die Preise aufgrund der aktuellen Nachfrage dynamisch an. In vielen Betrieben wird das durch einen Revenue Manager umgesetzt. Dieser Ansatz ist sehr kostenintensiv, da zur Bestimmung der Preisstrategie manuell die Auslastungsgraphen, wie auch die Konkurrenzpreise analysiert werden müssen. Deshalb suchen Hotels günstigere und skalierbarere Ansätze. Eine mögliche technische Lösung ist das datengetriebene Dynamic Pricing.

Das Algorithmic Business Research Lab (ABIZ) der Hochschule Luzern forscht zusammen mit dem Startup «Room Price Genie» zu diesem Thema. Mit Daten aus dieser Forschungsarbeit soll in einem explorativen Verfahren untersucht werden, wie und ob mit Reinforcement Learning (RL) eine solche Preisstrategie gelernt werden kann. Dabei soll ein Agent trainiert werden, welcher den Gewinn eines Hotels maximiert und gleichzeitig erklärbar bleibt. So kann bei den Hotelbetrieben und den Endkunden die notwendige Akzeptanz für dynamische Preise geschaffen werden.

In der vorliegenden Arbeit wird für die Entwicklung einer datenbasierten Preisstrategie Reinforcement Learning verwendet. Reinforcement Learning bietet im Gegensatz zur Methode des «Supervised Learning» die direkte Möglichkeit die Maximierung des Gewinns über den «Reward» zu modellieren. Konkret wird Offline Reinforcement Learning eingesetzt, welches im Unterschied zum Online Reinforcement Learning ausschliesslich auf historischen Daten basiert und keine Exploration während des Trainings ermöglicht. Damit soll die Preisstrategie aus Reservations- und Preisdaten gelernt werden.

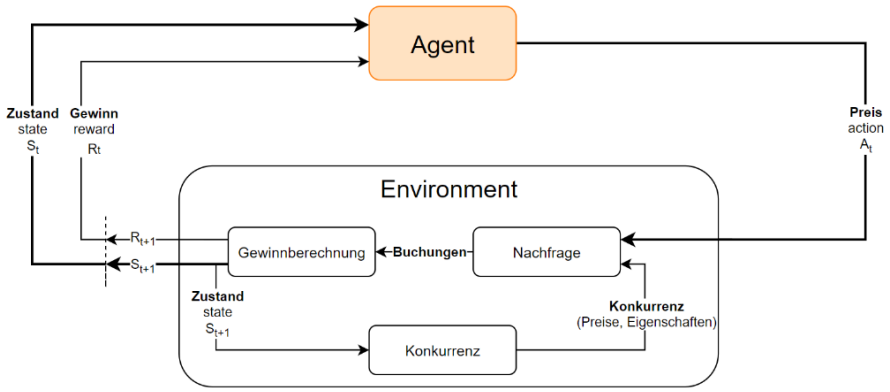


Abbildung 1: Problemstellung "Dynamic Pricing mit Reinforcement Learning in der Hotellerie"

Damit die Daten für das Training mit einem Offline Reinforcement Learning Algorithmus verwendet werden können, müssen diese zusätzlich verarbeitet werden. Dies bedingt eine Modellierung der Umgebung der Hotels, welche folgende Komponenten beinhaltet:

- Nachfrage
- Konkurrenz
- Gewinnberechnung

Mit dieser Umgebung und den historischen Daten wird ein Offline Reinforcement Learning Modell trainiert, welches eine gewinnmaximierende Preisstrategie erlernt. Dieses Modell wird validiert und für zwei Zeiträume analysiert. Dies soll dazu dienen, mittels qualitativer und quantitativer Betrachtung die erzielten Gewinne und die erlernte Preisstrategie zu evaluieren.

Schlussendlich soll damit der Grundstein gesetzt werden, dass eine datenbasierte Preisstrategie auch für kleinere Gasthäuser eine skalierbare und kostengünstige Art des Dynamic Pricing wird.

Vereinfachungen der Problemstellung

Eine Modellierung einer Dynamic Pricing Umgebung kann beliebig komplex umgesetzt werden. Verschiedenste Komponenten wie Konkurrenz, Kundenverhalten sowie Stornierungen können in verschiedensten Realitätsgraden konzipiert werden.

Um den Rahmen der Arbeit einzugrenzen und die Komplexität der Modellierung zu reduzieren, werden in der Aufgabenstellung folgende Vereinfachungen definiert:

- Das Buchungszeitfenster beträgt einen Tag (folgende Nacht)
- Keine Stornierungen
- Keine Rabatte

Zusätzlich wird immer nur ein einziges Hotel als Reinforcement Learning Agent simuliert, welches in Konkurrenz zu genau drei anderen Hotels steht. Folglich wird in der vorliegenden Arbeit ein Single-Agent Ansatz betrachtet.

Im Rahmen des **Ausblicks** werden mögliche Erweiterungen durch grössere Zeiträume, einen Einbezug von Stornierungen und die Erweiterung auf einen Multi-Agent Ansatz besprochen.

Stand der Forschung

Um die Beantwortung der Problemstellung auf Basis von bisherigen Forschungsresultaten umzusetzen, werden im folgenden Kapitel die Konzepte des **Dynamic Pricing** näher erläutert. Dabei werden unter anderem verschiedene Methoden aufgezeigt und auf die technischen Hintergründe des **Offline Reinforcement Learning** und im spezifischen des **Conservative Q-Learning** Algorithmus eingegangen.

Dynamic Pricing

Durch die Verlagerung von einem Verkäufer- zu einem Käufermarkt sind viele Unternehmen mit den Themen der Preissetzungsstrategien konfrontiert. Deshalb werden die Preise immer mehr nicht mehr statisch für ein Produkt definiert, sondern vielmehr dynamisch angepasst. Im Verlauf der Zeit wird der Preis aufgrund einer veränderten Zahlungsbereitschaft des Kunden oder dem Markteintritt eines Konkurrenten bestimmt.

Heute werden mit Hilfe von Datenauswertungen die Preise anhand des Nachfrageverhaltens der Kunden, der Konkurrenzaktivitäten oder der Verfügbarkeit des Produktes dynamisch berechnet. Dabei ist es das Ziel, dass damit eine Kenngrösse wie der Gewinn oder die Nettomarge maximiert werden soll. Folglich ist es beispielsweise möglich, in Zeiten von unelastischer Nachfrage einen höheren Preis zu verlangen. Bei unelastischer Nachfrage führt eine Erhöhung des Preises um 1% um eine Verringerung der Nachfrage von weniger als 1%. Dies ist bei Flügen beispielsweise am Beginn des Urlaubs der Fall, wo die Tickets auch zu einem höheren Preis gebucht werden. Andererseits werden in Phasen, in welcher die Nachfrage sensibler reagiert, die Preise gesenkt. Folglich kann das Dynamic Pricing als eine preisorientierte Möglichkeit der Nachfragesteuerung betrachtet werden. (Klein & Steinhardt, 2008)

Um dieses Dynamic Pricing umzusetzen, gibt es unterschiedlichste Ansätze, welche Teils in der Praxis bereits eingesetzt werden.
Zurzeit wird in der Hotellerie oftmals ein dediziertes Team von Revenue Managern eingesetzt, um anhand der Auslastung und des Preises der Konkurrenz einen Preis für den nächsten Tag festzulegen. Diese dynamische Anpassung der Preise basiert auf einer grossen Menge an Expertenwissen, welche eine korrekte Interpretation diverser Kurven, wie beispielsweise Auslastungsgraphen, voraussetzen. Eine Skalierung dieses manuellen Revenue Management Prozesses wird von Bayoumi et al. vorgeschlagen. Dabei wird für ein Hotel durch das Personal ein saisonaler Referenzpreis definiert. Anschliessend variiert der Preis rund um diesen Referenzwert aufgrund von verschiedenen Variablen, die als Multiplikatoren dienen. Diese Einflussvariablen sind zum Beispiel die aktuelle Hotelbelegung oder die Zeit bis zur effektiven Übernachtung. Mit Hilfe eines Optimierungsalgorithmus, welcher auf einer Monte Carlo Simulation basiert, wird anschliessend der Preis bestimmt. Dieser wird aufgrund der Simulation der Reservationen, Saisonalität, Trend oder auch Stornierungen berechnet. Dieser Ansatz wurde im Rahmen einer Case-Study für das Plaza Hotel in Alexandria, Ägypten, überprüft und validiert. (Bayoumi et al., 2013)

Ein anderer Ansatz, welcher vor allem in der Forschung betrachtet wird, ist das Berechnen des Preises mit Hilfe der Spieletheorie. Dabei wird ein Equilibrium gesucht, bei welchem zwei konkurrierende Hotels den Preis nicht mehr verändern wollen. Dieser Ansatz modelliert somit eine Duopol-Situation, welche in der Realität nur sehr bedingt anwendbar ist. Jedoch kann mit diesem vereinfachten Modell sehr gut gezeigt werden, wie die Konkurrenz auf veränderte Preise reagiert, was in den meisten Revenue Management Systemen nicht betrachtet wird. (Martínez-de-Albéniz & Talluri, 2011)

Weiter werden in der volkswirtschaftlichen Forschung ökonometrische Modelle betrachtet. Dabei werden Modellannahmen über die Nachfrage, die Belegung der Hotels, das Kundenverhalten und die Konkurrenzpreise gemacht und damit eine Funktion erzeugt. Basierend darauf wird eine Parameteroptimierung berechnet, um schliesslich eine optimale Funktion für den Preis zu erhalten. (Lee, 2019)

Zusätzliche Möglichkeiten bietet das Gebiet des Deep Learning, wo mit Hilfe von historischen Daten eine Vorhersage des Preises getroffen werden soll. Zhang et. al erreichte dies in einem 3-Schritt-Verfahren. Dabei wird im ersten Teil das Hotel und seine Konkurrenz analysiert, um damit einen Basispreis zu definieren. In einem weiteren Schritt wird die Belegung des Hotels in der Zukunft mit einem selbstentwickelten seq2seq Modell vorhergesagt. Und zuletzt wird ein Deep Neural Network verwendet, um den Preis vorherzusagen. Als Input verwendet das Neuronale Netz den Basispreis, die vorhergesagte Hotelbelegung und zusätzliche Faktoren. Ein Vergleich mit einer regelbasierten Dynamic Pricing Strategie, welche durch Revenue Management Experten erstellt wurde, zeigt, dass das Modell Preisanpassungen macht, welche zu mehr Umsatz führen und das Revenue Management zusätzlich weniger personalintensiv wird. (Zhang et al., 2019)

Im Rahmen der vorliegenden Arbeit werden ausschliesslich Reinforcement Learning Modelle betrachtet. Auch in diesem Fachgebiet gibt es bereits eine beträchtliche Menge an Forschungsergebnissen. Dabei ist ein Fokuspunkt die Vorhersage von fairen Preisen. Dieses Gebiet der Forschung beschäftigt sich mit der Problemstellung, dass man trotz einer Gewinnmaximierung zwischen den einzelnen Kundengruppen faire Preise anbietet. Dadurch kann ein möglicher Reputationsschaden durch die Kunden aufgrund von unfairer Preispolitik verhindert und somit Langzeitverluste vermieden werden. Im Gegensatz zu den bereits vorgestellten Verfahren, fokussiert sich der Algorithmus nicht ausschliesslich auf die Maximierung des Gewinns des Hotels, sondern auch auf die Maximierung der Fairness. Diese Fairness wird dabei mit Hilfe des Jain Index gemessen und überprüft. Zur Berechnung des Jain Index wird der Variationskoeffizient, ein relatives Streuungsmass, verwendet. Die Kundengruppen in einem solchen Experiment werden als vier verschiedene logistische Funktionen modelliert, wobei diese unterschiedliche Preissensitivitäten aufweisen. Eine Preissensitivität bezeichnet den Einfluss des Preises auf die Kaufentscheidung eines potenziellen Kunden. Die ersten beiden Kundengruppen akzeptieren bei tieferem Preis eher das Angebot. Sie folgen somit einer Nachfragefunktion eines regulären Gutes. Die zweite Kundengruppe akzeptiert hingegen viel höhere Preise. Die dritte Kundengruppe, die modelliert wird, hat das inverse Verhalten und akzeptiert die Angebote bei höherem Preis mit einer höheren Wahrscheinlichkeit. Die letzte Kundengruppe reagiert nicht auf den Preis und akzeptiert somit jedes Angebot. Folglich soll das Modell den Umsatz jeder Gruppe maximieren, wobei jedoch dabei die Preisunterschiede zwischen den Kundengruppen möglichst klein gehalten wird. Dazu wird für die Schätzung der Q-Values des Q-Learning auf ein neuronalen Netz, welches mit Gradient Descent trainiert wird, zurückgegriffen. In der Arbeit von Maestre et al. konnte gezeigt werden, dass es möglich ist eine gewisse Fairness in einem Dynamic Pricing Modell zu erreichen. Dazu muss sichergestellt werden, dass zusätzlich zum Umsatz auch die Fairness maximiert wird. Zusätzlich wird darauf hingewiesen, dass zusätzliche Möglichkeiten mit einem evolutionären Algorithmus in einer komplexeren Umgebung ausgeschöpft werden können. (Maestre et al., 2018)

Kastuis et. al vergleicht in seiner Arbeit die Anwendung eines Deep Q-Networks (DQN) mit einem Soft Actor Critic (SAC) Algorithmus für Dynamic Pricing Modelle. Ein DQN berechnet, wie der Name impliziert, die Q-Values, anhand welcher in einem Folgeschritt mit dem höchsten erwartbaren Reward eine optimale Policy evaluiert wird. Das heisst, der DQN Algorithmus basiert auf dem relativ einfachen Prinzip von Q-Learning. Der andere betrachtete Algorithmus, der SAC, ist ein Policy Gradient Algorithmus. Das heisst, der Algorithmus berechnet direkt die beste Policy (Preisstrategie) und verwendet keine Value Funktion.

Dabei wird die Performance der Algorithmen mit der optimalen Policy aus dem Dynamic Programming Verhalten verglichen. Dies wird ausschliesslich für den Duopol Ansatz umgesetzt, wo man zusätzlich überprüfen kann, ob es Tendenzen zur Kartellbildung gibt. Eine Kartellbildung würde bedeuten, dass es zu impliziten Preisabsprachen zwischen den beiden Agenten kommt, was zu überhöhten Preisen und folglich überhöhtem Gewinn führt. Um diesen Duopol Markt zu simulieren, muss der Agent das Kundenverhalten und den Preis der Konkurrenz verstehen. Dazu wird der Kunde als logistische Funktion modelliert, welcher auf realen Daten basiert. Das Modell gibt die Wahrscheinlichkeit eines Kaufs zurück, anhand von gewissen Features wie Preisrangierung und Preisdifferenz zur Konkurrenz. Grundsätzlich wird gezeigt, dass tiefere Preise die Verkäufe erhöhen. Um den Umsatz zu maximieren, muss der Agent das richtige Preisniveau finden und auch seine Konkurrenz unterbieten, um so die Verkäufe zu erhöhen.

Der Zustandsraum wird relativ einfach definiert und ist ausschliesslich abhängig vom Preis der Konkurrenz. Im Action-Space werden alle möglichen Preise in einem diskreten Bereich definiert. Für das DQN wird dies diskretisiert auf ganze Zahlen, was für den SAC nicht notwendig ist. Das Deep Q-Network besteht dabei aus drei «hidden Layers» mit jeweils 128 Neuronen. Das Soft Actor-Critic Modell hingegen besteht aus jeweils drei verketteten «hidden Layers» mit 128 Neuronen für den «Critic» und den «Actor». Beide Modelle verwenden Experience Replay, um so bereits berechnete Episoden erneut für die Policy Optimierung zu verwenden.

Um die Konkurrenz zu modellieren, werden zwei Arten von manuell aufgesetzten Verhalten definiert. Einerseits die deterministischen Modelle und andererseits die nicht-deterministischen Modelle. Für den Vergleich wird der Erwartungswert der Modelle mit dem optimalen Erwartungswert verglichen. Die deterministische Konkurrenz wird in verschiedenen Varianten modelliert. Eine Möglichkeit ist es, dass die Konkurrenz einen fixen Preis definiert. Die optimale Strategie ist es in diesem Fall, die Konkurrenz mit einer kleinen Preisdifferenz zu unterbieten, um so den Umsatz zu maximieren. In diesem Fall ist die Performance des DQN besser. Das SAC konvergiert nicht, das DQN jedoch schwankt sehr stark um den optimalen Wert, was das Modell unzuverlässig macht. Eine weitere Strategie ist das Unterbieten. In diesem Fall senkt die Konkurrenz den Preis um einen fixen Betrag unter den Preis des Agenten. Um eine einigermassen gute Profitabilität zu erreichen, sollte der Agent es vermeiden, den Preis unter die Profitabilitätsgrenze zu senken. Ansonsten wird die Konkurrenz einen solch tiefen Preis setzen, dass der Gesamtumsatz für alle marginal wird. In der Simulation wird sichtbar, dass das DQN sehr häufig zustandsunabhängige Fixpreise setzt und sehr schlecht mit komplexen Preisstrategien umgehen kann. Der SAC hingegen kann sehr gut mit komplexen Szenarien umgehen, performt dann aber unzureichend, wenn die Konkurrenz fixe Preise setzt.

In den Simulationen mit der nicht deterministischen Konkurrenz kann gezeigt werden, dass dies kein Problem für sowohl den DQN als auch den SAC Algorithmus darstellt. Grundsätzlich konnte für alle Problemstellungen dargelegt werden, dass beide Algorithmen gute Ergebnisse erzielen. Jedoch sind in einigen Fällen eine sehr grosse Anzahl Episoden >400k notwendig, um eine gute Performance zu ermöglichen. (Kastius & Schlosser, 2022)

Offline Reinforcement Learning

Eine Art der Reinforcement Learning Algorithmen sind die so genannten Offline Reinforcement Learning Algorithmen. Diese werden oftmals als «datengetriebenes» Reinforcement Learning beschrieben. Das Ziel dieses Vorgehens ist es den kumulativen Reward zu maximieren. Jedoch hat der Agent im Gegensatz zum Online Reinforcement Learning keine Möglichkeit mehr mit dem Environment zu interagieren und Exploration zu verwenden.

Die möglichen Rewards zwischen den States sind durch einen statischen Datensatz definiert und die beste Policy muss anhand dieser Daten gelernt werden. Dadurch erreicht man ein Setting, welches sehr nahe am Supervised Learning Problem ist, wobei man diesen statischen Datensatz als Trainingsset ansehen kann. Folglich muss der Offline RL Algorithmus im Stande sein mit diesem fixen Datensatz die Policy $\pi(\boldsymbol{a}|s)$[1] mit dem maximalen Reward zu erarbeiten. Die Zustände und Rewards in diesem Datensatz wurden dabei zuvor durch eine Behaviour Policy, welche unbekannt sein kann, erzeugt.

Dieses Problem kennt man bereits aus den Off-Policy RL Algorithmen wie zum Beispiel Q-Learning oder actor-critic Algorithmen, bei welchen die aktuellen Übergänge D mit einer anderen Policy als der aktuellen $\pi(\boldsymbol{a}|s)$ gesammelt wurden. Im Gegensatz zum Offline Reinforcement Learning findet beim Off-Policy Reinforcement Learning zu gewissen Zeitpunkten eine Interaktion mit der Umgebung statt. Das ist sogleich auch der Hauptunterschied der beiden Verfahren. Deshalb spricht man beim Offline Reinforcement Learning häufig auch von «full off-policy» Algorithmen, da zu keinem Zeitpunkt des Trainings eine online Interaktion mit der Umgebung möglich ist.

Dieser Aufbau ermöglicht es, einen RL Agent zu trainieren, ohne dabei ein vollständiges Environment zu erzeugen. Dies spart einerseits sehr viel Aufwand in der Modellierung. Zusätzlich reduziert es auch falsche Annahmen der Umgebung über das richtige Verhalten, da man effektives Verhalten aus der realen Welt als Daten verwendet.

Dennoch gibt es auch einige Schwierigkeiten in der Verwendung von Offline Reinforcement Learning Algorithmen. Wie bereits erwähnt ist es unmöglich, Exploration zu machen. Das heisst, wenn das Datensatz D keine High-Reward Szenarien abbildet, kann es unmöglich sein, dass der Algorithmus dieses Verhalten lernt. Eine noch wichtigere und in der Praxis relevantere Herausforderung ist das Ausführen von «counterfactual queries». In diesem Fall muss das System eine Antwort haben auf Aktionen, welche unterschiedlich sind als die, welche das Datensatz D beinhaltet. Dies muss unterstützt werden, weil man eine Policy erlernen will, welche besser als die Behaviour Policy ist. Die grösste Herausforderung sind dabei die «distributional shifts». Das heisst, das System wird mit einer gewissen Verteilung der Zustände trainiert und dann auf einer anderen evaluiert. Folglich werden andere Zustände besucht, was zu einem anderen Verhalten führt. Dieser Unterschied der Verteilung kann zu schwerwiegenden Fehlern führen. Um dies zu verhindern, gibt es die Möglichkeit, Einschränkungen einzuführen, welche sicherstellen, dass die gelernte Policy $\pi(\boldsymbol{a}|s)$ nicht zu stark von der Behaviour Policy abweicht. (Levine et al., 2020)

[1] $\pi(\boldsymbol{a}|s)$: Bedingte Wahrscheinlichkeit der Wahl der Aktion $\boldsymbol{a}$ im Zustand s

Conservative Q-Learning

Ein Algorithmus, welcher vielversprechende Verbesserungen gegenüber klassischen Offline-RL Algorithmen ermöglicht, ist das Conservative Q-Learning (CQL). Es verbessert dabei die Performance von RL Algorithmen damit, dass es die optimistischen Schätzungen der Value Funktion beim Bootstrapping ausserhalb der Verteilung des Datensatzes D reduziert. Dabei wird in jeder Policy Iteration gegen eine untere Grenze der Wertfunktion optimiert. Folglich reduziert man die Tendenz zur Überschätzung der Q-Values. Konkret wird dies erreicht, indem eine Regularisierung der Q-Values während des Trainingsprozesses angewendet wird. Im Gegensatz zu den klassischen Offline RL Algorithmen, sind durch diese Änderung der Q-Funktion keine zusätzlichen Einschränkungen notwendig.

Die Implementation basierend auf klassischen Online RL Algorithmen wie Q-Learning kann sehr einfach umgesetzt werden. So kann in 20 Zeilen Code eine Implementation basierend auf der Standardversion des soft actor-critic (SAC) für kontinuierliche Probleme umgesetzt werden. Dies ist ebenfalls auf Basis des QR-DQN für diskrete Probleme möglich.

Im Artikel wurde gezeigt, dass im Offline RL Setting vorherige Methoden übertroffen werden. Es erreicht dabei eine bis zu 2-5x bessere Performance auf gängigen Benchmark Aufgaben. Es kann ebenfalls gezeigt werden, dass es der einzige Algorithmus ist, welcher das Behavioral Cloning in bestimmten Fällen übertreffen kann. Dies wird mit diversen Experimenten auf verschiedenen Gym-Standardproblemen wie dem Adroit Task, AntMaze, Kitchen Tasks und Atari dargelegt.

Eine vollständige theoretische Analyse des CQL muss in Zukunft noch durchgeführt werden. Ebenfalls ist sehr klar sichtbar, dass sich Offline RL Methoden sehr stark überanpassen (Overfitting), wie es von den Supervised Learning Methoden bekannt ist. Deshalb müssen einfache «early stopping» Methoden entwickelt werden, ähnlich wie man es aus dem Supervised Learning bereits mit dem Validation Error kennt. (Kumar et al., 2020)

Ideen und Konzepte

Das Hauptziel der vorliegenden Arbeit liegt in der Vorhersage eines Preises für ein Hotelzimmer für einen bestimmten Zeitpunkt in der Zukunft mittels Offline Reinforcement Learning. Dafür werden Teilprobleme definiert und weiter auf die Konzepte der Reinforcement Learning Umgebung eingegangen.

Teilprobleme

Um die Aufgabenstellung der Arbeit «Dynamic Pricing mit Reinforcement Learning für die Hotellerie» besser zu verstehen, wird diese in vier Teilprobleme unterteilt.

Konkurrenz bestimmen und modellieren

Um ein interessantes Dynamic Pricing Modell entwickeln zu können, muss für jedes zu simulierende Hotel die Konkurrenz bestimmt und modelliert werden. Dabei soll diese Konkurrenz eine bestimmte Anzahl Hotels definieren, zwischen welchen in der Realität ein Kunde entscheidet.

Dies bringt erhebliche Herausforderungen mit sich, da in den Daten keine einzelnen Kunden und ihr Entscheidungsprozess vorhanden sind. Deswegen muss die Konkurrenz anhand der Daten sinnvoll bestimmt werden können. Zusätzlich muss die Preisstrategie der Konkurrenz in unterschiedlichen Situationen (Wochenende, Jahreszeit, Preis der anderen Hotels, …) modelliert werden.

Nachfrage vorhersagen

Damit für das zu simulierende Hotel und seine Konkurrenz eine Bestimmung der Anzahl Buchungen gemacht werden kann, muss zuerst die Nachfrage modelliert werden. Anders ausgedrückt soll abhängig von gewissen Eigenschaften der Hotels (Preis, Lage, …) und gewissen zeitlichen Features wie Wochenende oder Saison eine Anzahl Kunden berechnet werden.

Kundenverhalten modellieren

Anhand der Nachfrage wird eine eine Anzahl Kunden vorhergesagt, welche anhand gewisser Kriterien eine Buchung vornehmen. Dabei kann der Kunde zwischen dem simulierenden Hotel und seinen Konkurrenten entscheiden.

Grundsätzlich kann davon ausgegangen werden, dass diese Entscheidung vom Preis der Übernachtung abhängig ist. Jedoch ist es auch denkbar, dass der Kunde aufgrund gewisser Eigenschaften der Hotels nicht unbedingt das preisgünstigste Angebot wählt.

Preisstrategie berechnen und evaluieren

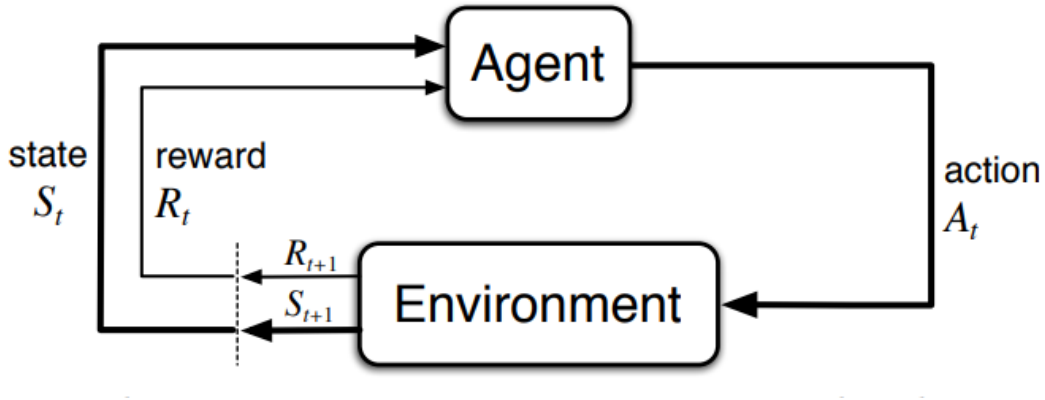
Um eine Preisstrategie mittels eines Reinforcement Learning Modells modellieren zu können, müssen die Komponenten der Konkurrenz, der Nachfrage und der Kunden in einer einzigen Reinforcement Learning Umgebung kombiniert werden. Damit kann ein Algorithmus eine Preisstrategie anhand der historischen Daten lernen.

In einem weiteren Schritt muss diese Preisstrategie analysiert und evaluiert werden. Aufgrund des beschränkten Zeitraums und des finanziellen Risikos ist es nicht möglich den Algorithmus in der Realität mit einem konkreten Hotel zu testen. Folglich müssen die Algorithmen anhand festdefinierten Metriken wie auch durch eine qualitative Betrachtung im Rahmen einer Simulation evaluiert und verglichen werden.

Reinforcement Learning Umgebung

Reinforcement Learning (RL) ist ein Teilgebiet des Machine Learnings (ML). Im Gegensatz zum Supervised Learning wird kein Klassifikations- oder Regressionsproblem gelöst. Sondern RL hat das Ziel, den kumulativen erwarteten Reward zu maximieren, während dem das System mit einer komplexen Umgebung interagiert. (Sutton & Barto, 2018)

Das Reinforcement Learning Setup wird in zwei Teile unterteilt. Einerseits besteht es aus dem Agenten, welcher durch ein Reinforcement Learning (RL) Modell simuliert wird. Andererseits reagiert der Agent, im vorliegenden Fall das Hotel, auf den Zustand, welcher durch eine Umgebung erzeugt wird.



Quelle: Sutton & Barto, 2021

Abbildung 2: Agent-Environment Interaktion

Anhand dieses neuen States und des erhaltenen Rewards kann dann der Agent eine neue Aktion für den Zeitschritt t+1 berechnen.

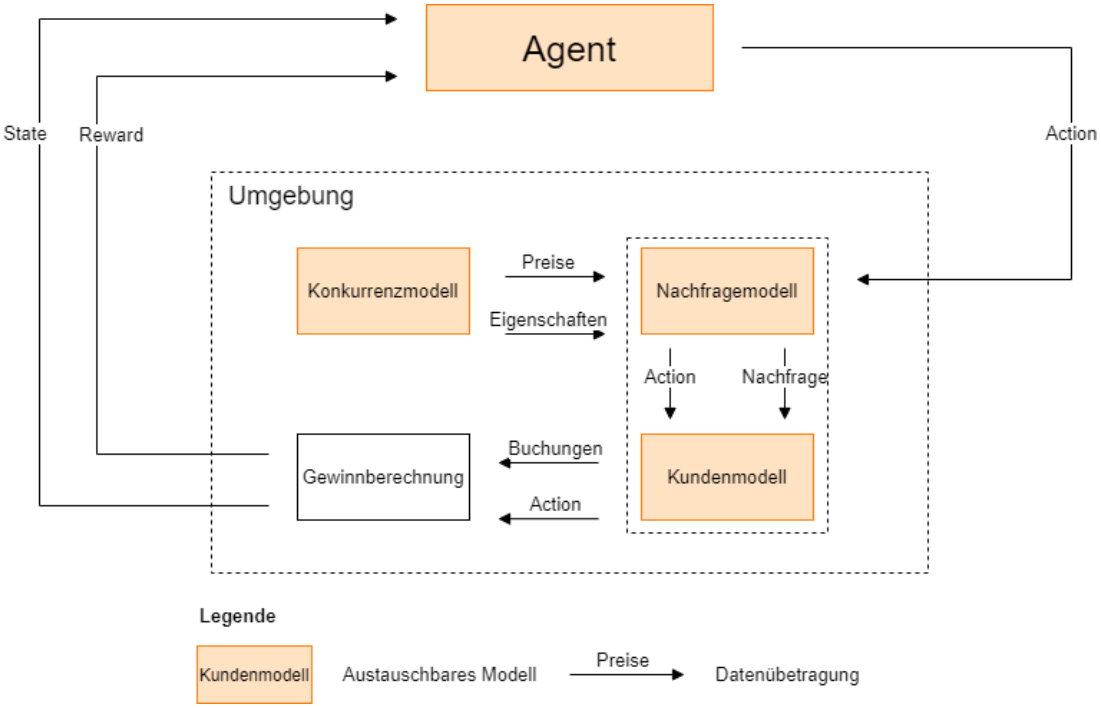


Abbildung 3: Reinforcement Learning Umgebung mit den austauschbaren Modellen

In der vorliegenden Arbeit werden sowohl der Agent als auch die Umgebung mit Konkurrenz und Nachfrage als austauschbare Modelle umgesetzt.

Agent - Das Hotel

Wie bereits im **Stand der Forschung** beschrieben, wird im Dynamic Pricing bei vielen Reinforcement Learning Modellen ein Hotel modelliert. Im vorliegenden Fall wird ein Hotel der einzige Agent im System sein. In der weiteren Arbeit wird dieses zu simulierende Hotel als Agentenhotel bezeichnet.

Aktion

Für jeden Zeitschritt t führt der Agent eine Aktion A_t aus und übergibt diese dem Environment. Im vorliegenden Fall ist die Aktion der Preis pro Nacht für den Tag t. Dabei wird jeweils nur der Preis für die folgende Nacht zurückgegeben.

Die Wahl dieser Aktion ist abhängig vom gewählten Agenten in der jeweiligen Phase. Es wird unterschieden zwischen der Generierungsphase und der Trainings/Evaluierungsphase.

	Generierungsphase	**Trainings/Evaluierungsphase**
Aktion	Preis für eine Nacht	Preis für eine Nacht
Aktionsquelle	Reservationsdaten	RL Algorithmus
Entscheidungsgrundlage	Datum, Hotel	Zustand der RL Umgebung

Tabelle 1: Vergleich Generierungsphase und Trainingsphase

In der Generierungsphase wird die Aktion durch den bestehenden Datensatz definiert. Anhand des Tages und des Hotels wird der effektiv bezahlte Preis ausgegeben. Hingegen in der Trainings- und Evaluierungsphase wird die Aktion durch einen Reinforcement Learning Algorithmus berechnet. Dieser entscheidet anhand seines gelernten Verhaltens und dem erhaltenen Zustand S_t welche Aktion gewählt werden soll.

In beiden Fällen wird ein Preis pro Nacht in Franken gesetzt. Dies ist ein kontinuierliches Intervall und es können auch Preise wie beispielsweise 95.50 CHF pro Nacht gesetzt werden.

Umgebung

Die Umgebung bestimmt die elementaren Funktionalitäten der Simulation. Es repräsentiert den Zustand des Systems und berechnet den Reward, den der Agent durch seine Aktion erzielt.

Reward

Pro Aktion A_t berechnet die Umgebung einen Reward. Im vorliegenden Fall wird der Reward anhand des gesetzten Preises, der Preise der Konkurrenz und der Kundennachfrage berechnet.

Der Reward ist der Gewinn, der für diese Nacht für dieses Hotel bei einem gesetzten Preis A_t erzielt wird. Der Reward berechnet sich folgendermassen:

$$R_{t+1} = A_t * N_t - C_t$$

Das heisst, der Reward berechnet sich aus dem Preis A_t, der Anzahl verkauften Zimmer N_t und den Kosten des Hotels C_t.

In dieser vereinfachten Version sind die Kosten des Hotels linear zur Anzahl Zimmer. Dabei sind die Kosten unabhängig von der Benutzung des Hotels.

Folglich berechnen sich die Hotels folgendermassen:

$$C_t = Z * CZ$$

Wobei Z die Anzahl Zimmer des Hotels sind und CZ die Kosten pro Zimmer repräsentieren. Diese Reward-Funktion dient der Modellierung der Betriebskosten des Hotels und soll einen zu tiefen Preis verhindern. Die entwickelte Reinforcement Learning Umgebung stellt eine einfache Schnittstelle zu Verfügung, welche die Einbindung von komplexeren Kostenfunktionen erlaubt.

State

Für jeden Zeitschritt t wird zusätzlich zum Reward R_t auch der State S_{t+1} zurückgegeben. Dieser repräsentiert den Zustand, in dem sich das System befindet.

Der State des Systems besteht grundsätzlich aus den folgenden fünf Komponenten:

- Aktueller Tag
- Beschreibung des Agentenhotels
- Preis des Agenten bei t-1
- Beschreibung der Konkurrenten
- Preis der Konkurrenten bei t-1

Diese Informationen sind an Daten angelehnt, die einem Revenue Manager zur Verfügung stehen, wenn er einen Preis bestimmt. Einerseits ist der Tag relevant, andererseits die Eigenschaften des Hotels und seiner Konkurrenten. Ebenfalls sind historische Preise wichtig, da diese eine Grundlage bilden, um gute Entscheidungen zu treffen. Dies gibt dem Agenten Informationen, um anhand dieser eine Aktion A_{t+1} zu bestimmen.

Der Zustand des Systems und die Berechnung des Rewards wird massgeblich durch die Kundennachfrage, wie auch die Konkurrenz bestimmt.

Konkurrenz

Der Agent selbst steht in Konkurrenz zu anderen Hotels, welche ebenfalls ihre Zimmer für einen gewissen Preis zur Verfügung stellen. Dabei entscheiden sich schlussendlich die Kunden für das Agentenhotel oder aber für eines der Konkurrenzhotels.

Dabei ist es eine offene Frage, was als Konkurrenz für ein Hotel gilt. Folgende Möglichkeiten für die Bestimmung der Konkurrenz werden betrachtet:

Wahl der Konkurrenz	**Vorteile**	**Nachteile**
Manuelle Definition durch Experten	- Nachvollziehbarkeit - Einfachheit	- Keine Skalierung - Subjektiv
Hotel in unmittelbarer Nähe	- Einfachheit - Automatisierung - Skalierbarkeit	- Kunden entscheiden sich nicht direkt für einen Ort
Hotel mit ähnlichem Preis	- Einfachheit - Skalierbarkeit - Automatisierung	- Kunden entscheiden nicht anhand des Preises, wohin sie gehen - Zielvariable als Selektionskriterium
Ähnliche Hotels aufgrund von Beschreibungen	- Automatisierung - Skalierbarkeit - Echte Similarität	- Auswahl der Beschreibungen und Vergleich

Tabelle 2: Wahl der Konkurrenz

Die erste Möglichkeit mittels manueller Definition wird ausgeschlossen, da es in einem produktiven Einsatz nicht möglich ist, Personal zu finden, welches die lokalen Märkte kennt und skalierbar Konkurrenzhotels zur Verfügung stellen kann. Hotels in unmittelbare Nähe als Konkurrenz zu definieren ist zwar technisch über die Koordinaten und der Haversine-Distanz sehr einfach. Jedoch ist klar, dass Hotels in der Nachbarschaft teils sehr unterschiedliche Zielgruppen haben und dass die Jugendherberge und das 300m entfernte Hilton Hotel nicht konkurrieren.

Um diesen Effekt zu reduzieren, kann die Annahme getroffen werden, dass ähnliche Hotels ähnliche Preise haben. Dadurch kann zum Beispiel mit Hilfe des Vergleichs des Median Preises in der Vergangenheit die Konkurrenz bestimmt werden. Dies führt aber in folgenden Schritten zu Implikationen, da die Zielvariable, der Preis, als Wert für das Bestimmen der Konkurrenz verwendet wird. Folglich wird in der vorliegenden Arbeit eine technische Similarität aufgrund der Beschreibungen der Hotels verwendet. Dies basiert auf der Hypothese, dass sich Hotels konkurrieren, weil sie ähnlich sind. Das heisst die Hotels sind zum Beispiel eher luxuriös und bieten einen Swimming Pool oder das Hotel hat eine Nähe zum Flughafen und hat Arbeitsplätze im Hotelzimmer. Dies kann aus qualitativer Sicht als sinnvolle Konkurrenzsituation betrachtet werden.

Technisch soll dies aufgrund einer numerischen Beschreibung des Hotels und der Kosinus-Ähnlichkeit die Konkurrenten bestimmt werden, welche dem Agentenhotel am ähnlichsten sind. Eine alternative zur Berechnung der Ähnlichkeit ist die euklidische Distanz. Die Kosinus-Ähnlichkeit wird gewählt, damit einfach interpretierbare Werte zwischen -1 und 1 verwendet werden können. Zusätzlich bietet die Kosinus-Ähnlichkeit den Vorteil, dass die Skalierung der Werte einen geringeren Einfluss hat. In einer weiteren Arbeit können mit einem grösseren Datensatz die Unterschiede der Konkurrenz aufgrund des Ähnlichkeitsmasses betrachtet werden.

Nachfrage und Kundemodell

Die Berechnung des **Reward** hängt wie beschrieben von der Anzahl gebuchter Zimmer für das Agentenhotel ab. Die Bestimmung dieser Anzahl ist nicht trivial. Grundsätzlich werden in dieser Arbeit zwei verschiedene Möglichkeiten betrachtet und evaluiert.

Einerseits gibt es ein zweistufiges Verfahren. Es besteht aus einem **Nachfragemodell** und einem **Kundenmodell**. Das Nachfragemodell bestimmt die Anzahl Personen, die Hotels buchen wollen und das Kundenmodell bestimmt in einem zweiten Schritt, für welche Hotels sich diese Kunden schliesslich entscheiden. Die Möglichkeiten und Schwierigkeiten, wie man das Nachfrage- und Kundenmodell kombinieren kann, sind im Anhang unter **Konzepte der Nachfrage- und Kundenmodellierung** beschrieben. Die grundsätzliche Problematik ist dabei, wie man eine kombinierte Nachfrage aus den Eigenschaften (Preis, Ort, Infrastruktur, …) des Agentenhotels und seiner Konkurrenz berechnet.

Dieses sehr komplexe Problem soll das einstufige sogenannte «**Kombiniertes Modell**» lösen, welches aus den Eigenschaften des Agentenhotels und der Konkurrenz direkt die Anzahl gebuchter Zimmer für das Agentenhotel berechnet.

Nachfragemodell

Das Nachfragemodell gibt für einen bestimmten Zeitpunkt t für ein oder mehrere Hotels die Anzahl Kunden zurück. Das Nachfragemodell selbst kann auf mehrere Arten modelliert werden.

In der Volkswirtschaftslehre wird die Nachfrage meist abhängig vom Preis modelliert. Das heisst, für einen bestimmten Preis *x* wird eine Nachfrage *y* berechnet.

In der vorliegenden Arbeit kann diese Nachfrage bestimmt werden, indem ein Modell entwickelt wird, welches anhand der Reservationspreise eine Anzahl Kunden approximiert. Dies ist eine Annäherung, da in den Daten nur Buchungen vorhanden sind und nicht die effektive Nachfrage, welche zu diesen Buchungen geführt haben.Dieses Modell vereinfacht die Nachfrage insofern, dass davon ausgegangen wird, dass die jeweiligen Hotels perfekte (vollkommene) Substitutionsgüter sind und ausschliesslich der Preis die Nachfrage bestimmt. Das würde implizieren, dass die Qualität der Hotels vollständig identisch ist. Dies entspricht jedoch nicht der Realität und die Nachfrage ist sowohl vom Preis wie auch von den Eigenschaften des Hotels abhängig. Folglich sollen diese ebenfalls modelliert werden.

Kundenmodell

Nachdem eine Nachfrage für alle Hotels berechnet ist und die Anzahl Kunden bekannt sind, bestimmt das Kundenmodell die Wahl des Hotels.

Die Wahl des Hotels kann auf verschiedene Arten definiert werden.

- Tiefster Preis
- Kombination aus Preisrang, tiefster Preis, Preisdifferenz und durchschnittlicher Preis (Schlosser & Richly, 2019)
- Kombination aus Preisrang und Eigenschaften des Hotels

Die einfachste Möglichkeit der Entscheidung des Hotels für jeden Kunden ist die Wahl des tiefsten Preises. In diesem Fall werden die Angebote der Hotels (Agent und Konkurrenten) nach aufsteigendem Preis sortiert. Anschliessend wählen die Kunden nacheinander jeweils das günstigste verfügbare Hotel. Dies ist eine sehr starke Vereinfachung der Realität und führt zu einer einfachen dominanten Preisstrategie: Unterbieten der Konkurrenz.

Eine andere Möglichkeit: Die Berechnung der Wahrscheinlichkeit für die Wahl eines Hotels aufgrund der Preisinformationen, zeigt Schlosser & Richly in ihrem Paper auf. Dabei wählt der Kunde anhand der Features:

- Preisrang
- Ob das Hotel der tiefste Preis setzt
- Die Preisdifferenzen zur Konkurrenz
- Der durchschnittliche Preis aller Angebote

mit einer gewissen Wahrscheinlichkeit ein Hotel. Dies bietet auch dem Hotel, welches nicht den tiefsten Preis anbietet, eine Möglichkeit, den Kunden zu erhalten. Für die Bestimmung der Gewichtung der eben erwähnten Features hat Schlosser & Richly eine Simulation durchgeführt, anhand welcher eine logistische Funktion berechnet wurde. Diese Resultate sind nur sehr bedingt auf die vorliegende Arbeit anwendbar, da Schlosser & Richly ein Duopol annehmen. Ebenfalls werden die Eigenschaften des Hotels nur im geringen Mass mit einbezogen. (Schlosser & Richly, 2019)

Schlussendlich kann der Preisrang mit den Eigenschaften des Hotels kombiniert werden. Ein Einbezug dieser Eigenschaften (luxuriös, Swimming Pool, elegant, …) kann berechnet werden, indem man eine zusätzliche Gewichtung der Eigenschaften ermittelt. Denn sowohl für die Agenten als auch für die Konkurrenzhotels stehen die Beschreibungen zur Verfügung. Diese erlauben es, für jeden Kunden eine lineare Kombination zwischen Preisrangierung und Übereinstimmung der Eigenschaften zu berechnen.

Dabei müssen folgende Parameter definiert werden:

- Bestimmung der gewünschten Eigenschaften der Kunden
- Bestimmung der Eigenschaftenähnlichkeit
- Gewichtung Preisrangierung vs. Eigenschaftenähnlichkeit

Dabei ist die Bestimmung dieser Eigenschaften, welche sich die Kunden wünschen, sehr wichtig, da diese einen signifikanten Einfluss auf das Resultat haben.

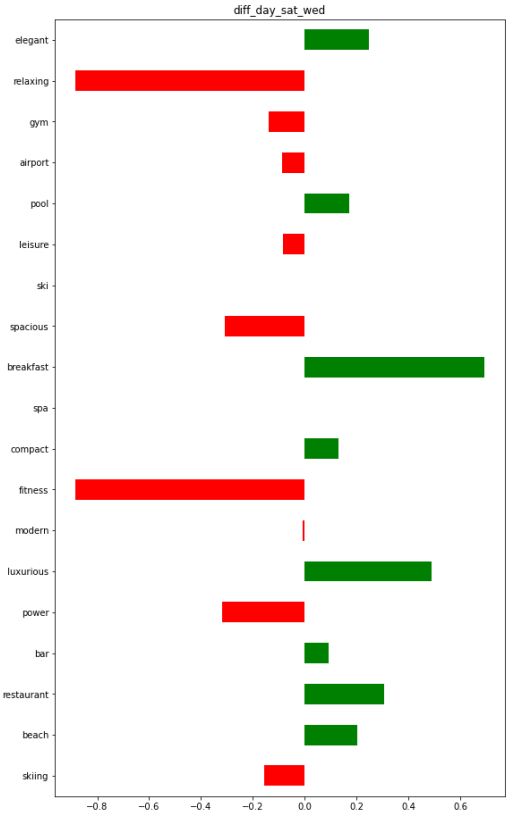
Manuelle Definition der Eigenschaften

Eine mögliche Wahl dieser Eigenschaften kann manuell aufgrund von Expertenwissen geschehen. So wäre es beispielsweise möglich zwei Kunden zu definieren. Der Tourist, welcher vor allem auf Dinge wie Swimming Pool, Freizeit, … achtet oder ein Geschäftskunde, welcher Eigenschaften wie Nähe zum Flughafen oder Konferenzräume als wichtig empfindet.

Diese manuelle Definition der Eigenschaften birgt aber die grosse Gefahr, dass der RL Agent ausschliesslich diese Präferenzen, welche nicht unbedingt der Realität entsprechen, lernt und keine generelle Erkenntnisse aus den Resultaten gezogen werden kann.

Datenbasierte Definition der Eigenschaften

Diese Eigenschaften könnten stattdessen auch datenbasiert, beispielsweise aus dem Forschungsprojekt der Hochschule Luzern, entnommen werden. Im Rahmen des Forschungsprojektes «A Machine-Learning Approach to Small-Hotel Revenue-Management» [2] des ABIZ Teams der Hochschule Luzern wurde auf den Reservationsdaten eine Feature-Wichtigkeit Analyse durchgeführt. Dabei wurde mit Hilfe eines linearen Modells (Lasso) untersucht, welche Keywords relevant sind für die Entscheidung, ob die Preise am Wochenende höher sind (Freizeit) oder unter der Woche (Business).



Quelle: Daniel Pfäffli, 2022

Abbildung 4: Feature Wichtigkeiten für Freizeit (Rot) und Business (Grün) Kunden

Dabei repräsentieren die negativen Werte (Rot) die wichtigen Features für die Freizeitkunden und die positiven Werte (Grün) die Features der Businesskunden.

[2] A Machine-Learning Approach to Small-Hotel Revenue-Management: https://www.hslu.ch/en/lucerne-university-of-applied-sciences-and-arts/research/projects/detail/?pid=5880

Ein Vorschlag zum Einbezug dieser Feature Wichtigkeiten wird im Anhang unter **Mögliche Berechnung des Kundenwerts bei datenbasierten Eigenschaftspräferenzen** beschrieben.

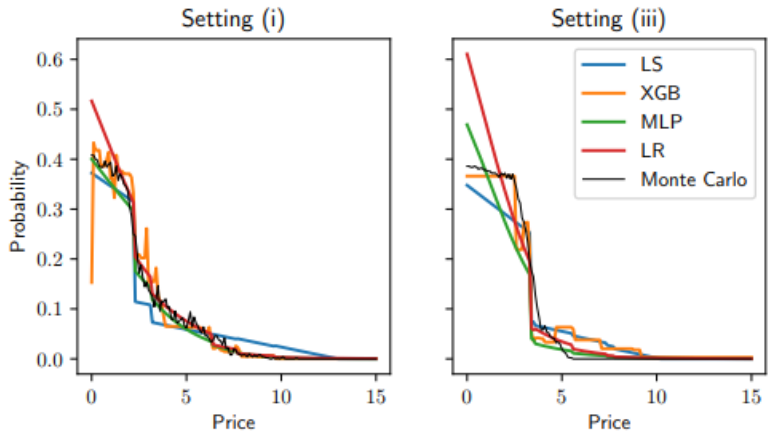
Dennoch ist die Gewichtung zwischen Preisrangierung und den Features unbekannt. Folglich kann die Festlegung dieses Werts einen starken Einfluss auf die Resultate haben. Zusätzlich sind diese Feature Wichtigkeiten in der Realität eher nicht für jeden einzelnen Kunden genau gleich, womit eine Art des Samplings spezifiziert werden müsste.

Kombiniertes Modell

Um die Anzahl der Annahmen zu reduzieren, wird in der Literatur sehr häufig ein kombiniertes Modell verwendet. Dabei wird aus Features wie:

- Agentenpreis
- Agenteneigenschaften
- Konkurrenzpreise
- Konkurrenzeigenschaften
- Historischen Buchungsdaten

direkt die Anzahl der gebuchten Räume beziehungsweise die Wahrscheinlichkeit für die Buchung eines Raumes des Agenten berechnet. Dies wird beispielsweise von Schlosser & Boissier umgesetzt. Dabei wird anhand des Preises eine Wahrscheinlichkeit für die Buchung eines Raumes kalkuliert. Diese Werte wurden mit einer Duopol Simulation berechnet, wobei Annahmen über das Verhalten der Kunden und der Konkurrenzpreise gemacht wurden. Schlussendlich ergab dies für verschiedene Preise die jeweiligen Verkaufswahrscheinlichkeiten.



Quelle: (Schlosser & Boissier, 2018)

Abbildung 5: Verkaufswahrscheinlichkeiten abhängig des Preises bei n=100'000 Simulationen

Ein kombiniertes Modell ermöglicht eine Bestimmung der Anzahl Buchungen, ohne explizit das einzelne Kundenverhalten zu modellieren. Jedoch bedingt diese Modellierung eine grosse Anzahl Annahmen bezüglich der Simulation, welche auf Expertenwissen basieren. (Schlosser & Boissier, 2018) Alternativ kann diese Anzahl Buchungen direkt aus den Daten berechnet werden. Dies ist jedoch sehr schwierig zu erreichen, da für die Konkurrenten keine konkreten Buchungen vorhanden sind und ein kombiniertes Modell sehr komplex ist. Folglich ist dann sowohl die Berechnung und Validierung eines solchen Modells sehr aufwändig und die Interpretierbarkeit der Resultate ist sehr viel schwieriger. Zusätzlich bedingt es, dass in den Daten einen Einfluss der Konkurrenz vorhanden ist, da ansonsten die Möglichkeit besteht, dass ein Monopol modelliert wird.

Methoden

Um die beschriebenen Ideen und Konzepte umzusetzen, wird eine klar definierte Methodik befolgt. Folglich wird in diesem Kapitel der übergreifende **Projektablauf** dargelegt und die Methoden beschrieben, mit welchen die Modelle im Rahmen einer **Evaluation** verglichen werden.

Projektablauf

Beim vorliegenden Projekt handelt es sich um ein Innovationsprojekt, welches einen grossen Teil an Datenanalyse, Datenaufbereitung und Machine Learning beinhaltet. Zu Beginn sind die einzelnen Arbeitsschritte zur Erreichung der **Anforderungen** unklar. Deshalb wird ein exploratives-iteratives Verfahren nach dem Plan-Do-Check-Act Zyklus definiert.

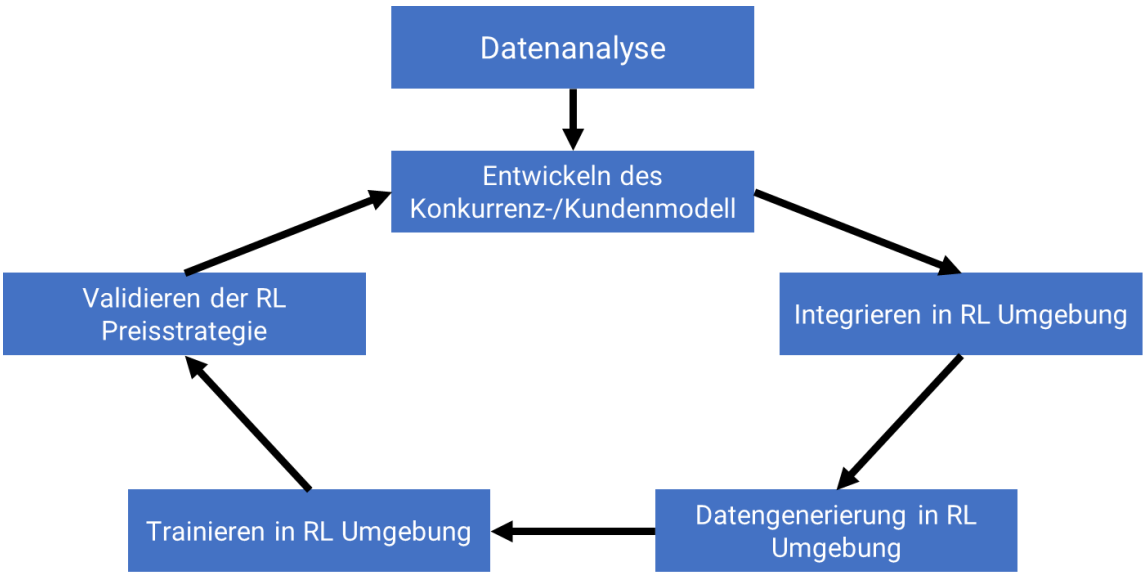


Abbildung 6: Iteratives Projektvorgehensmodell

In einem ersten Schritt werden die vorhandenen Daten analysiert. In den Folgeschritten werden iterativ Modelle entwickelt, trainiert und validiert.

Die Validierung des aktuellen Konzepts wird über bestimmte Metriken (siehe **Evaluation der Konkurrenzpreis- und Nachfragemodelle** und **Evaluation des Reinforcement Learning Algorithmus**), über die qualitative Analyse der Preisstrategie und durch die Besprechung der Ergebnisse mit den Projektmitgliedern sichergestellt. Dabei wird in wöchentlichen Meetings mit dem Projektverantwortlichen der Projektstand laufend geprüft und der Informationsfluss sichergestellt.

Die Zwischenpräsentation in der Mitte des Projektes ermöglicht es zusätzlich, den Projektfortschritt an den Projektverantwortlichen und den Experten mitzuteilen. Dies ermöglicht eine zusätzliche Standortbestimmung und erlaubt eine Neuausrichtung im Projekt.

Die Ergebnisse der Experimente werden im **Anhang F: Study Doc** dokumentiert. Die Risikoanalyse wird als separates Dokument bereitgestellt und der **Rahmenplan** kann dem Anhang entnommen werden.

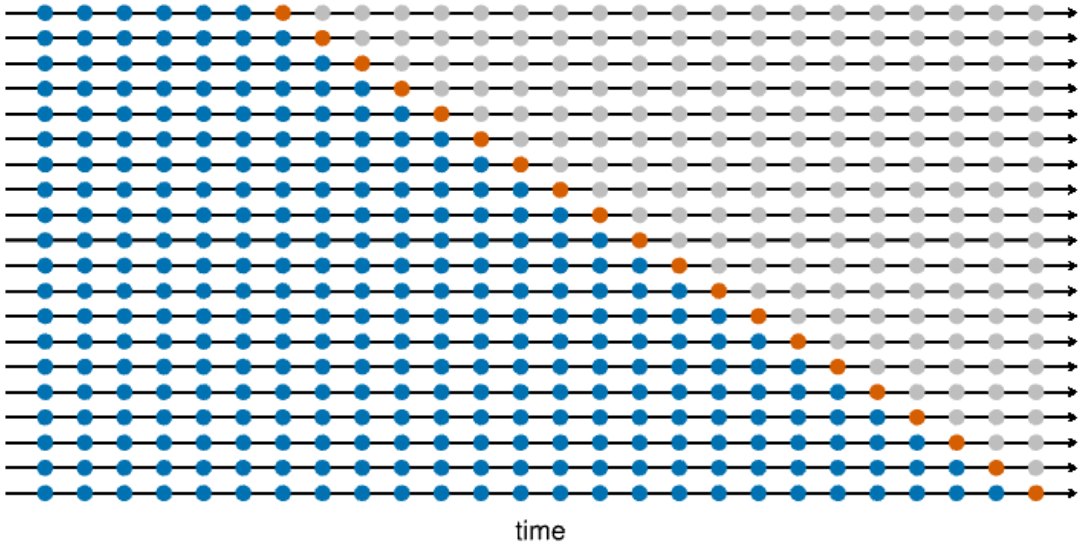
Evaluation

Mit der Evaluation wird die Performance der Modelle während des Projektes sichergestellt. Einerseits können für Konkurrenz- und Nachfragemodelle auf klassische Evaluationsmethoden für das «Supervised Learning» zurückgegriffen werden. Andererseits bedarf es beim Reinforcement Learning Algorithmus andere Verfahren, um die Qualität des Systems zu messen, da keine Ground-Truth-Daten vorhanden sind.

Evaluation der Konkurrenzpreis- und Nachfragemodelle

Da sowohl die Konkurrenzpreise wie auch die Nachfrage eines Hotels zeitabhängig sind, kann das Modell nicht mit einem regulären Train/Validation/Testsplit validiert werden.

Stattdessen wird die Technik der Time Series Cross Validation aus den Zeitreihendaten verwendet. Dabei werden die Splits nicht zufällig definiert, sondern es gibt einen rollenden Datensatz, welches nach Datum sortiert ist, wobei immer wieder der Validationsdatensatz verschoben wird. (Hyndman & Athanasopoulos, 2021) Damit wird erreicht, dass nur immer Daten aus der Vergangenheit für das Training des Modells verwendet werden.



Quelle: Hyndman & Athanasopoulos, 2021

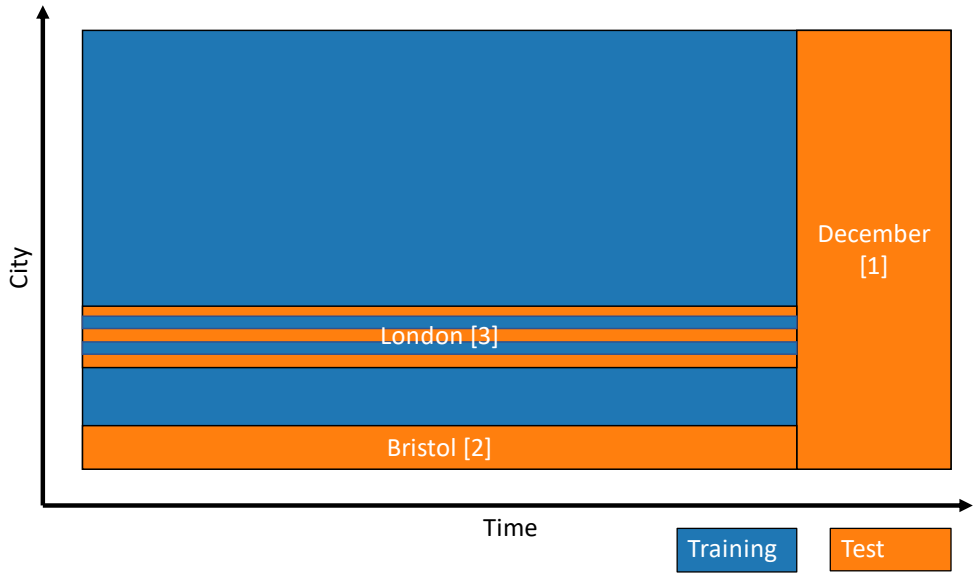
Abbildung 7: Cross-Validation bei Zeitreihendaten

Für jeden einzelnen Split des Validationsdatensatzes werden jeweils die Metriken berechnet. Wenn diese Resultate akzeptiert werden, kann die Performance auf dem Testdatensatz überprüft werden. Diese Testdaten werden während des Trainingsprozesses nicht verwendet.

Für die beiden Modelle Konkurrenzpreise und Nachfrage werden unterschiedliche Trainings- und Test-Datensätze definiert.

Aufteilung Datensatz Konkurrenzpreisemodell

Für das Konkurrenzpreismodell werden die Preisdaten aus dem Jahr 2020 in vier verschiedene Datensätze unterteilt. Diese Datensätze sind disjunkte Mengen. Somit wird eine Messung der Performance auf Daten ausserhalb des Trainingssets ermöglicht.



**Die Fläche der Balken ist nicht repräsentativ und dient einzig der Visualisierung*

Abbildung 8: Aufteilung der Daten für das Konkurrenzpreismodel

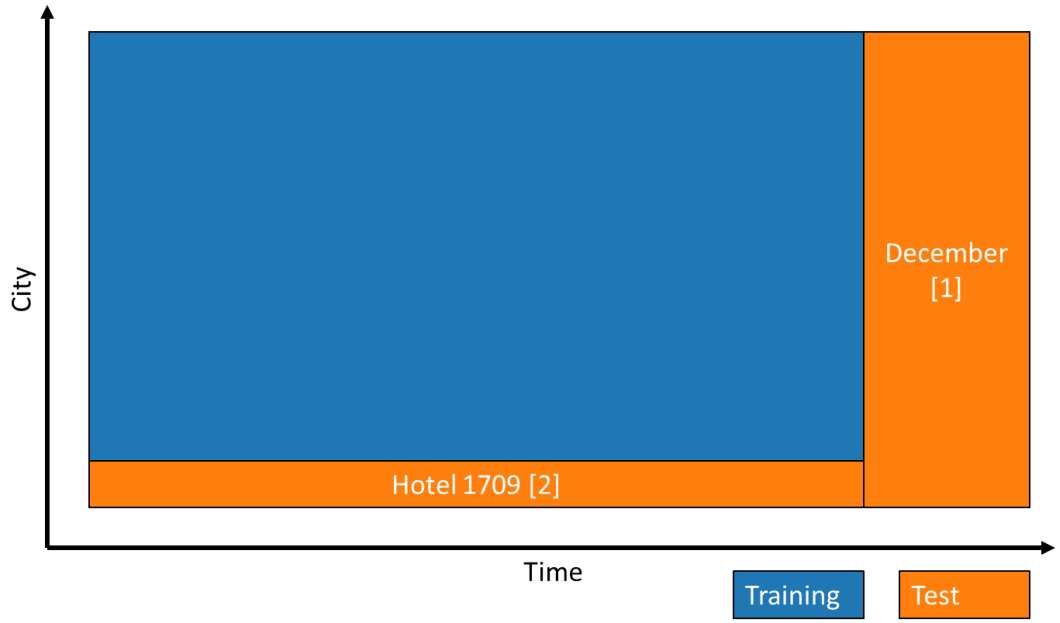
Dabei wird das Modell zuerst mit dem Trainingsset (blau) auf die rollende Performance mit Cross-Validation (siehe **Abbildung 8: Aufteilung der Daten für das Konkurrenzpreismodel**) überprüft. Wenn die Performance auf diesem Set zufriedenstellend ist, kann das auf neuen Daten überprüft werden. Dabei wird die Performance in die Zukunft (1-Dezember), die Generalisierung auf einen anderen Ort (2- Bristol) und die Generalisierung innerhalb eines Ortes (3 – London) überprüft.

ID	Daten	Beschreibung	Zeitraum	Ziel
Training / Validation				
0	Train/Val	Cross-Validation Daten exkl. 1, 2, 3	01.2020 -11.2020	Rollende Performance.
Test				
1	Dezember	Testdaten, aus dem Dezember.	12.2020	Performance in der Zukunft.
2	Bristol	Testdaten von Januar bis November, wo der Ort Bristol eingetragen ist.	01.2020 -11.2020	Performance für anderen Ort
3	London	50%: Der Hotels: Testdaten 50%. Der Hotels: Trainingsdaten	01.2020 -11.2020	Performance: Generalisierung innerhalb eines Ortes

Tabelle 3: Beschreibung der Datenaufteilung für das Konkurrenzmodell

Aufteilung Datensatz Nachfragemodell

Die Daten des Nachfragemodells (Reservationen) werden analog zum Konkurrenzpreismodell aufgeteilt. Für diese Daten werden drei verschiedene Datensätze definiert. Der erste Datensatz dient erneut dem Training und der Cross-Validation (blau). Die beiden anderen Datenmengen werden für die Überprüfung des Modells auf ungesehenen Daten verwendet.



**Die Fläche der Balken ist nicht repräsentativ und dient einzig der Visualisierung*

Abbildung 9: Aufteilung der Daten für das Nachfragemodel

Im konkreten Fall werden die Daten aus dem Dezember und die Daten für ein zufälliges Hotel, in unserem Fall das Hotel mit der ID 1709, verwendet. Dies ermöglicht eine Überprüfung des Modells für einen Zeitraum in der Zukunft und die Generalisierung auf ein anderes Hotel.

ID	Daten	Beschreibung	Zeitraum	Ziel
Training / Validation				
0	Train/Val	Cross-Validation Daten exkl. 1, 2, 3	< 12.2020	Rollende Performance.
Test				
1	Dezember	Testdaten, aus dem Dezember.	12.2020	Performance in der Zukunft.
2	Hotel 1709	Testdaten vor dem Dezember 2020 für das Hotel 1709	< 12.2020	Performance für ein anderes Hotel

Tabelle 4: Beschreibung der Datenaufteilung für das Nachfragemodell

Evaluations Metriken

Für die Evaluation des Konkurrenzpreis- wie auch Nachfragemodelle wird folgende Metrik verwendet:

Abkürzung	Name	Beschreibung
R2	R^2/ Bestimmtheitsmass	Zeigt die durch das Modell erklärbare Varianz [0-1]

Tabelle 5: Evaluations Metrik Konkurrenz- und Nachfragemodelle

Die Wahl der R^2 Metrik basiert darauf, dass es sich beim Konkurrenzpreis-, wie auch beim Nachfragemodell um ein Regressionsproblem handelt. R^2 bietet gegenüber dem Mittleren quadratischen Fehler (MSE) den Vorteil, dass die Werte nie grösser als 1 werden und folglich ein einfacherer Vergleich möglich ist.

Evaluation des Reinforcement Learning Algorithmus

Im Rahmen der Evaluation der Reinforcement Learning Algorithmen muss sichergestellt werden, dass die Resultate von unterschiedlichen RL Algorithmen miteinander vergleichbar sind und die jeweiligen Experimente reproduzierbar werden.

Jedoch gibt es sehr viele Einflussfaktoren, welche die Performance sehr stark verändern. So zeigt Henderson et al. auf, dass zum Beispiel Hyperparameter, die Skalierung der Rewards oder Random Seeds einen sehr starken Einfluss haben. So kann die zufällige Initialisierung der neuronalen Netze durch unterschiedliche Random Seeds signifikant andere Resultate erzeugen. Bei einem Experiment mit dem Standardenvironment «HalfCheetah» kann gezeigt werden, dass beim Durchschnitt von zwei Sets mit jeweils 5 zufälligen Seeds statistisch unterschiedliche Verteilungen entstehen.

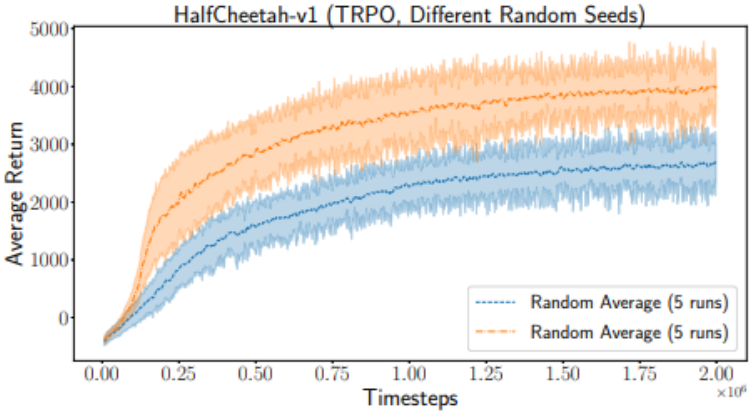


Abbildung 10: TRPO auf dem HalfCheetah-v1 Datensatz mit zwei Gruppen mit gleichen Hyperparametern, aber unterschiedlichen Random Seeds
(Quelle: Henderson et al., 2019)

Dies ist besonders problematisch, wenn nur die Top-N Experimente gezeigt werden oder nur ein Durchschnitt über eine sehr kleine Menge an Versuchen (N < 5) gemacht werden. Folglich sollten für eine gute Aussagekraft die Random Seeds für eine grosse Anzahl Versuche variiert werden.

Evaluations Metriken

In der Praxis werden RL Algorithmen meist ausschliesslich durch die Darstellung des durchschnittlichen Returns beziehungsweise des durchschnittlichen kumulativen Rewards evaluiert. Ebenfalls wird immer häufiger der maximale Reward in einem gewissen Zeitintervall verglichen. Nur den maximalen oder durchschnittlichen Return auszuweisen ist nicht genug, um einen fairen Vergleich zu ermöglichen. Denn diese Metriken können irreführend sein, wenn nicht klar ist, wie die Resultate von den verschiedenen Durchführungen und den unterschiedlichen zufälligen Seeds abhängen. Deshalb wird empfohlen, dass Vertrauensintervalle z.B. 95% dargestellt werden und eine genügend hohe Anzahl an Versuchen durchgeführt wird.

Zusätzlich kann es wichtig sein, dass man die statistische Signifikanz der erreichten Resultate gezeigt werden kann. Im Bereich des RL gibt es jedoch keine Verfahren, die dies ermöglichen. Aus dem Supervised Learning sind Methoden wie der t-Test bekannt. Jedoch treffen die darunterliegenden Annahmen, wie z.B. statistische Unabhängigkeit der Messungen, über die Daten hinweg nicht unbedingt zu. Daher sind weitere Forschungsarbeiten notwendig, um den korrekten Signifikanztest für Reinforcement Learning zu bestimmen.

Folglich sollen wenn möglich die Experimente mit mehreren Random Seeds durchgeführt und dabei die Average Return Kurven verglichen. Ebenfalls werden alle Hyperparameter und die Umgebung definiert und dokumentiert. (Henderson et al., 2019)

Überprüfen der Performance für zwei Zeiträume mit einer Dauer von zwei Wochen
Zusätzlich zur Betrachtung der Reward Kurven im Trainingsprozess wird die Performance auf zwei Zeiträumen von jeweils zwei Wochen validiert. Dazu wird ein Startzeitpunkt ausgewählt und von diesem Zeitpunkt zwei Wochen simuliert. Dabei wird Schritt für Schritt der Zustand der Umgebung, die Aktion des Agenten und der jeweilige Reward aufgezeichnet. Dies ermöglicht eine Evaluation der Performance eines Agenten über diesen Zeitraum und gibt Einblicke wie die konkrete Strategie des Agenten aussieht.

Für diese Evaluation werden nacheinander für den gleichen Startzustand der Umgebung folgende Agenten evaluiert:

- Trainierter Offline RL Algorithmus
- Fixpreis Algorithmus
- Historische Reservationsdaten

Dadurch kann überprüft werden, ob die Strategie des trainierten Offline RL Algorithmus besser als die historischen Daten ist, anhand welcher er trainiert wurde. Zusätzlich wird als Überprüfung die Performance mit einem Algorithmus verglichen, welcher zu jedem Zeitpunkt t einen konstanten Wert $A_t = c$ zurückgibt.

Dabei werden für jede Evaluation folgende Kenngrössen aufgezeichnet:

- Simulationsinformationen: Hotelbeschreibung, Zeitraum
- Alle Modelle: R_t, Summe aller R_t, A_t
- Alle Konkurrenten: Gesetzter Preis, Beschreibung

Dadurch kann eine qualitative Betrachtung der Resultate ermöglicht werden und eine bestimmte Nachvollziehbarkeit der gewählten Strategie überprüft werden.

Realisierung

Um die Evaluation der Modelle zu ermöglichen, wird in einem ersten Schritt eine **Datenanalyse** durchgeführt. Diese dient als Basis für die Entwicklung des **Konkurrenzmodells** wie auch des **Nachfrage- und Kundenmodells.** Diese Modelle werden in einem weiteren Schritt in eine **Reinforcement Learning Umgebung** integriert in welcher unterschiedliche **Agenten** ihre Preisstrategie anwenden können.

Datenanalyse

Um die Grundlage für die Entwicklung der Modelle zu ermöglichen, werden in der Datenanalyse die vorhandenen Datenquellen analysiert und bereinigt.

Verfügbare Daten

Im Rahmen dieses Projektes werden unterschiedlichste Datensätze verwendet. Im folgenden Abschnitt werden diese kurz beschrieben. In den weiteren Kapiteln werden diese Datensätze jeweils referenziert.

Datensatz	**Beschreibung**	**Quelle**
	Konkurrenzdaten	
Konkurrenz	Konkurrenzhotels (Sterne, Location, …) Von N = 35’707 Hotels (UK only)	Booking.com / ABIZ
Konkurrenzpreise	Showpreise der Konkurrenz an einem bestimmten Datum	Booking.com / ABIZ
Beschreibung Konkurrenzhotels	Similaritäten der Konkurrenzbeschreibung zu gewissen Keywords (luxuriös, elegant, …)	Booking.com / ABIZ
	Reservationsdaten	
Reservationsdaten	Reservationsdaten (Datum, Länge, …) inkl. Durchschnittlicher Preis von N = 11 Hotels	RoomPriceGenie / ABIZ
Beschreibung Reservationshotels	Similaritäten der Hotelbeschreibung zu gewissen Keywords (luxuriös, elegant, …)	RoomPriceGenie / ABIZ

Tabelle 6: Verfügbare Datensätze

Der Konkurrenzdatensatz in **Tabelle 6: Verfügbare Datensätze** beinhaltet die Beschreibung von 35’707 Hotels aus dem Vereinigten Königreich. Zusätzlich sind im Konkurrenzpreisdatensatz, welcher ebenfalls von Booking.com stammt, Preise für diese Hotels für jeden dritten Tag verfügbar. In diesem Datensatz sind keine effektiven Buchungen von einzelnen Kunden vorhanden. Jedoch hat das ABIZ Team aus den Hotelbeschreibungen eine Similarität zu gewissen Schlüsselwörtern wie luxuriös, Flughafen und weiteres berechnet. Diese kontinuierlichen Werte können als Eigenschaften des Hotels verwendet werden.

Zusätzlich wird durch Room Price Genie und das ABIZ Team ein Reservationsdatensatz zur Verfügung gestellt. In diesen Daten sind die einzelnen Buchungen der Kunden inklusive Dauer und bezahlter Durchschnittspreis für 11 Hotels ersichtlich. Des Weiteren sind für diese Hotels ebenfalls die Beschreibungen anhand der Eigenschaften verfügbar. Weitere Details können dem **Datenmodell** im Anhang entnommen werden.

Die Datensätze liegen jeweils als CSV-Datei vor und wurden durch das ABIZ Team der HSLU bereits vorbereitet und in einem ersten Schritt bereinigt.

Datensäuberung

Obwohl die Datensätze bereits bereinigt sind, werden mit Absprache des Auftraggebers einige Anpassungen vorgenommen. Dazu werden sowohl die Konkurrenz- als auch die Reservationsdaten analysiert.

Konkurrenz und Konkurrenzpreise

Vor der Bereinigung der Konkurrenzdaten von Booking.com werden folgende Auffälligkeiten festgestellt:

- Konkurrenzpreiseinträge ohne Hotel im Konkurrenzdatensatz
- Pro Tag und Raum gibt es mehrere Preise
- Pro Raum sind nur mindestens alle 3 Tage Preise verfügbar

Diese Artefakte sind durch das Intervall des Crawlers des ABIZ Teams der Booking Website entstanden und werden durch folgende Schritte in Absprache mit Daniel Pfäffli korrigiert:

1. Entfernung aller Preise ohne Hotel im Konkurrenzdatensatz
2. Löschen der doppelten Zeilen und behalten des tiefsten Preises pro Tag und Raum

Es wurde von Fabian Gröger und Daniel Pfäffli entschieden, dass der tiefste Preis pro Tag und Raum behalten wird. Diese Entscheidung basiert darauf, dass wenn der Raum verkauft würde, dann wahrscheinlich der tiefste Preis bezahlt wird.

Zusätzlich werden alle Hotels mit weniger als fünf Preisen entfernt, da hier kaum sinnvolle Vorhersagen gemacht werden können. Diese Zahl 5 wird gewählt, damit sichergestellt werden kann, dass es durchschnittlich pro Quartal im Jahr mindestens einen Preis hat.

In einem letzten Schritt wird der Zeitraum auf das Jahr 2020 reduziert und die Datenpunkte nach Datum sortiert. Diese Reduktion des Zeitraums basiert darauf, dass viele Hotels ausschliesslich Preise im Jahr 2020 bereitstellen.

Reservationsdaten

Bei den Reservationsdaten wird festgestellt, dass zwischen 2 und 30% der Reservationen pro Hotel einen negativen oder einen Preis von 0 hatten. Diese Einträge entstehen durch die Stornierungen der Buchungen. Zusammen mit Daniel Pfäffli wurde entschieden, dass diese Preise ignoriert werden können und in einem weiteren Schritt die Stornierungen als einfacher Prozentsatz eingeführt werden könnten.

Folglich werden diese beiden Datenbereinigungsvorgänge durchgeführt:

- Entfernen aller Reservationen mit einem negativen oder Preis von 0
- Entfernen von irrelevanten Spalten:
    - "hotel_desc" (Beschreibung des Hotels)
    - "daily_rates_array" (Fluktuationen der Preise als Array)
    - "address" (Adresse des Hotels als String)
    - "Unnamed: 0" (Artefakt aus Pandas)

Mit dieser Datenbereinigung kann die Datenmenge reduziert und allfällige Artefakte aus der Datensammlung entfernt werden.

Umsetzung der Konkurrenz- und Nachfragemodelle

Auf Basis dieser bereinigten Datensätzen werden in einem nächsten Schritt die Konkurrenz- und Nachfragemodelle entwickelt.

Für die Konkurrenz- und Nachfragemodelle wird gemäss Absprache mit der Projektleitung ein XGBoost[3] Modell verwendet. Damit werden die vorhandenen Datensätze trainiert und ein Modell evaluiert. Diese Modelle sollen anschliessend in der Reinforcement Learning Umgebung verwendet werden, um die Daten für den Offline RL Algorithmus zu erzeugen.

Die Konkurrenz- und Nachfragemodelle werden bereits im aktuellen Kapitel **Realisierung** evaluiert, da die jeweils fortfolgenden Modelle auf den Ergebnissen der vorangehenden Modelle basieren.

Konkurrenzmodell

Im Konkurrenzmodell soll der Preis der Konkurrenz des Agentens vorhergesagt werden. Dieser Preis soll anhand der Konkurrenz und Konkurrenzpreise Datensätze erlernt werden. Das Ziel ist es, dass eine gewisse Generalisierung erreicht wird und das Modell anhand der Eigenschaften eines Hotels den Preis vorhersagen kann.

Dabei soll das Modell grundsätzlich folgende drei Eigenschaften besitzen:

- Beschreibung des Hotels durch Features basierend auf öffentlich zugänglichen Daten
- Generalisierung der Performance auf einen Zeitraum in der Zukunft
- Generalisierung der Performance auf unbekannte Hotels

Folglich muss aufgrund des ersten Elements sichergestellt werden, dass keine Daten aus der Zukunft oder interne Daten des Hotels verwendet werden. Dies wird mit der Methode des «Time Series Cross-Validation» aus **Evaluation der Konkurrenzpreis- und Nachfragemodelle** erreicht.

Datengrundlage

Für das Konkurrenzmodell werden folgende Datensätze verwendet.

Datensatz	**Relevante Informationen**
Konkurrenz	Information über die Konkurrenzhotels, z.B. Ort, Anzahl Sterne, …
Konkurrenzpreise	Preise der Hotels an einem Datum, welche vorhergesagt werden sollen
Beschreibung der Konkurrenz	Beschreibung des Hotels anhand von Schlüsselwörtern

Tabelle 7: Datengrundlage des Konkurrenzmodels

In **Tabelle 7: Datengrundlage des Konkurrenzmodels** ist ersichtlich, dass die Datensätze von Booking.com, wo Preisinformationen und Beschreibungen von 35’707 Hotels verfügbar sind, verwendet werden.

[3]XGBoost Bibliothek: https://xgboost.readthedocs.io/en/stable/

Bestimmung des besten Modells

Für die Bestimmung der Features und des besten Modells werden unterschiedlichste Kombinationen evaluiert (Details siehe Study-Doc **Konkurrenzmodelle**). Dabei wird eine Maximierung der Performance auf dem Cross-Validation Datensatz angestrebt. Dazu werden drei unterschiedliche XGBoost-Modelle mit unterschiedlichen Features entwickelt, die Hyperparameter auf dem Trainingsset bestimmt und ein Vergleich auf den Cross-Validation-Daten durchgeführt.

Alle drei Modelle weisen folgende Gemeinsamkeiten auf:

Zielgrösse

Preis eines Hotels aus dem Konkurrenzdatensatz an einem gewissen Tag

Features

Folgende Gruppen von Features aus dem Konkurrenz- und Konkurrenzpreisedatensatz werden für alle drei Modelle erzeugt und verwendet:

- Hotelinfos (Ort, Sternebewertung, …)
- Zeit (Quartal, Wochentag, …)
- Preis (Letzter Preis, Preis vor einem Monat, …)

Die genauere Beschreibung dieser Features kann dem Anhang unter **Gemeinsame Features der Konkurrenzmodelle** entnommen werden. Die einzelnen Modelle weisen zusätzliche Features auf, welche eine genauere Beschreibung des Hotels oder mehr Informationen zu den Preisen beinhalten.

Features Modell 1

Das Modell 1 verwendet zusätzlich als One-Hot-Vektoren die Werte aus dem «most_popular_facilites» Array. Das heisst, es werden Informationen über die Infrastruktur wie Parkhaus oder Terrasse miteinbezogen. Im Anhang **Features Modell 1** werden die genauen Features beschrieben.

Features Modell 2

Das Modell 2 verwendet anstatt der «most_popular_facilites» die Similaritäten von den Schlüsselwörtern zu den Beschreibungen der Hotels als Zusatzinformationen. Diese sollen auf eine andere Art die Eigenschaften des Hotels wie luxuriös, elegant, … abbilden. Die genauen Features können dem folgenden Kapitel entnommen werden: **Features Modell 2**

Features Modell 3

Das Modell 3 verwendet zusätzlich zu den Features des Modells 2 den Mittelwert der letzten Preise der Nachbarn im Umkreis von 1km des jeweiligen Hotels. Die genaue Berechnung dieses Features ist im Anhang unter **Features Modell 3** zu finden.

Resultate

Um die Performance der drei Modelle zu validieren, wird wie in **Evaluation der Konkurrenzpreis- und Nachfragemodelle** beschrieben das Bestimmtheitsmass (R^2) verwendet. Dabei werden für den Cross-Validation Datensatz für die Modelle 1-3 folgende Werte erzielt.

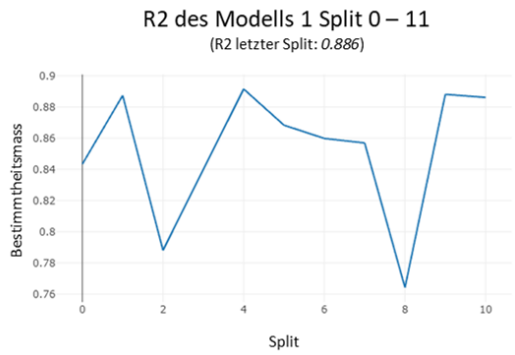


Abbildung 11: Cross-Validation R2 des Nachfragemodells 1

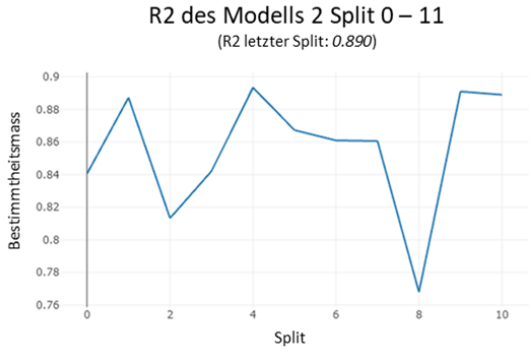


Abbildung 12:Cross-Validation R2 des Nachfragemodells 2

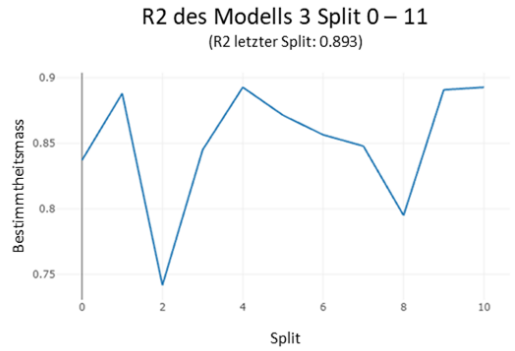


Abbildung 13: Cross-Validation R2 des Nachfragemodells 3

Eine grafische und quantitative Betrachtung zeigt auf, dass alle Modelle eine sehr ähnliche Performance erreichen. Jedoch erreicht das Modell 3 **Abbildung 13: Cross-Validation R2 des Nachfragemodells 3** am Ende den höchsten R^2 Wert (0.893) und weist über die Zeit (Split) eine sich reduzierende Schwankung auf. Zusätzlich bietet dieses Modell durch das zusätzliche Feature die Möglichkeit auf Preise der direkten Nachbarn zu reagieren, was im Kontext des RL Modells eine zusätzliche dynamische Komponente ermöglichen kann.

Wenn man die zehn wichtigsten Features des XGBoost Modells betrachtet, geben diese einen guten Aufschluss darüber, welche Spalten des Datensatzes wichtig für die Vorhersage des Preises sind.

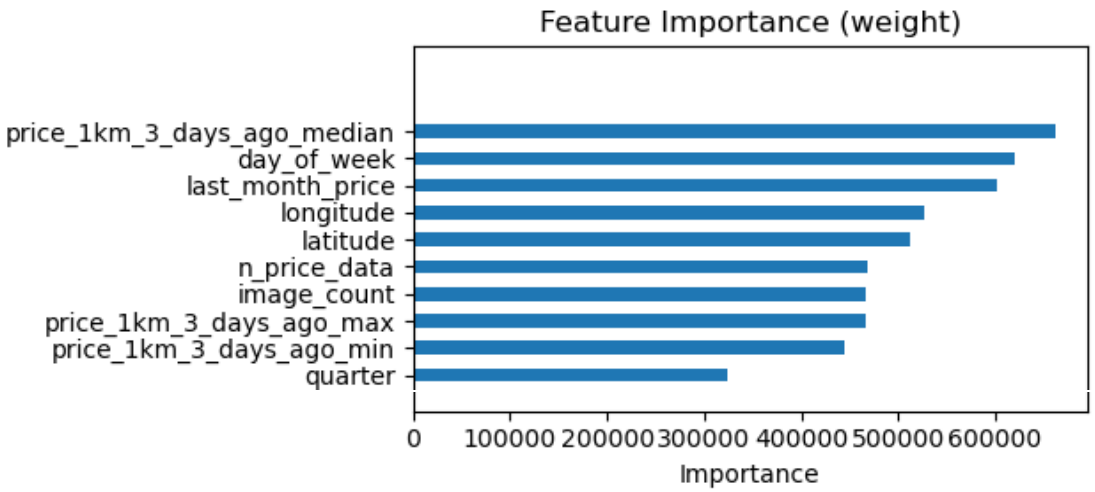


Abbildung 14: Feature Wichtigkeiten des Nachfragemodells 3

In **Abbildung 14: Feature Wichtigkeiten des Nachfragemodells 3** wird ersichtlich, dass der Median Preis der Nachbarn, sowie der Tag der Woche eine essenzielle Rolle für die Bestimmung des Preises spielen.

Das Modell 3 wurde aufgrund des erreichten R^2 Wertes und der Feature Wichtigkeiten vom Projektverantwortlichen Daniel Pfäffli am 12.04.2022 akzeptiert.

Testdatensatz

Folglich kann dieses Modell auf den Testdaten evaluiert werden. Das Modell 3 erreicht auf den unterschiedlichen Testdatensätze folgende Resultate:

Datensatz	**Bestimmtheitsmass R^2**
Dezember	0.718
Bristol	0.811
London	0.62

Tabelle 8: Resultate des Konkurrenzpreismodells 3 auf den Testdaten

In **Tabelle 8: Resultate des Konkurrenzpreismodells 3 auf den Testdaten** wird eine erste Tendenz sichtbar, dass eine sehr gute Generalisierung auf den anderen Ort Bristol erreicht wird. Ebenfalls werden für die Preise im Dezember ein R^2 von 0.718 erreicht. Für die Hotels in London kann das Modell nur mit einiges tieferer Zuverlässigkeit die Preise vorhersagen.

Um diese Resultate für einzelne Hotels zu überprüfen, wird pro Testdatensatz für zwei zufällige Hotels der Preis vorhergesagt. Wenn man dieses Verfahren für zwei Hotels aus dem Bristol Datensatz anwendet und für die verfügbaren Tage jeweils den ersten Preis vorhersagt, erhält man folgendes Resultat.

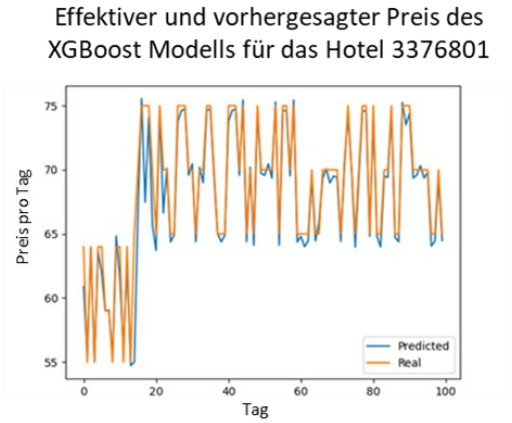


Abbildung 15: 100 erste vorhergesagte Preise des Modells für das Hotel 3376801 in Bristol

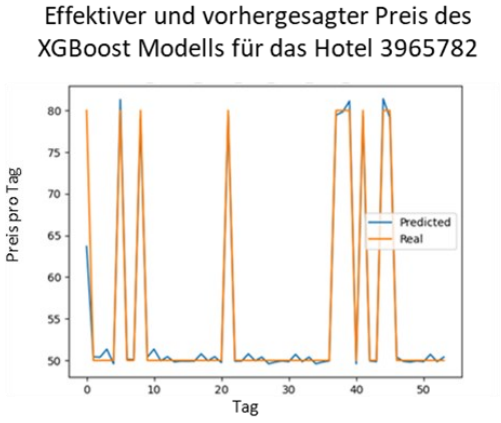


Abbildung 16: 54 erste vorhergesagte Preise des Modells für das Hotel 3965782 in Bristol

In der **Abbildung 15: 100 erste vorhergesagte Preise des Modells für das Hotel 3376801 in Bristol** und **Abbildung 16: 54 erste vorhergesagte Preise des Modells für das Hotel 3965782 in Bristol** kann bei rein qualitativer Betrachtung festgestellt werden, dass der Preis mit hoher Zuverlässigkeit vorhergesagt werden kann.

Datensatz	**Bestimmtheitsmass R^2**
Hotel 3376801 / erster Preis pro Tag	0.91
Hotel 3965782 / erster Preis pro Tag	0.961

Tabelle 9: Performance der Hotels 3376801 und 3965782 des Testdatensatzes Bristol

Dies bestätigt sich in **Tabelle 9: Performance der Hotels 3376801 und 3965782 des Testdatensatzes Bristol** wo die R^2 Metrik gute Resultate zeigt.

Analog wird eine solche Auswertung auch für zwei Hotels aus dem Dezember und London Testdatensatz durchgeführt. Die Performance ist dabei auf den Dezemberdaten gut und auf den London Daten leicht schwächer. Genauere Resultate können dem Anhang unter **Resultate auf dem Testdatensatz Konkurrenzmodell** betrachtet werden.

Validierung

Das Modell zeigt eine gute Generalisierung auf einen anderen Ort (Bristol) und eine relativ gute Generalisierung auf einen anderen Zeitraum (Dezember). Innerhalb des Ortes Londons sind die Resultate weniger vielversprechend. Dies ist aber weniger relevant, da hauptsächlich Hotels aus Bristol betrachtet werden.

Folglich wird das Modell auf den gesamten verfügbaren Daten trainiert und es kann so auf den Trainigsdaten ein R^2 von 0.964 erreicht werden. Die wichtigsten 5 Features (Wichtigkeit), die das Modell erkannt hat, sind die folgenden:

1. price_1km_3_days_ago_median
2. price_1km_3_days_ago_max
3. price_1km_3_days_ago_min
4. day_of_week
5. last_month_price

Diese Resultate zeigen auf, dass die Preisfeatures eine sehr hohe Wichtigkeit haben. Ebenfalls ist der Wochentag ein relevantes Feature, um den Preis zu bestimmen. Um den Einfluss dieser Features zu messen werden einzelne Datenpunkte analysiert.

In einem ersten Schritt wird sich dabei auf den Einfluss der Konkurrenz fokussiert. Dies wird umgesetzt, indem für fünf zufällige Datenpunkte alles bis auf den Preis der Nachbarn ('price_1km_3_days_ago_median') fixiert wird und dann Vorhersagen für den Preis gemacht werden. Dadurch kann untersucht werden, wie sich der eigene Preis verändert, wenn die Konkurrenz den Preis erhöht oder senkt.

Aus ökonomischer Sichtweise gäbe es als Hotels mehrere Möglichkeiten, wie man sich verhalten kann, wenn die Nachbarn die Preise erhöhen, entweder erhöht man mit der Konkurrenz der Preis oder senkt den Preis gegenläufig. Um diesen Effekt zu überprüfen wird der Preis der Nachbarn jeweils zwischen 0.5 und 2-fachen des effektiven Durchschnittspreises variiert. Anschliessend wird die Korrelation zwischen dem Median Preis der Nachbarn und dem vorhergesagten Preis des Modells berechnet.

Resultate

Die fünf zufälligen gewählten Hotels reagieren im Konkurrenzmodell bei verändertem Konkurrenz- bzw. Nachbarpreis (X-Achse) folgendermassen.

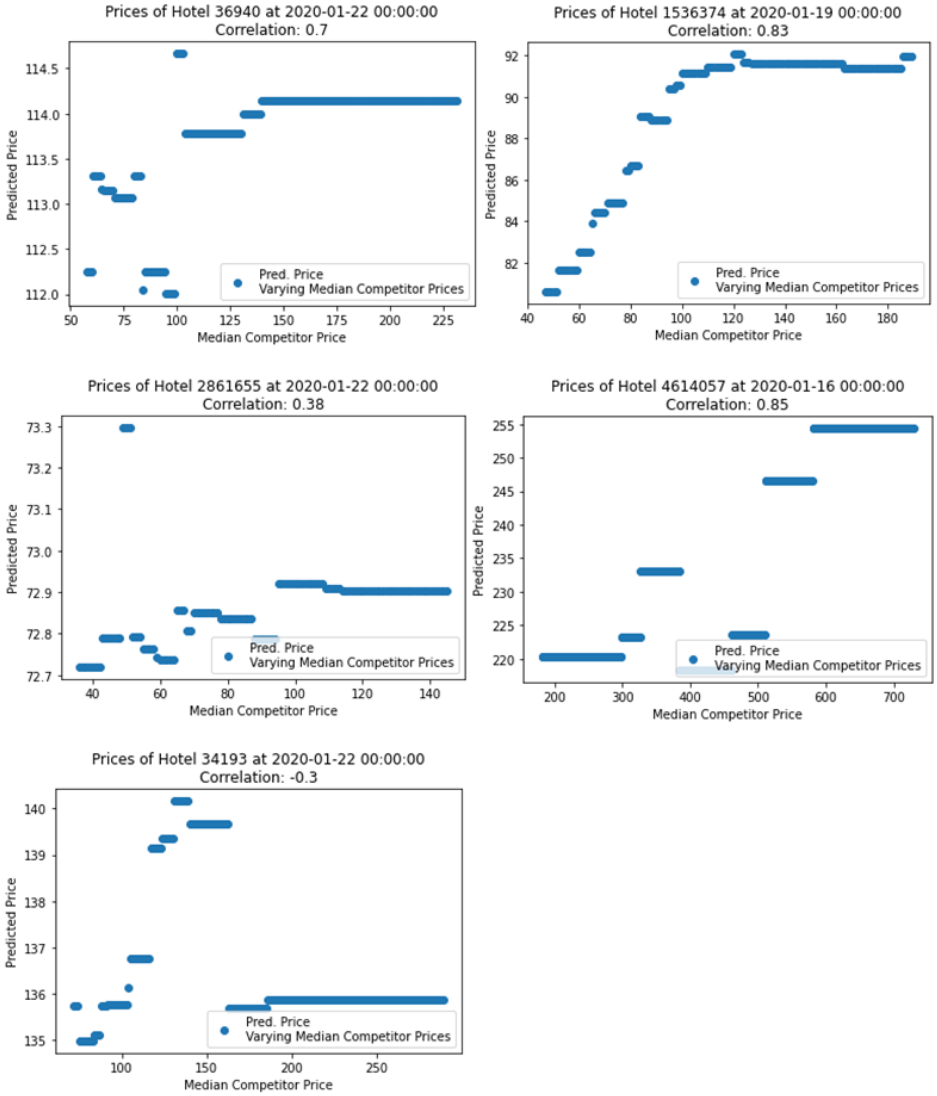


Abbildung 17: Abhängigkeiten des vorhergesagten Preises vom Median Preis der Nachbarn von 5 zufälligen Hotels

Wie erwartet erhöhen sehr viele Hotels die Preise, wenn die Konkurrenz die Preise erhöht (siehe **Abbildung 17: Abhängigkeiten des vorhergesagten Preises vom Median Preis der Nachbarn von 5 zufälligen Hotels**). Jedoch gibt es auch Hotels, die initial den Preis erhöhen und dann bei zu hohen Konkurrenzpreisen den eigenen Preis wieder senken. Jedoch ist klar sichtbar, dass ab einer gewissen Preishöhe der eigene Preis unterhalb des Konkurrenzpreises verbleibt.

Fazit

Das Konkurrenzmodell ermöglicht es, verschiedene Konkurrenzpreise für die RL Umgebung zu berechnen und zeigt dabei eine zuverlässige Performance. Die vorhergesagten Preise sind abhängig vom Wochentag und dem eigenen Preis in der Vergangenheit. Ebenfalls hat das Modell eine gewisse Tendenz erlernt, dass man mit der eigenen Nachbarhotels den Preis erhöhen kann. Dabei werden die Preise aber weniger stark angehoben und die vorhergesagten Preise liegen oftmals unter den Preisen der Nachbarhotels.

Nachfrage- und Kundenmodell

Zusätzlich zur Modellierung der Konkurrenz wird auch ein Nachfrage- bzw. Kundemodell entwickelt. Das Nachfragemodell dient dazu für ein bestimmtes Hotel oder eine Gruppe von Hotels die Nachfrage zu generieren. Das Kundenmodell hingegen bestimmt, welches Hotel schlussendlich gebucht wird. Ebenfalls besteht die Möglichkeit, ein kombiniertes Modell zu entwickeln, welches direkt die Buchungen zurückgibt.

Im Rahmen dieser Arbeit werden fünf verschiedene Nachfrage- und Kundenmodellkombinationen entwickelt. Diese basieren jeweils auf den Erkenntnissen der Experimente der vorangehenden Modelle. Folglich befinden sich im folgenden Kapitel sowohl Umsetzung als auch die daraus folgenden Resultate. Zusätzlich wird für detaillierte Resultate auf den **Anhang F: Study Doc** verwiesen.

Datengrundlage

Für die Nachfrage- und Kundenmodelle werden folgende Datensätze verwendet.

Datensätze	**Relevante Informationen**
Reservationen	Buchungspreis, Anzahl Buchungen am Tag, Buchungsdatum
Beschreibung der Reservationshotels	Beschreibung des Hotels anhand von Schlüsselwörtern
Konkurrenz	Information über die Konkurrenzhotels, z.B. Ort
Konkurrenzpreise	Preise der Hotels an einem Datum, welche vorhergesagt werden sollen
Beschreibung der Konkurrenz	Beschreibung des Hotels anhand von Schlüsselwörtern

Tabelle 10: Datengrundlage des Nachfrage- und Kundenmodells

Grundhypothese

Die Grundhypothese anhand welcher ein Grossteil dieser Modelle entwickelt werden, basiert darauf, dass anhand der Reservationen der 11 Hotels aus dem Reservationsdatensatz ein generelles Nachfrage- und Kundenmodell berechnet werden kann.

Dazu wird jeweils als Zielgrösse die Summe der Gäste an einem bestimmten Tag in einem spezifischen Hotel vorhergesagt.

Modelle

Im Rahmen der vorliegenden Arbeit werden verschiedene Modellkombinationen entwickelt. Das **Nachfragemodell** bestimmt dabei die Anzahl Kunden, welche sich für bestimmte Hotels interessieren. Daraus berechnet das **Kundenmodell** die Anzahl Buchungen, die das Agentenhotel erhält.

ID	Nachfragemodell	Kundenmodell
1	Feature-basiertes XGBoost Modell	-
2	Konstante Anzahl Kunden (N=3)	Modellierung dreier Kundenarten mit verschiedenen Entscheidungsfunktionen
3	Regressionsbasiertes Modell	Tiefster Preis
4	Regressionsbasiertes Modell mit 2 Kundengruppen	Tiefster Preis
5	Feature-basiertes XGBoost Modell mit Einbezug der Konkurrenz	

Tabelle 11: Übersicht Kombinationen Kunden- und Nachfragemodell

Es ist zu beachten, dass bei der Modellkombination mit ID=1 aufgrund der Erkenntnisse aus dem Nachfragemodell kein Kundenmodell entwickelt wird.

Feature-basiertes XGBoost Modell

Das Feature-basierte XGBoost Modell ist ein baumbasiertes Machine Learning Model, welches folgenden Wert vorhersagen will:

> *Anzahl Personen, welche an einem bestimmten Tag ein bestimmtes Hotel buchen wollen*

Diese Vorhersage soll dabei aufgrund folgender Features berechnet werden:

- Preis der Übernachtung
- Zeit (Wochentag, Quartal, …)
- Eigenschaften des Hotels (luxuriös, …)

Sowohl der Median Preis einer Übernachtung wie auch der Zeitpunkt wird aus den Reservationsdaten entnommen. Die Eigenschaften des Hotels werden als One-Hot Vektoren aus den Beschreibungen des ABIZ-Teams verwendet.

Resultate

Das Modell wird gemäss **Aufteilung Datensatz Nachfragemodell** auf dem Cross-Validation-Set trainiert und validiert. Der R2-Wert erreicht dabei im 11. Split einen Wert von 0.856, was für eine gute Performance spricht.

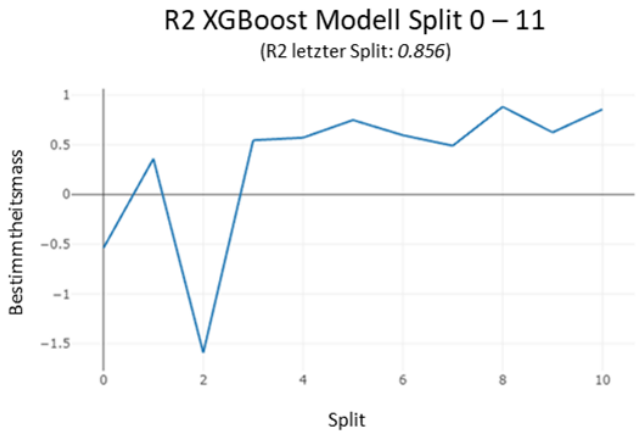


Abbildung 18: R2 Werte des XGBoost Nachfrage Modells zwischen Split 0 und 11

Eine Analyse der Feature-Wichtigkeiten zeigt auf, dass der Median des Preises an einem Tag und die Zeitinformationen wie Tag der Woche, Monat und Quartal eine hohe Wichtigkeit für die Vorhersage der Nachfrage haben.

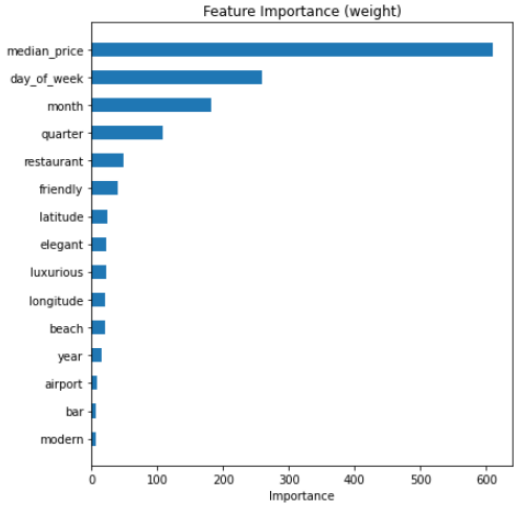


Abbildung 19: Feature Wichtigkeiten des XGBoost Nachfragemodells

Zusammen mit Daniel Pfäffli wurde dieses Modell akzeptiert. Es erreicht einen guten R^2 Wert auf dem Cross-Validation-Set und die Wichtigkeit der jeweiligen Features weisen aus manueller Betrachtung auf eine sinnvolle Gewichtung der Features hin. Bei einer Evaluation auf dem Testdatensatz bestätigen sich diese Resultate.

Datensatz	**Bestimmtheitsmass R^2**
Dezember	0.756
1709	0.953

Tabelle 12: Resultate des XGBoost Nachfragemodells auf den Testdaten

Es gibt eine gute Generalisierung auf das Hotel 1709 und akzeptable Resultate auf den Dezemberdaten. Auch eine Betrachtung der ersten 100 Tage für das Hotel 1709 zeigen, dass die Nachfrage mit hoher Genauigkeit vorhergesagt werden kann.

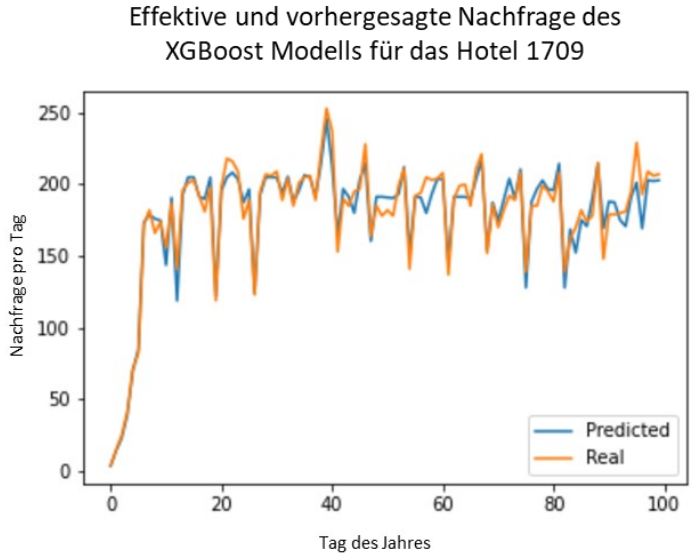


Tabelle 13: Effektive und vorhergesagte Nachfrage des XGBoost Modells für das Hotel 1709

Wie in **Abbildung 19: Feature Wichtigkeiten des XGBoost Nachfragemodells** ersichtlich hat der Median-Preis einen signifikant hohen Einfluss auf die Resultate der Nachfrage. Gemäss den Grundregeln der Ökonomie sollte bei regulären Gütern eine Erhöhung des Preises zu einer Reduktion der Nachfrage führen. Diese Hypothese wird überprüft, indem bei 5 zufälligen Datenpunkten aus den Reservationsdaten alle Variablen mit Ausnahme des Median Preises fixiert werden. Folglich wird de Median Preis variiert und der Einfluss auf die vorhergesagte Anzahl Personen betrachtet.

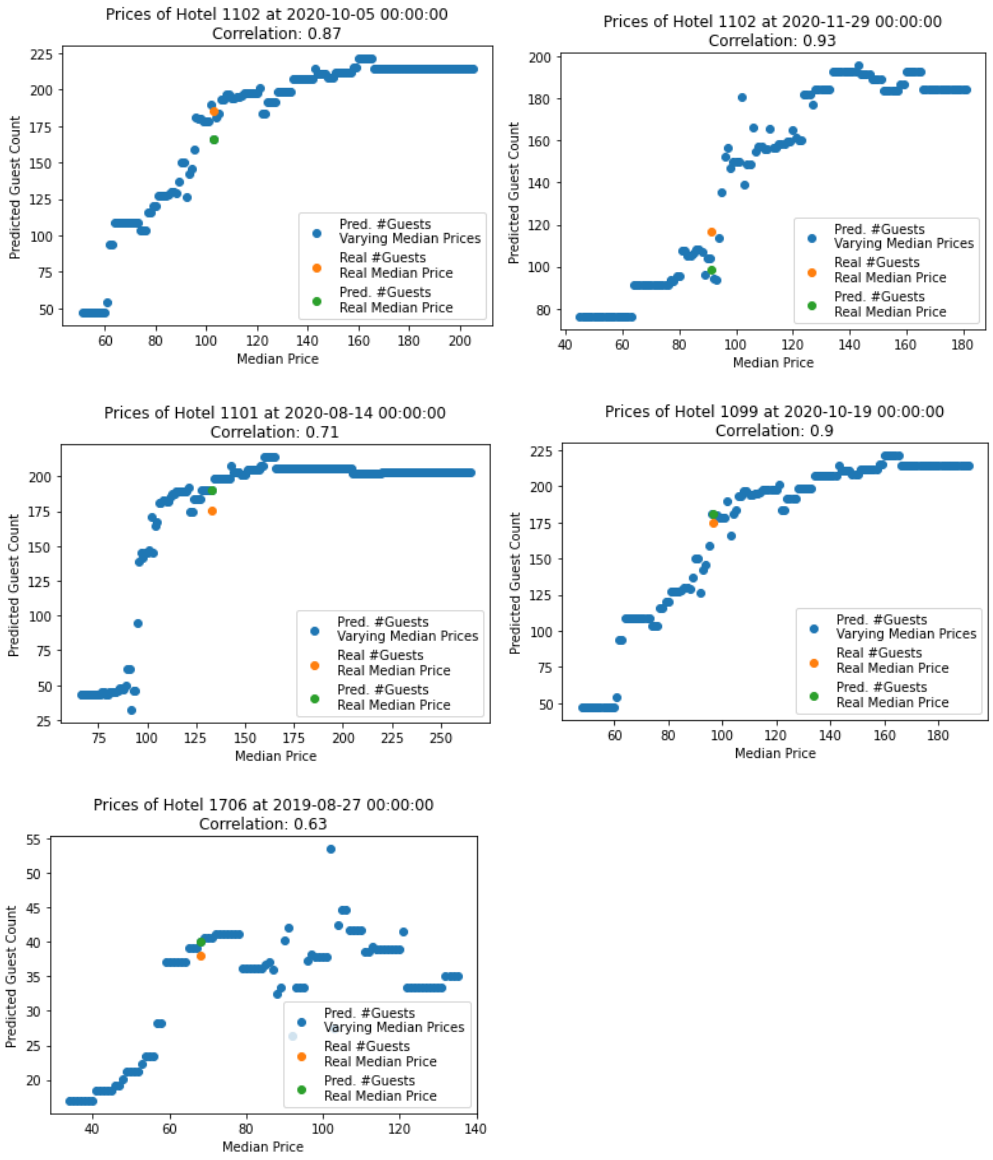


Abbildung 20: Veränderung der vorhergesagten Nachfrage abhängig von der Variation des Median Preises

Diese Hypothese lässt sich nicht bestätigen, das heisst in **Abbildung 20: Veränderung der vorhergesagten Nachfrage abhängig von der Variation des Median Preises** wird sichtbar, dass eine Erhöhung des Preises zu grossen Teilen zu einer Erhöhung der Nachfrage führt.

Die genaueren Resultate aus dem ersten Versuch ohne den Median Preis und Details zu den eben beschriebenen Resultaten inklusive Hyperparameteroptimierung können dem Study Doc unter **Feature-basiertes XGBoost Modell** entnommen werden.

Fazit

Das Modell scheint eine kontraintuitive Beziehung zwischen Preis und Nachfrage gelernt zu haben. Dies wäre höchstens mit dem Veblen-Effekt zu erklären, welcher aussagt, dass ein steigender Preis zu einer höheren Nachfrage führt. Dies tritt sehr häufig bei Luxusgegenständen auf. (Michalos, 2014) Jedoch sprechen im konkreten Fall von Hotels empirische Studien dagegen. Corgel et al. stellt mit Hilfe der Berechnung von Preiselastizitäten fest, dass bei Hotels eine negative Beziehung zwischen dem Preis und der Nachfrage besteht. (Corgel et al., 2012) Deshalb würde ein Einsatz dieses Nachfragemodells im Rahmen der Reinforcement Learning zu kontraintuitiven Resultaten führen.

Zusätzlich gibt es folgende Schwierigkeiten, welche beim vorliegenden Modell nicht gelöst sind:

- Kein expliziter Einbezug der Konkurrenz
- Berechnung der effektiven Buchungen aus der Nachfrage

Folglich ist dieses Modell als Nachfragemodell nicht sinnvoll einsetzbar. Der Grund für diese Ergebnisse müssen genauer untersucht werden. Folgende Vermutungen bestehen bereits:

- Die Daten repräsentieren keine Nachfrage
- Im Sommer, wenn die Nachfrage hoch ist, sind auch die Preise höher
- Das Modell generalisiert aufgrund der kleinen Anzahl Hotels nicht

Basierend aus diesen Resultaten soll ein einfacheres Modell entwickelt werden, welches die Kunden mit einbezieht.

Konstantes Kundemodell

Das konstante Kundenmodell entfernt sich von einer datenbasierten Nachfrage und versucht anhand von manuellen Regeln eine Nachfrage und ein Kundenmodell zu entwickeln. Dies hat das Ziel, dass die Resultate des Reinforcement Learning Modells nachvollziehbarer werden, eine Preiselastizität besteht und der Kunde anhand der Eigenschaften der Hotels seine Buchungen macht.

Dazu wird folgendes Modell entwickelt:

- Nachfrage: Konstant 3 Personen
- Kunden: 1 Student, 1 Tourist und 1 Businesskunde

Die drei Kunden (Student, Tourist und der Businesskunde) wählen anhand ihrer festdefinierten Entscheidungslogik (Preis und Features) das Hotel (Agent und 3 Konkurrenten) aus.

Kundentyp	**Preis**	**Gewichtung Preis**	**Features**	**Gewichtung Features**
Student	Tiefster zuerst	100%	-	0%
Tourist	Tiefster zuerst, Preis zwischen 50 und 100	50%	Leisure, Swimming Pool, …	50%
Business	Höchster zuerst	50%	Business, Desk, …	50%

Tabelle 14: Gewichtung zwischen Preis und Features der Kunden des konstanten Kundenmodells

Pro Tag gibt es folglich drei verschiedene Kunden, welche sich zwischen den einzelnen Angeboten entscheiden müssen. Dabei haben die Kundegruppen verschiedene Ziele, wie beim Studenten einen tiefen Preis , einen mittleren Preis mit Eigenschaften eines Freizeithotels beim Touristen und ein luxuriösen Hotels mit hohen Preisen beim Businesskunden.

Resultate

Dieses Modell wird in **Reinforcement Learning mit konstantem Nachfragemodell** validiert. Dabei wird ein grosser Einfluss dieses Modells auf die Resultate festgestellt.

Fazit

Das konstante Kundenmodell bringt einige Vorteile. So erlaubt es beispielsweise verschiedene Preissensitivitäten zu modellieren. Ebenfalls ermöglicht es, ein Kundemodell zu definieren, welches eine Entscheidung nicht nur abhängig vom Preis, sondern auch von den Eigenschaften eines Hotels macht. Weiter ermöglicht das vorliegende Modell eine gute Interpretierbarkeit der Resultate.

Dies ist jedoch wiederum auch der grösste Nachteil: Die Wahl der Features und die Definition der Gewichtungen beeinflusst die Resultate sehr stark. Ebenfalls ist die Nachfrage von konstant drei Kunden nicht realitätsgetreu.

Durch diese Einschränkungen ermöglicht dieses Modell zwar einen ersten Versuch in einem RL-Setup, jedoch ist die Modellierung nicht datenbasiert und ermöglicht deshalb keine Generalisierung der Resultate. Folglich soll die Nachfrage mit einem Regressionsmodell analog zum «**Feature-basiertes XGBoost Modell**» für ein bestimmtes Hotel aufgrund der Daten generiert werden.

Regressionsbasiertes Kundenmodell

Das regressionsbasierte Kundenmodell hat das Ziel, dass es die folgenden Nachteile der vorherigen Modelle **Feature-basiertes XGBoost Modell** und **Konstantes Kundemodell** ausmerzt:

- Fixe Nachfrage
- Höherer Preis führt zu höherer Nachfrage
- Schlechte Generalisierung

Folglich wird ein einfaches Modell entwickelt:

- Nachfrage: Abhängig von Preis und Tag
- Kunden: Wählen des tiefsten Preises

Dabei wird sowohl ein lineares, als auch ein quadratisches Regressionsmodell trainiert und verglichen. Dies wird pro Gruppierung nach Wochentag und Sommersaison (Mai-August) oder der restlichen Saison durchgeführt. Diese Wahl wurde zusammen mit dem Projektteam bestimmt. In einem weiteren Schritt könnten andere Gruppierungen betrachtet und evaluiert werden.

Folglich gibt es 14 verschiedene Modelle, die berechnet werden. Dazu werden alle Daten aus dem Reservationsdatensatz verwendet. Dieses Vorgehen wird aufgrund der Erkenntnisse aus **Feature-basiertes XGBoost Modell** gewählt, damit eine höhere Datenmenge vorhanden ist. Die Daten werden anschliessend pro Tag und Hotel gruppiert und die Anzahl Personen und der tiefste bezahlte Preis an diesem Tag entnommen.

Resultate

Eine beispielhafte Nachfragekurve für den Sommer am Donnerstag inklusive der einzelnen Datenpunkte kann in der **Abbildung 21: Nachfrage in Abhängigkeit der Preise des Regressionsmodells für den Donnerstag im Sommer** betrachtet werden.

Dabei ist zu beachten, dass in allen folgenden Nachfragediagrammen die mathematische Darstellung mit dem Preis auf der x-Achse und nicht die ökonomische Darstellung mit der Nachfrage auf der x-Achse gewählt wird.

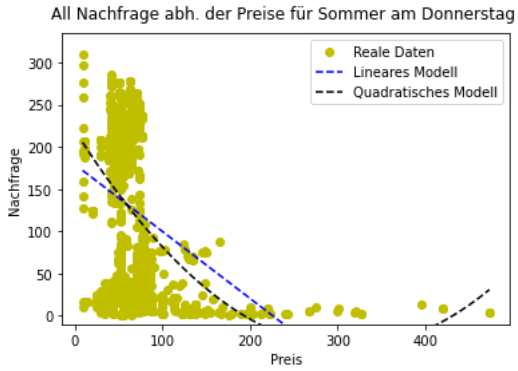


Abbildung 21: Nachfrage in Abhängigkeit der Preise des Regressionsmodells für den Donnerstag im Sommer

Auf den Trainingsdaten werden fortfolgend der R^2 Wert und die Preiselastizitäten berechnet. Wenn man den Mittelwert über alle Modelle Tage und Saisons betrachtet, wird ersichtlich, dass aufgrund der simplen Modelle ein niedriger R^2 Wert erreicht wird.

Modell	**Mean MSE (Std)**	**Mean R2 (Std)**
Linear	7593.60 (1178.92)	0.116 (0.026)
Quadratisch	7460.64 (1158.705)	0.131 (0.033)

Tabelle 15: Resultate des linearen und quadratischen Regressionsmodells auf dem Trainingsset

Die **Tabelle 15: Resultate des linearen und quadratischen Regressionsmodells auf dem Trainingsset** zeigt auf**,** dass das lineare Modell leicht schlechtere Resultate erzielt. Jedoch sind die Differenzen zum quadratischen Modell sehr gering und die Interpretierbarkeit ist viel einfacher sichergestellt. Deswegen wird entschieden, dass das lineare Modell verwendet wird.

Die in **Abbildung 21: Nachfrage in Abhängigkeit der Preise des Regressionsmodells für den Donnerstag im Sommer** sichtbare negative Beziehung zwischen dem Preis und der Nachfrage ist für alle Teilmodelle zu beobachten.

Wenn man die Preissensitivitäten Sommer, *Nicht Sommer* der Nachfrage des linearen Nachfragemodells betrachtet, erhält man folgende Werte:

Erhöhung Preis Von -> Nach	**Montag**	**Dienstag**	**Mittwoch**	**Donnerstag**	**Freitag**	**Samstag**	**Sonntag**
50 -> 60 CHF	-0.319 *-0.263*	-0.317 *-0.277*	-0.34 *-0.308*	-0.284 *-0.264*	-0.203 *-0.170*	-0.192 *-0.160*	-0.247 *-0.192*
100 -> 120 CHF	-0.938 *-0.714*	-0.926 *-0.768*	-1.041 *-0.892*	-0.794 *-0.717*	-0.508 *-0.411*	-0.476 *-0.380*	-0.657 *-0.475*
150 -> 180 CHF	-2.647 *-1.666*	-2.588 *-1.869*	-3.256 *-2.416*	-1.977 *-1.675*	-1.023 *-0.775*	-0.937 *-0.703*	-1.469 *-0.935*

Tabelle 16: Preiselastizitäten des regressionsbasierten Nachfragemodells

Bei einer Preiserhöhung um 20% bei einem Preis von 50 Franken ist eine Preiselastizität von -0.170 und -0.34 zu beobachten. Je höher der Preis desto höher ist auch die Preiselastizität. Dabei ist sehr spannend zu sehen, dass die Nachfrage im Sommer elastischer ist. Dies ist sehr wahrscheinlich dem Umstand geschuldet, dass im Sommer die Leute zum Teil nicht buchen, wenn die Preise zu teuer werden. Eine mögliche Erklärung der tieferen Preiselastizität im Winter könnte der höhere Anteil an notwendigen Reisen sein.

Ebenfalls sind die Preise gegen Ende der Woche weniger elastisch. Dies könnte darauf hinweisen, dass am Wochenende sehr wahrscheinlich die Touristen unabhängiger vom Preis einen Ort besuchen wollen und das Hotelzimmer trotzt erhöhter Preise buchen werden.

Fazit

Grundsätzlich stimmen diese Elastizitäten ungefähr mit den empirischen Werten von Corgel et. al überein, wo die Werte zwischen -0.08 und -1.36 liegen. Ebenfalls bestätigt sich in den vorliegenden Daten, dass es bei höherpreisigen Hotels eine höhere Elastizität herrscht. (Corgel et al., 2012) Zusätzlich erlaubt dieses Modell eine sehr einfache generalisierte Nachfrage zu erzeugen.

Dennoch hat auch dieses Modell einige Einschränkungen:

- Tiefe R^2 und hohe MSE Werte
- Kein Einbezug des Kundenverhaltens (nur eine Kurve)
- Schwieriger Einbezug der Konkurrenz

Regressionsbasiertes Kundenmodell mit Kundengruppen

Dieses Kundenmodell basiert auf dem Vorgehen des «**Regressionsbasiertes Kundenmodell**» und versucht zusätzlich zwei Kundengruppen zu separieren. Dadurch kann pro Tag und Saison unterschiedliche Preiselastizitäten im Nachfragemodell ermöglicht werden.

Aus den Erkenntnissen des ABIZ Team der Hochschule Luzern (siehe **Datenbasierte Definition der Eigenschaften**) wird ersichtlich, dass es grosse Unterschiede zwischen den Geschäftskunden (ausschliesslich unter der Woche) und den Freizeitkunden gibt.

Auf Basis dieser Erkenntnis werden alle Reservationen in zwei Gruppen unterteilt:

- Gruppe Business: Keine der Buchungstage am Wochenende
- Gruppe Tourist: Mind. einer der Buchungstage am Wochenende

Anhand dieser Aufteilung werden erneut jeweils pro Wochentag und Sommer/*Nicht Sommer* sowohl ein lineares als auch ein quadratisches Regressionsmodell trainiert und evaluiert.

Resultate

Durch diese Unterscheidung der zwei Kundengruppen gibt es für den Donnerstag im Sommer sowohl Geschäfts-, wie auch Freizeitkunden, welche durch unterschiedliche Nachfragekurven repräsentiert werden.

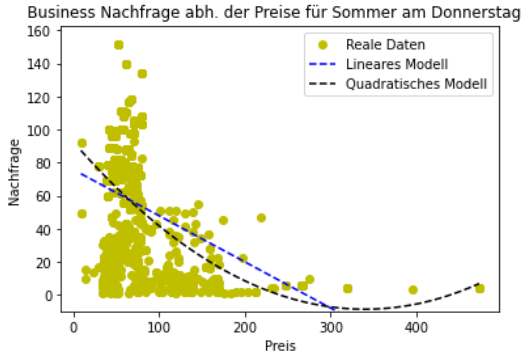


Abbildung 22: Geschäftskundenachfrage am Donnerstag

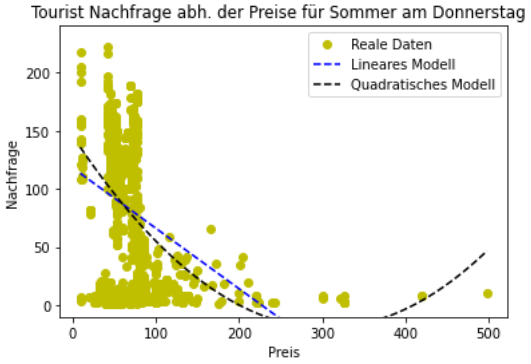


Abbildung 23: Touristennachfrage am Donnerstag

In **Abbildung 22: Geschäftskundenachfrage am Donnerstag** zeigt sich, dass die Geschäftskunden im Gegensatz zu den Touristenkunden in **Abbildung 23: Touristennachfrage am Donnerstag** im Sommer am Donnerstag eher bereit sind höhere Preise zu zahlen.

Wenn man all diese Modelle evaluiert, werden folgende Ergebnisse erzielt.

Gruppe	Modell	Mean MSE (Std)	Mean R2 (Std)
Business	Linear	880.27 (426.86)	0.156 (0.079)
Business	Quadratisch	858.339 (415.759)	0.175 (0.086)
Tourist	Linear	5240.73 (2417.65)	0.110 (0.020)
Tourist	Quadratisch	5156.21 (2424.15)	0.127 (0.030)

Tabelle 17: Resultate des Regressionsbasierten Kundemodells mit zwei Kundegruppen auf den Trainingsdaten

Erneut erreicht das quadratische Model minimal bessere Resultate, jedoch sind diese Unterschiede auch wieder nur sehr klein. Folglich wird auch hier aufgrund der einfacheren Interpretierbarkeit ein lineares Modell gewählt. Grundsätzlich erzielt das Modell als Folge der Aufteilung in Business und Touristen und somit der kleineren Datenmengen pro Gruppe bessere Resultate.

Eine Analyse der Preiselastizitäten für den Sommer, *Nicht Sommer* der Nachfrage bringt folgende Ergebnisse hervor.

Erhöhung Preis Von -> Nach	**Montag**	**Dienstag**	**Mittwoch**	**Donnerstag**	**Freitag**	**Samstag**	**Sonntag**
Business							
50 -> 60 CHF	-0.285 *-0.325*	-0.274 *-0.314*	-0.284 *-0.338*	-0.226 *-0.353*	-0.086 *-0.062*	-	-
100 -> 120 CHF	-0.795 *-0.962*	-0.753 *-0.915*	-0.792 *-1.019*	-0.584 *-1.090*	-0.188 *-0.132*	-	-
150 -> 180 CHF	-1.981 *-2.782*	-1.812 *-2.528*	-1.965 *-3.116*	-1.238 *-3.596*	-0.310 *-0.212*	-	-
Tourist							
50 -> 60 CHF	-0.288 *-0.228*	-0.299 *-0.243*	-0.312 *-0.261*	-0.282 *-0.233*	-0.200 *-0.168*	-0.192 *-0.160*	-0.247 *-0.192*
100 -> 120 CHF	-0.810 *-0.591*	-0.854 *-0.641*	-0.908 *-0.707*	-0.786 *-0.609*	-0.499 *-0.405*	-0.476 *-0.380*	-0.657 *-0.475*
150 -> 180 CHF	-2.041 *-1.257*	-2.235 *-1.414*	-2.495 *-1.639*	-1.943 *-1.313*	-0.998 *-0.761*	-0.938 *-0.703*	-1.469 *-0.935*

Tabelle 18: Preiselastizitäten des regressionsbasierten Nachfragemodells mit Kundengruppen

In **Tabelle 18: Preiselastizitäten des regressionsbasierten Nachfragemodells mit Kundengruppen** wird ersichtlich, dass der Businesskunde ein umgekehrtes Verhalten zum Touristen zeigt. So weist das Modell der Business Kunden ausserhalb des Sommers eine höhere Preiselastizität auf. Am Freitag ist die Elastizität des Businesskundemodels sehr gering, da sich die wenigen Businesskunden, die am Freitag dort sind, sehr wahrscheinlich unabhängig vom Preis für eine Übernachtung entscheiden. Hingegen das Touristennachfragemodel zeigt im Sommer eine höhere Preiselastizität. Dies könnte dem Umstand geschuldet sein, dass die Touristen im Sommer ein klares Budget haben und damit nicht beliebig hohe Preise bezahlen können.

Weitere Auswertungen und Hintergründe zur Entwicklung des vorliegenden Modells können dem Study-Doc unter **Regressionsbasiertes Kundenmodell mit Kundengruppen** entnommen werden.

Fazit

Die Gruppierung der Reservationen nach Geschäfts- und Touristenreservationen zeigt einen spannenden Unterschied in den Preiselastizitäten. Diese könnten in einer weiteren Forschungsarbeit weiter analysiert werden. Durch die Kombination der zwei Modellen Business und Tourist muss vom Reinforcement Learning Modell eine gewisse Non-Linearität gelernt werden.

Dennoch bestehen für dieses Modell erneut zwei Themengebiete, welche offene Fragen aufwerfen.

Berechnung der Nachfrage mit der Konkurrenz

- Wie wird die Gesamtnachfrage inklusive der Konkurrenzhotels modelliert?
- Wird jeder einzelne Preis des Agenten und der Konkurrenz mit dem Model evaluiert und die Nachfrage summiert?
- Wird eine Aggregierung des Preises der Agenten und der Konkurrenz als Preis für das Model verwendet?
    - Mittelwert: Ein einzelnes Hotel kann mit einem hohen oder tiefen Preis die Nachfrage verzerren
    - Median: Bei vier Hotels haben nur die beiden Hotels mit den «mittleren» Preisen einen Einfluss auf die Nachfrage

Wahl des Hotels

- Wählt der Kunde immer anhand des tiefsten Preises unabhängig der Hoteleigenschaften? Das ergibt eine einfache Preisstrategie: Unterbiete die Konkurrenz!
- Angenommen der Kunde interessiert sich für das Hotel mit Preis 100 CHF, dieses ist nun ausgebucht. Wählt er trotzdem das Konkurrenzhotel mit einem Preis von 400 CHF?

Feature-basiertes XGBoost Modell mit Einbezug der Konkurrenz

Um die in den Modellen 1 bis 4 besprochenen Problemen zu beheben, soll mit dem vorliegenden Modell ein **Kombiniertes Modell** entwickelt werden. Das heisst, es gibt keinen Unterschied mehr zwischen Nachfrage- und Kundemodell, sondern das Modell berechnet direkt aus den Informationen des Hotels und der Konkurrenz die Anzahl Buchungen für das Agentenhotel. Dadurch kann implizit die Konkurrenzsituation und das Kundenverhalten modelliert werden.

Folglich wird auch hier ein XGBoost Modell entwickelt, welches folgende Zielgrösse hat:

Anzahl Personen, welche für an einem bestimmten Tag ein Hotel aus den Reservationsdaten buchen

Diese Vorhersage soll dabei aufgrund folgender Features berechnet werden:

- Pro Hotel (Agent und Konkurrenten)
    - Preis der Übernachtung
    - Eigenschaften des Hotels (luxuriös, …)
- Zeit (Wochentag, Quartal, …)

Die Konkurrenz, welche als Inputfeatures dient, kann im vorliegenden Fall auf zwei verschiedene Arten definiert werden:

- Alle anderen Hotels aus dem Reservationsset
- **Konkurrenz** aus Konkurrenzdatensatz gemäss Kosinus-Ähnlichkeit

Damit dieses Modell eine Aussagekraft hat, muss ein signifikanter Einfluss durch die Eigenschaften bzw. der Preise der Konkurrenten ersichtlich sein. Dies kann erneut über die Feature Wichtigkeiten des XGBoost Modells überprüft werden.

Resultate

Das Modell wird gemäss **Aufteilung Datensatz Nachfragemodell** auf dem Cross-Validation-Set trainiert und validiert. Dabei wird als Konkurrenz alle anderen Hotels aus dem Reservationsdatensatz verwendet. Mit dieser Konkurrenzsituation kann auf dem Cross-Validation-Set im letzten Split ein sehr hoher R2 Wert von 0.958 erreicht werden.

Wenn man die Feature Wichtigkeiten betrachtet (**siehe Abbildung 24: Feature Wichtigkeiten des XGBoost Modell mit Konkurrenz aus Reservationsdatensatz**), wird ersichtlich, dass die Eigenschaften der «Konkurrenz» einen kleinen Einfluss haben, dieser aber vernachlässigbar ist.

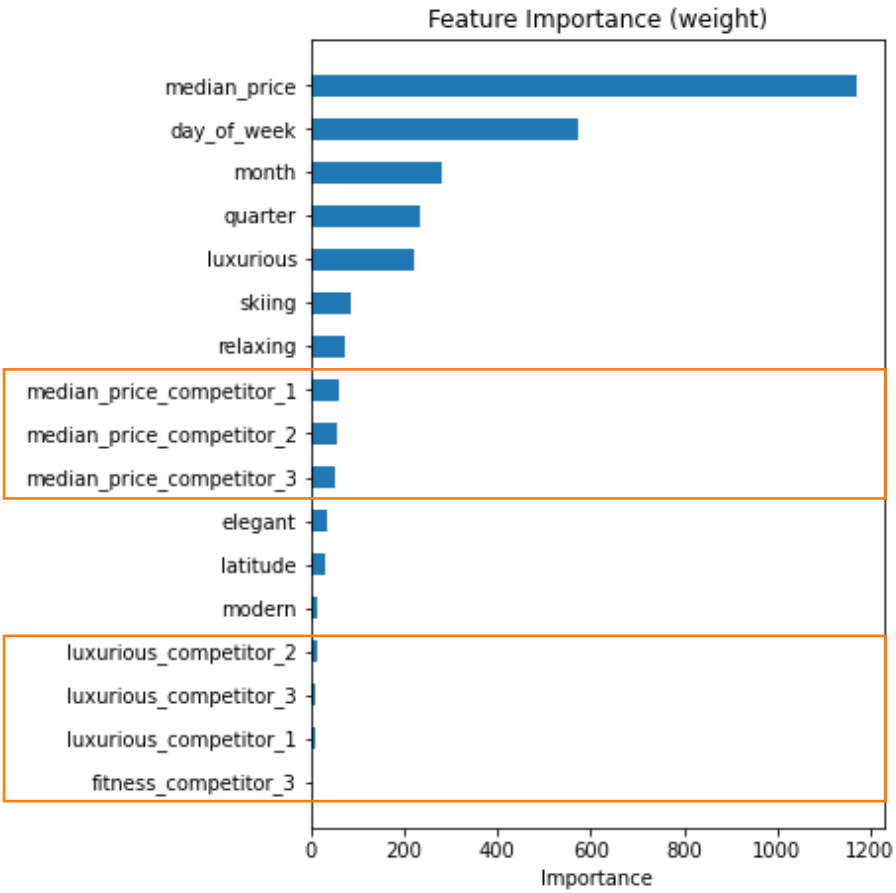


Abbildung 24: Feature Wichtigkeiten des XGBoost Modell mit Konkurrenz aus Reservationsdatensatz

Dieser sehr kleine Einfluss der Eigenschaften der Konkurrenz auf die Nachfrage ist insofern nachvollziehbar, dass diese Hotels nicht in wirklicher Konkurrenz stehen und nur eine kleine Anzahl an Hotels und Konkurrenten vorhanden ist. Folglich ist dieses Modell nicht geeignet für die Bestimmung der Nachfrage in Abhängigkeit von den Preisen des Hotels und der Eigenschaften der Konkurrenten.

Um eine realistischere Konkurrenzsituation zu ermöglichen, wird ein analoges Modell mit dem **Konkurrenzdatensatz** trainiert. Dabei wird pro Agentenhotel (Reservationsdaten) mehrere 3er Kombinationen aus den 10 Hotels mit der grössten Kosinus-Ähnlichkeit als Konkurrenz gewählt. Damit kann ein R^2 Wert von 0.801 im letzten Split erreicht werden.

Jedoch konnte erneut kein messbarer Einfluss der Konkurrenz auf die Resultate der Vorhersage festgestellt werden, was man der **Abbildung 25: Feature Wichtigkeiten des XGBoost Modell mit Konkurrenz gemäss Kosinus-Ähnlichkeit** entnehmen kann.

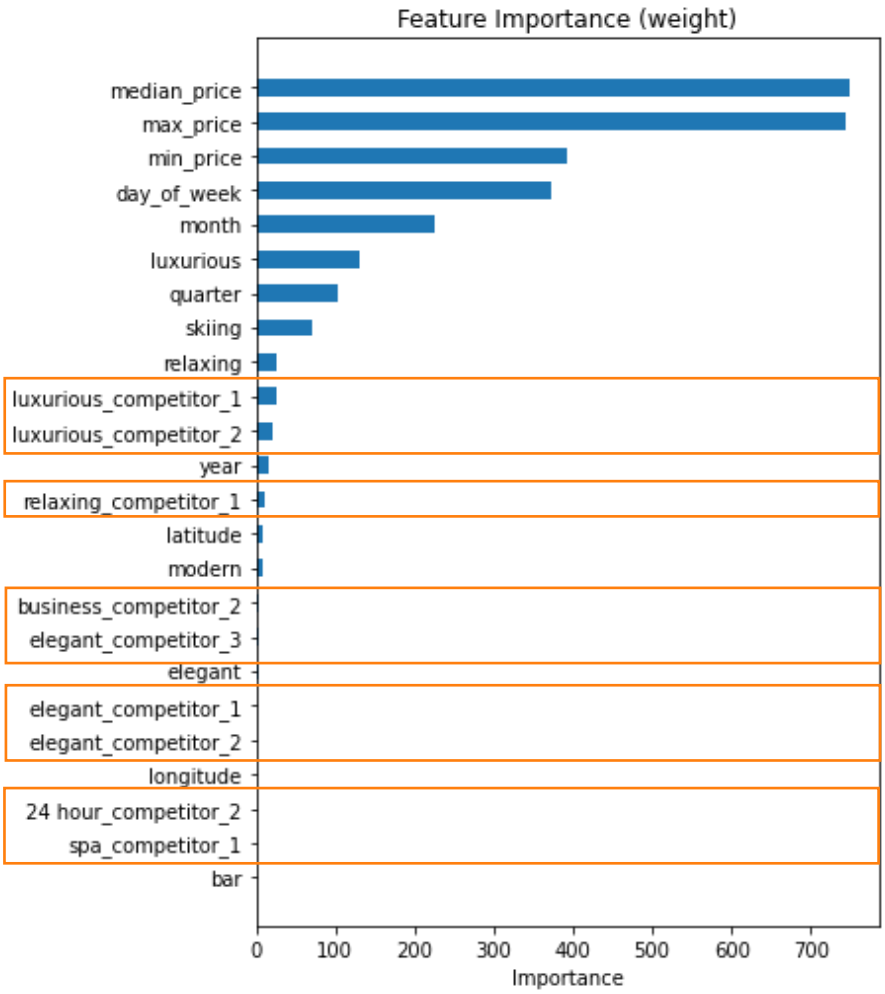


Abbildung 25: Feature Wichtigkeiten des XGBoost Modell mit Konkurrenz gemäss Kosinus-Ähnlichkeit

Ähnlich zu den Resultaten mit dem Reservationsdaten sind die Preisdaten der Konkurrenz fast gar nicht wichtig für die Bestimmung der Nachfrage des Agentenhotels.

Fazit

Schlussendlich kann festgestellt werden, dasss es mit einem XGBoost Modell und den vorliegenden Daten nicht möglich ist, ein kombiniertes Modell zu entwickeln. Denn es lässt sich keinen Zusammenhang zwischen den Konkurrenzeigenschaften und der Nachfrage für ein Hotel zu feststellen. Mögliche Gründe dafür könnten folgende sein:

- Zu wenige Hotels im Reservationsdatensatz
- Konkurrenz nicht korrekt definiert
- Falsches Machine Learning Modell

Reinforcement Learning Umgebung

Für eine einfache Integration von unterschiedlichen Reinforcement Learning Algorithmen, Konkurrenz- und Nachfragemodellen wird eine Reinforcement Learning Umgebung (`PricingEnv`) konzipiert und entwickelt. Die Umgebung ist parametrisierbar entworfen und erlaubt den einfachen Austausch von Konkurrenz (`CompetitorPriceModel`)- und Nachfragemodellen (`DemandCalculator`). Diese Umgebung (siehe **Abbildung 26: Aufbau der Reinforcement Learning Umgebung**) erlaubt es eine Simulation des Dynamic Pricing Verhaltens mit unterschiedlichen Agenten durchzuführen.

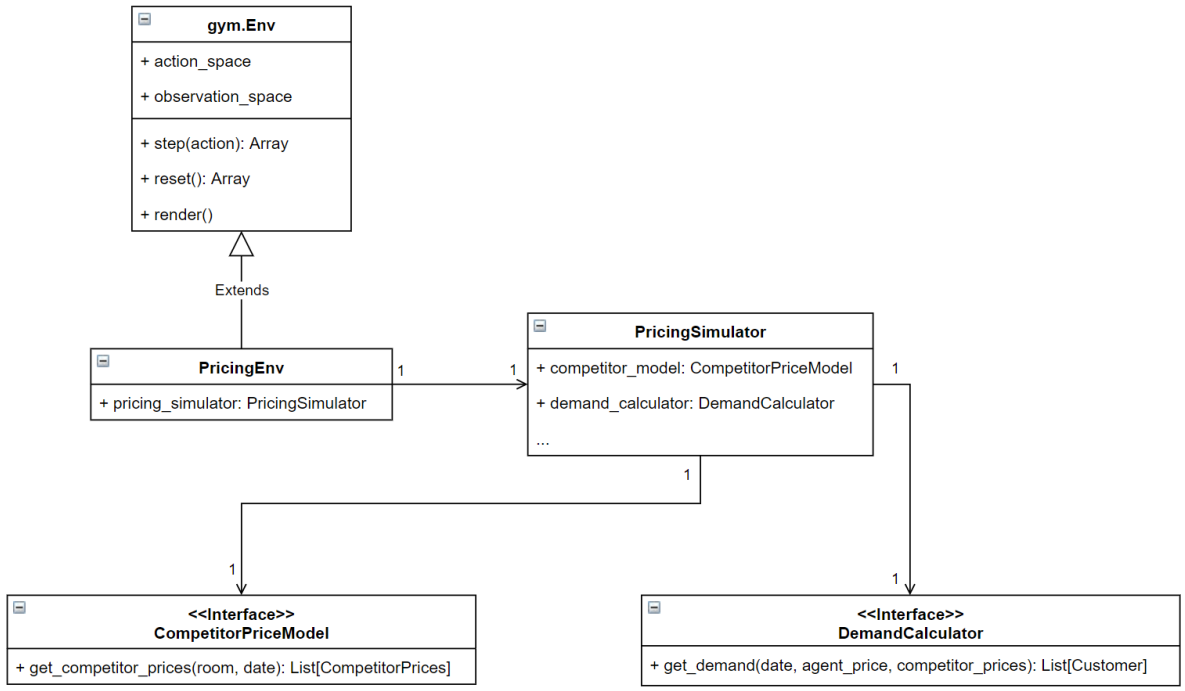


Abbildung 26: Aufbau der Reinforcement Learning Umgebung

Umgebung

Technisch wird die Umgebung als Gym[4] Environment umgesetzt. Gym ist eine in der Forschung viel verwendete Bibliothek, welche es ermöglich eine Reinforcement Learning Umgebung standardisiert zu modellieren. Für viele Standardprobleme gibt es bereits bestehende Gym Umgebungen. Da es sich jedoch bei der vorliegenden Arbeit um ein eigenes Problem handelt, muss diese Umgebung selbst modelliert werden.

Die Details zur Implementation der einzelnen Methoden kann dem Anhang unter **Reinforcement Learning Umgebung mit Gym** entnommen werden.

Action

Die Aktion ist wie bereits beschrieben der gesetzte Preis pro Hotelzimmer für die folgende Nacht. Dies wird als kontinuierlicher Wert zwischen 0 Franken und einem parametrisierbaren Maximalwert repräsentiert. Die genaue technische Implementation wird im Kapitel **Modellierung der Action** beschrieben.

[4] Gym: https://www.gymlibrary.ml/

Reward

Der Reward wird, wie in den Ideen beschrieben, anhand der Anzahl der gebuchten Zimmer berechnet. Der Reward R_{t+1} repräsentiert den Gewinn, die der Agent erzielt, wenn er die Aktion $A_{t,}$ also den Preis für eine bestimmte Nacht setzt, erhält. Dieser Reward R_{t+1} ist folglich ebenfalls eine kontinuierliche Zahl, welche sowohl negativ (Verlust) wie auch positiv (Gewinn) sein kann.

State

Der State repräsentiert den aktuellen Zustand des Systems und ändert sich aufgrund der Aktionen des Agents. Der State beinhaltet die Beschreibungen und letzten Preise des Agentenhotels sowie der Konkurrenz. Weitere Details sind im Anhang unter **Modellierung des State** zu finden.

Erweiterbarkeit der Umgebungsmodelle

Für die Integration der Nachfrage- und Konkurrenzmodelle wird eine Erweiterbarkeit sichergestellt. Die Nachfrageberechnung wird als Schnittstelle (`DemandCalculator`)[5] definiert. Diese beinhaltet eine Methode, welche folgende Informationen entgegennimmt.

- Aktueller Tag
- Preis und Eigenschaften des Agentenhotels
- Liste der Preise und Eigenschaften der Konkurrenzhotels

Dies ermöglicht eine einfache Erweiterung von zusätzlichen Nachfragemodellen, welche für einen bestimmten Tag anhand der erhaltenen Eigenschaften und Preisen eine Anzahl potenzieller Kunden zurückgeben können.

Für die Bestimmung der Konkurrenten und ihre Preise wird ebenfalls eine Schnittstelle (`CompetitorPriceModel`)[6] entwickelt, welche anhand folgender Parameter

- Aktueller Tag
- Eigenschaften des Agentenhotel

eine Liste von Konkurrenten und ihrer Preisen zurückgibt. Dabei kann die Implementation der Schnittstelle bestimmen, wie die Konkurrenz definiert ist und welche Preise die Konkurrenz für einen bestimmten Tag anbieten.

Eine Veränderung der Anzahl Konkurrenten bedingt zusätzlich eine Anpassung des **State.** Hier wäre bei einer variablen Anzahl Konkurrenten denkbar, dass die Grösse des Zustandsraums auf die maximal mögliche Anzahl gesetzt wird und nicht vorhandene Konkurrenzwerte auf 0 gesetzt werden.

[5] Nachfrage Schnittstelle: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/demand/demand_calculator.py#L17
[6] Konkurrenz Schnittstelle: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/competitors/competitor_price_model.py#L24

Agenten

Im Rahmen der vorliegenden Arbeit werden unterschiedliche Agenten entwickelt und überprüft.

Dabei kann zwischen drei Arten von Agenten unterschieden werden:

- Referenzagenten
- Einfache Agenten
- Reinforcement Learning Agenten

Der Referenzagent wählt den Preis anhand der historischen Reservationsdaten und dient somit als Vergleich zu den echten Daten. Einfache Agenten hingegen prüfen, ob der Einsatz eines Reinforcement Learning Agenten überhaupt Sinn macht oder ob mit einem einfachen Verhalten nicht bereits bessere Resultate erzielt werden können.

Schlussendlich gibt es die Reinforcement Learning Agenten, welche mittels eines Offline RL Algorithmus mit der Bibliothek RLLIB[7] trainiert werden und abhängig vom Zustand S_t einen Preis setzen.

Referenzagent – Reservationsdaten

Der Referenzagent verwendet als Aktion A_t immer den Preis, welcher an diesem Tag laut den Reservationsdaten bezahlt wird. Dieser Agent dient als Vergleich, ob der Reinforcement Learning Algorithmus signifikant bessere Preise als die historischen Daten setzt.

Fixpreisagent

Der Fixpreis-Agent ist ein einfacher Agent. Er fällt seine Entscheidung über den Preis unabhängig vom Zustand S_t und setzt eine Aktion $A_t = c$. Folglich wird dieser Agent über den ganzen Zeitraum den gleichen konstanten Preis setzen.

Min-Konkurrenzpreisagent

Der Min-Konkurrenzpreisagent setzt als Preis A_t einen Wert 0.95-mal den Minimalwert der Konkurrenzpreise vom Vortag t-1.

Das heisst die Aktion A_t des Min-Konkurrenzpreisagenten wird folgendermassen berechnet:

$$A_t = 0.95 * min(Preis\ Konkurrent\ 1_{t-1}, Preis\ Konkurrent\ 2_{t-1}, Preis\ Konkurrent\ 3_{t-1})$$

Conservative Q-Learning Agent

Der Conservative Q-Learning (CQL)[8] Agent in RLLIB basiert auf der SAC Implementation des in Kapitel **Conservative Q-Learning** beschriebenen Offline Reinforcement Learning Algorithmus. Dieser Algorithmus berechnet in einem Trainingsprozess anhand der Daten aus **Offline-Datengenerierung** die Q-Values für einen bestimmten Preis. Nach dem Training soll dieser Algorithmus eine Preisstrategie gelernt haben, welche den **Referenzagent – Reservationsdaten** schlägt.

Weitere Reinforcement Learning Agenten wie **MARWIL Agent** und **Behavior Cloning (BC) Agent** können dem Anhang entnommen werden. Diese Algorithmen sind implementiert und integriert, werden aufgrund der Resultate aus dem **Nachfrage- und Kundenmodell** nicht in Experimenten verwendet.

[7] RLLIB Reinforcement Learning Bibliothek: https://docs.ray.io/en/latest/rllib/index.html
[8] Conservative Q-Learning RLLIB Implementation: https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#conservative-q-learning-cql

Offline-Datengenerierung

Alle Offline Reinforcement Learning Algorithmen basieren auf bereits generierten Erfahrungen. Diese beinhalten folgende Elemente:

- Zustand S_t
- Aktion A_t
- Reward R_t

In den historischen Reservationsdaten sind die Aktionen, also die Preise enthalten. Der effektive Zustand und der erhaltene Reward (Gewinn) sind jedoch nicht ersichtlich. Deshalb werden mit Hilfe des Environments (Konkurrenzmodell) und den Reservationsdaten Zustände und Rewards generiert. Dazu wird der **Referenzagent – Reservationsdaten** in die Reinforcement Learning Umgebung integriert. Damit werden dann verschiedenste Episoden simuliert und jeweils der Zustand, die Aktion und der erhaltene Reward aufgezeichnet. Diese Erfahrungen werden in einer JSON Datei gespeichert, welche das spätere Training der Offline Reinforcement Learning Algorithmen ermöglichen.

Limitationen

Dieses Vorgehen ist nicht optimal und hat einige Limitationen.

So wird beispielsweise der Zustand, also die Konkurrenz und ihre Preise, anhand der Kosinus-Similarität und des Konkurrenzmodells berechnet. Dies ist nur eine Annäherung an die reale Situation, da die wirklichen Konkurrenten in der echten Welt nicht bekannt sind. Ebenfalls basiert die Nachfrage auf den Reservationsdaten, diese sind auch nur sehr limitiert korrekt, da nur die Kunden sichtbar sind, die effektiv gebucht haben und nicht diese, welche sich aus irgendwelchen Gründen, z.B. Preis, gegen eine Reservation entschieden haben.

Zusätzlich ist die Berechnung des Rewards nur eine Approximation, denn das Entscheidungsmodell, welches Hotel gebucht wird, ist ebenfalls nur eine Annäherung an die Realität. Die effektiven Gründe für die Buchung bei Hotel x oder Hotel y sind pro Kunde individuell. Ebenfalls sind die Kosten, welche für ein Hotel entstehen für die Bereitstellung der Zimmer sehr vereinfacht dargestellt.

Diese Limitationen schränken zwar die Generalisierbarkeit der Resultate ein, ermöglichen es jedoch, dass die Ergebnisse einfacher interpretierbar sind und Annahmen über das effektive Kundenverhalten reduziert werden können.

Training

Im Rahmen des Trainings verwendet ein Offline-RL-Algorithmus die Erfahrungen aus der Offline-Datengenerierung. Dabei interagiert der Reinforcement Learning Algorithmus während des Trainingsprozesses nicht mit dem Environment.

Folglich verwendet der Algorithmus in einem Trainingsschritt einen Batch der Daten aus dem Offline-Datensatz und versucht seine Policy zu verbessern. Damit wird dann über eine hohe Anzahl an Iterationen ein Algorithmus trainiert, wobei immer wieder die Daten aus dem Offline-Datensatz gesampelt werden.

Als Abbruchsbedingung wird im vorliegenden Fall eine bestimmte Anzahl Iterationen festgelegt, welche der Algorithmus durchlaufen muss. Alle 100 Iterationen werden zusätzlich die Modelparameter (Checkpoints) zwischengespeichert. Dies erlaubt eine spätere Wiederaufnahme des Trainingsprozesses und eine Analyse der Policy, welche im Trainingsverlauf gelernt wurde.

Für die Messung der Performance wird das Modell alle 200 Iterationen auf dem Environment evaluiert. Dies gibt während des Trainingsprozesses einen Hinweis darauf, ob das Modell etwas lernt. Dabei ist jedoch wichtig zu beachten, dass diese erzielten Rewards nicht an das Modell zurückfliessen.

Schlussendlich nach Abschluss des Trainings das Modell erneut gespeichert und kann in einem nächsten Schritt in einer Validierung bereitgestellt und überprüft werden.

Bestimmung der Validierungszeiträume

Für die qualitative Auswertung des trainierten RL-Algorithmus, wird das Modell auf zwei zweiwöchigen Zeiträumen mit mindestens 1000 Preisen im Jahr 2020 evaluiert. Diese sind folgendermassen definiert.

- 1: 2 Wochen (Mo-Mo-So) mit der **höchsten** Standardabweichung der Preise im Reservationsdatensatz innerhalb dieser zwei Wochen
- 2: 2 Wochen (Mo-Mo-So) mit der **tiefsten** Standardabweichung der Preise im Reservationsdatensatz innerhalb dieser zwei Wochen

Dadurch können die Agenten in zwei unterschiedlichen Szenarien überprüft werden, ein Zeitraum mit stark variierenden, wie auch eher statischen Preisen.

Folgende Zeiträume werden dabei anhand der Reservationsdaten berechnet:

	Zeitraum	**Mean Preis**	**Standardabweichung**	**Max / Min**
Max Standardabweichung	Mo. 17.08 2020 – So. 30.08.2020	125.84	55.68	364 / 34
Min Standardabweichung	Mo. 09.11.2020 – So. 22.11.2020	92.54	26.39	327 / 38

Tabelle 19: Zeiträume für die Validierung

Wie in **Tabelle 19: Zeiträume für die Validierung** ersichtlich, befindet sich der Zeitraum mit der höchsten Standardabweichung im August und somit mit grosser Wahrscheinlichkeit in der Hochsaison. Hingegen der Zeitraum mit der minimalen Standardabweichung hat eher tiefere Preise und liegt im November in der Nebensaison.

Dies erlaubt eine Validierung der Agenten auf zwei «interessanten» Zeiträumen.

Evaluation und Validation

Die realisierten **Konkurrenzmodelle**, **Nachfrage- und Kundenmodelle** und **Agenten** werden gemäss der Methodik der **Evaluation** evaluiert und validiert. Dabei werden die Resultate der Konkurrenz und Nachfragemodelle zusammengefasst. Zusätzlich wird ein Experiment in der **Reinforcement Learning Umgebung** mit dem **Konkurrenzmodell** und dem Modell «**Konstantes Kundemodell**» durchgeführt und die Preisstrategie des **Conservative Q-Learning Agent** auf zwei Zeiträumen betrachtet.

Konkurrenzmodellierung

Die Evaluation der Konkurrenz besteht aus zwei Hauptkomponenten, einerseits ist die **Bestimmung der Konkurrenz** für das Agentenhotel ein Kernpunkt für den Zustand der Reinforcement Learning Umgebung. Andererseits ist die **Modellierung der Konkurrenzpreise** für diese Konkurrenten wichtig für die Bestimmung der Anzahl Buchungen, die für die jeweiligen Hotels gemacht werden.

Bestimmung der Konkurrenz

Für ein Agentenhotel werden jeweils drei Konkurrenten mit der grössten Kosinus-Ähnlichkeit der Beschreibungen ermittelt.

Aufgrund der verfügbaren Daten ist diese Bestimmung der Konkurrenz eine sehr natürliche Art, welche ebenfalls skaliert. In der Realität werden Hotels meist als Konkurrenten betrachtet, wenn sie ähnliche Eigenschaften besitzen.

Folgende Nachteile besitzt dieses Vorgehen dennoch:

- Die Qualität dieser Eigenschaften ist unklar (Veränderung der Beschreibung führt zu neuer Konkurrenz)
- Die physikalische Distanz wird nicht betrachtet
- Die Anzahl der Konkurrenten ist konstant

Deshalb könnte in einem nächsten Schritt die Anzahl der Konkurrenten variiert werden. Zusätzlich wäre es denkbar, dass zusätzliche Daten aus dem Buchungssystem mehr Informationen darüber geben, welche Hotels geklickt werden, bevor die effektive Buchung vollzogen wird.

Eine weitere Möglichkeit könnte die Validierung der Bestimmung der Konkurrenz durch Experten wie die Betreiber der Hotels sein. Dies könnte eine Art von Ground-Truth-Daten liefern, welche einen Abgleich der Resultate ermöglichen. Zusätzlich ist denkbar, dass die Resultate zwischen den unterschiedlichen Distanz- und Ähnlichkeitsmassen wie die Euklidische Distanz oder Kosinus-Ähnlichkeit verglichen werden.

Modellierung der Konkurrenzpreise

Für diese gemäss Kosinus-Similarität bestimmten Konkurrenten wurde für die Vorhersage der Preise aus dem Booking.com Datensatz ein Modell entwickelt. Dieses XGBoost-Modell berechnet diese Preisstrategie anhand der Features wie Hotelinformationen (Ort, Sternebewertung, …), Zeit (Quartal, Wochentag, …) und historischen Preis (Letzter Preis, Preis vor einem Monat, …).

Aus den **Resultaten** der Vorhersage der Konkurrenzpreise wird ersichtlich, dass das Bestimmtheitsmass R^2 auf ungesehenen Daten (Testdatensatz) zwischen 0.62 und 0.811 liegt.

Datensatz	**Bestimmtheitsmass R^2**
Dezember	0.718
Bristol	0.811
London	0.62

Tabelle 20: Resultate des Nachfragemodells auf den Testdaten

Die **Tabelle 20: Resultate des Nachfragemodells auf den Testdaten** zeigt ebenfalls, dass eine gute Generalisierung auf einen anderen Ort (Bristol) gemacht wird. Auch mit ungesehenen Zeiträumen wie Dezember scheint das Modell gut umgehen zu können.

In einem weiteren Schritt könnte betrachtet werden, warum das Modell für die Hotels in London weniger zuverlässige Resultate liefert. Zusätzlich könnten folgende zusätzliche Verbesserungen betrachtet werden:

- Rückkopplung zwischen Agenten und Konkurrenzpreise
- Andere Machine Learning Modelle (Neuronale Netze, Zeitreihenmodelle, …)
- Erweiterung durch Daten ausserhalb von UK

Damit könnte die Performance der Vorhersage der Preise aus dem Konkurrenzdatensatz gesteigert werden.

Nachfrage- und Kundenmodell

Neben der Entwicklung und Validierung des Konkurrenzmodells werden zusätzlich die unterschiedlichen Nachfrage- und Kundemodelle evaluiert. In **Feature-basiertes XGBoost Modell** werden zwar gemäss der **Evaluation der Konkurrenzpreis- und Nachfragemodelle** die Nachfrage zufriedenstellend ($R^2 > 0.75$) vorhergesagt. Jedoch kann in der Auswertung der **Resultate** festgestellt werden, dass eine kontraintuitive Beziehung zwischen dem Preis und der Nachfrage besteht. So hat ein Anstieg des Preises eine Erhöhung der Nachfrage zur Folge. Dieses Resultat widerspricht den volkswirtschaftlichen Grundlagen der Nachfragekurve und empirischen Resultaten.

Eine manuelle Definition der Kundengruppen (siehe **Konstantes Kundemodell**) birgt die Gefahr, dass unwissenschaftliche Annahmen zum Kundenverhalten getroffen werden, was signifikant grosse Auswirkungen auf die Resultate der Reinforcement Learning Simulation hat.

Einen Ausweg daraus bietet das **Regressionsbasiertes Kundenmodell mit Kundengruppen**, bei welchem mit Hilfe einer linearen Regression eine negative Preiselastizität festgestellt werden kann. Dennoch können auch bei diesem Modell einige Schwierigkeiten festgestellt werden. Die Qualität der Vorhersage gemäss R^2 ist auf den Trainingsdaten mit unter 0.2 gering. Zusätzlich ist ein Einbezug der Nutzerpräferenzen, der Nachfrage der Konkurrenz und die Entscheidung des Kunden ungeklärt.

Das **Feature-basiertes XGBoost Modell mit Einbezug der Konkurrenz** wird als **Kombiniertes Modell** umgesetzt. Hier werden aufgrund der Preise und Eigenschaften des Agentenhotels und der Konkurrenten mittels XGBoost Modell direkt die Anzahl Buchungen des Agentenhotels bestimmt. Dadurch werden das Kundenmodell und die Nachfrage implizit aus den Daten abgeleitet. Jedoch kann hier festgestellt werden, dass mit den verwendeten Methoden und Datenquellen nur ein sehr schwacher Einfluss der Konkurrenz vorhanden ist. Dies hat zur Folge, dass bei einem Einsatz dieses Modelles im Reinforcement Learning Setup die Konkurrenz keinen Einfluss hat und dass der RL Algorithmus für das Agentenhotel schlussendlich monopolitisch die Nachfrage lernen wird.

Aufgrund der soeben beschriebenen Ergebnisse kann mit den vorliegenden Nachfrage- und Kundenmodellen keine sinnvolle Reinforcement Learning Umgebung simuliert werden, ohne dabei ein starkes Bias zu erzeugen.

Grundsätzlich gibt es folgende Herausforderungen, welche die Entwicklung eines realistischen Modells kompliziert machen:

- Datenmenge: Es sind nur Buchungen für 11 Hotels vorhanden
- Art der Daten: Es sind keine Nachfragedaten, sondern effektive Buchungsdaten vorhanden
- Kundenmodell: Konkurrenz, zwischen welcher die Kunden entscheiden, ist unbekannt

Einer der aussichtsreichsten Ansätze ist das kombinierte Modell **Feature-basiertes XGBoost Modell mit Einbezug der Konkurrenz**, welches die Annahmen reduziert und ausschliesslich datenbasiert die Anzahl Buchungen des Agentenhotels bestimmt.

Um ein besseres Modell zu erzeugen können folgende Verbesserungen gemacht werden:

- Mehr Hotels mit Reservationsdaten
- Komplexere Machine Learning Modelle
- Bessere Bestimmung der Konkurrenz durch zusätzliche Daten (z.B. Recommender System)

Ein verbessertes Nachfrage- und Kundenmodell ist essenziell dafür, dass die Resultate aus der Reinforcement Learning Umgebung aussagekräftig sind. Auf diesen Bereich sollte bei einer Weiterführung dieser Arbeit ein Hauptaugenmerkt gelegt werden.

Reinforcement Learning

Im Rahmen der Evaluation der Reinforcement Learning Umgebung wird das Modell «**Konstantes Kundemodell** » zusammen mit dem **Konkurrenzmodell** im parametrisierbaren Reinforcement Learning Setup überprüft. Wie bereits beschrieben sind die Nachfragemodelle nur sehr bedingt aussagekräftig. Dennoch wird damit ein RL Setup aufgebaut, um die Möglichkeiten eines Offline Reinforcement Learning Algorithmus zu evaluieren.

Interpretation der Reinforcement Learning Ergebnisse

Bei der Interpretation der Reinforcement Learning Ergebnisse ist zu beachten, dass diese nicht generalisierbar sind und nur einen ersten Einblick in die Ergebnisse des Offline Reinforcement Learning im Dynamic Pricing gewähren.

Dies ist darauf zurückzuführen, dass die Resultate sehr stark von der Modellierung der Umgebung (Konkurrenz, Nachfrage, …) abhängen. Zusätzlich werden die Reinforcement Learning Algorithmen nur mit einem Random Seed trainiert.

Folglich sollen in einer Weiterführung dieser Arbeit eine Verfeinerung des Environments umgesetzt werden und der anschliessende Trainings- und Evaluationsprozess wie in **Evaluation des Reinforcement Learning Algorithmus** beschrieben mit einer hohen Anzahl verschiedener Random Seeds durchgeführt werden.

Dennoch geben die folgenden Evaluationen bereits einen guten ersten Eindruck in die Möglichkeiten eines Offline Reinforcement Algorithmus.

Reinforcement Learning mit konstantem Nachfragemodell

Beim vorliegenden Experiment wird das **Konstantes Kundemodell** in das Reinforcement Learning Setup integriert. In diesem Modell gibt es drei unterschiedliche Kunden, welche unabhängig voneinander zwischen den Hotels und ihrer **Konkurrenz** auswählen. Dabei haben die Hotels immer genug Platz für alle Kunden.

Evaluation Reward-Kurven

Während des Trainingsprozesses des Conservative Q-Learning Algorithmus kann nach einem ersten Einbruch des durchschnittlichen Rewards ein Anstieg beobachtet werden. Auf dem Niveau zwischen 11'000 und 36'000 Mean Reward stabilisiert sich der Trainingsprozess.

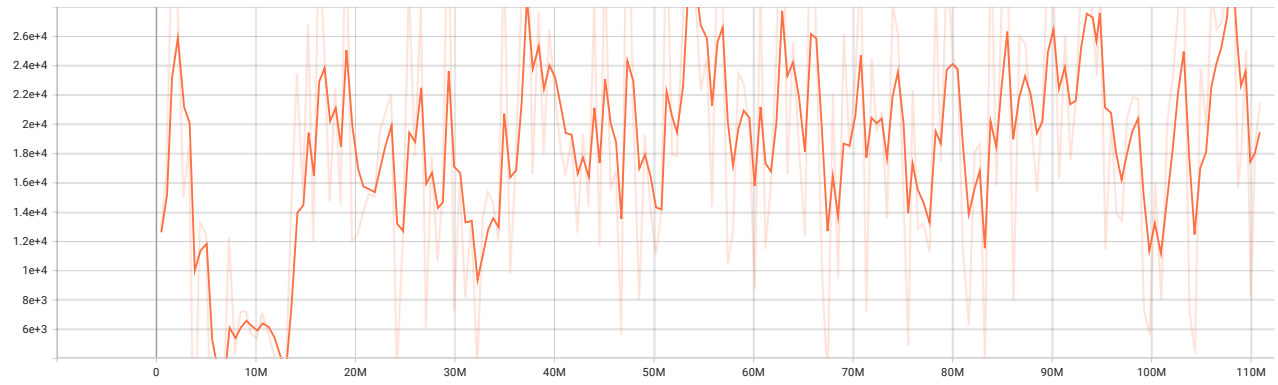


Abbildung 27: Mittlerer Reward des CQL Algorithmus mit dem konstanten Nachfragemodell über 250000 Trainingsschritte

Anhand der **Abbildung 27: Mittlerer Reward des CQL Algorithmus mit dem konstanten Nachfragemodell** kann angenommen werden, dass das Modell eine Preisstrategie erlernt hat, welche positive Rewards erzielt.

Evaluation auf den 2-Wochenzeiträumen

Um die exakte Preisstrategie zu analysieren, werden die folgenden Agenten auf allen Reservationshotels für die Zeiträume gemäss **Bestimmung der Validierungszeiträume** evaluiert.

- Referenzagent – Reservationsdaten
- Fixpreisagent (CHF 75)
- Min-Konkurrenzpreisagent (Faktor 0.95)
- Conservative Q-Learning Agent

Die anderen Reinforcement Learning Algorithmen werden mit diesem Nachfragemodell nicht trainiert, da zu dem Zeitpunkt der Evaluation das Nachfragemodell bereits invalidiert wurde.

Analyse der Hauptsaison (Mo. 17.08 2020 –So. 30.08.2020)

Für die Simulation der beiden Wochen im August zeigt sich sehr stark, dass der CQL Agent gute Ergebnisse abliefert und für alle Hotels im Gesamten positive Rewards erzielt.

Agent	Mean Total Reward (Std)	Max Total Reward	Min Total Reward
Referenz	219.00 (303.31)	560.00	-357.00
Fixpreis	455.00 (972.11)	980.00	-1120.00
Min-Konkurrenz	833.67 (378.45)	1038.05	220.52
CQL	1224.84 (1346.69)	3440.27	43.05

Tabelle 21: Resultate der Agenten bei der Evaluation des Zeitraums 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage

Was aber ebenfalls sichtbar wird in **Tabelle 21: Resultate der Agenten bei der Evaluation des Zeitraums 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage** ist, dass das CQL Modell sehr stark in der Performance schwankt. Dies ist ein Indiz dafür, dass die Strategie nicht bei allen Hotels gleich gut funktioniert und der Agent nicht besonders gut generalisiert.

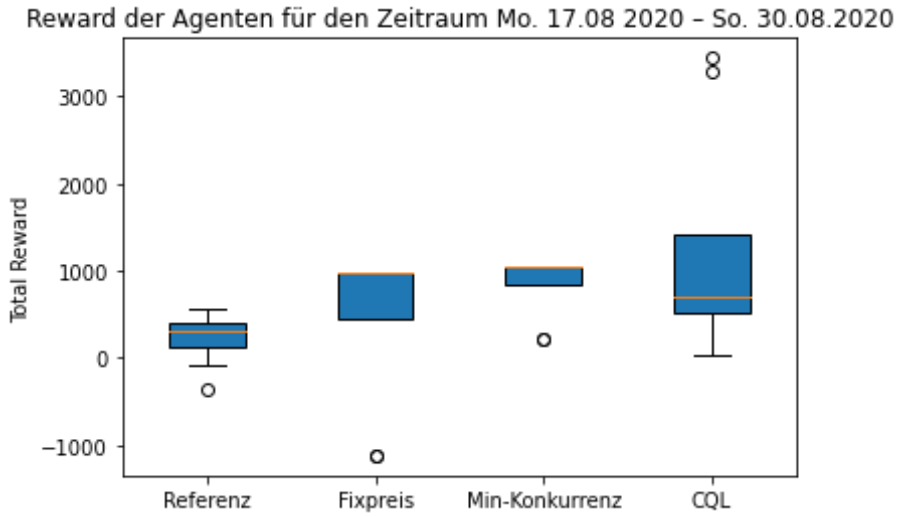


Abbildung 28: Boxplot des Total Rewards der Agenten für den Zeitraum Mo. 17.08 2020 –So. 30.08.2020

Diese Resultate bestätigen sich bei der Betrachtung der **Abbildung 28: Boxplot des Total Rewards der Agenten für den Zeitraum Mo. 17.08 2020 –So. 30.08.2020.** Der Medianwert des Rewards des CQL Agenten liegt über der Referenz aber auch unter des Fixpreis- und Min-Konkurrenzagenten. Zusätzlich sind erneut die hohe Streuung gegen oben sichtbar und die starke Schwankung der Resultate zeigt sich ebenfalls durch die grosse Interquartalsdifferenz.

Eine Analyse der Aktionen (gesetzte Preise) beim Hotel mit der besten Performance (Hotel 1527) im Vergleich zur Konkurrenz zeigt, dass der Agent gelernt hat hohe Preise zu setzen, um somit auf den Geschäftskunden abzuzielen.

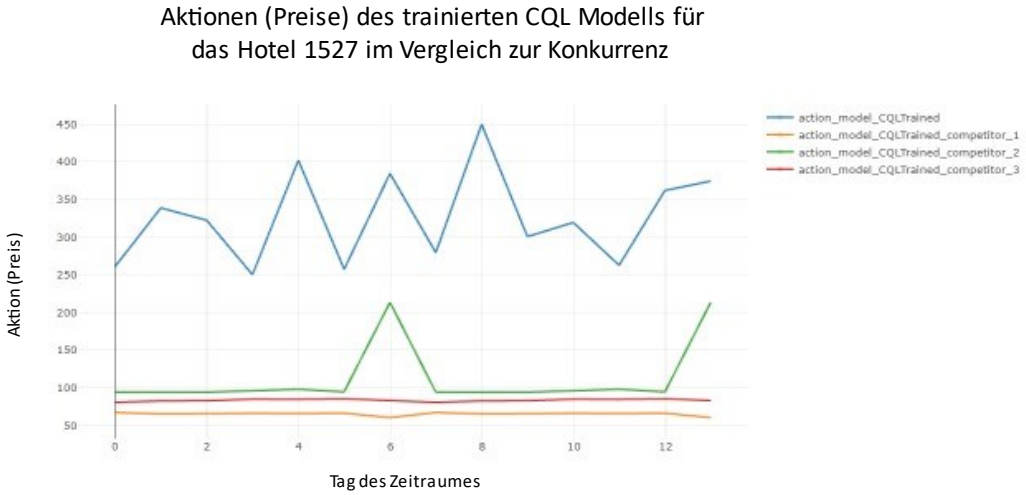


Abbildung 29: Aktionen des CQL Modells für das Hotel 1527 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage

Die Analyse der gesetzten Preise des CQL Agenten für das Hotel 1103 mit der schlechtesten Performance zeigt ein anderes Bild.

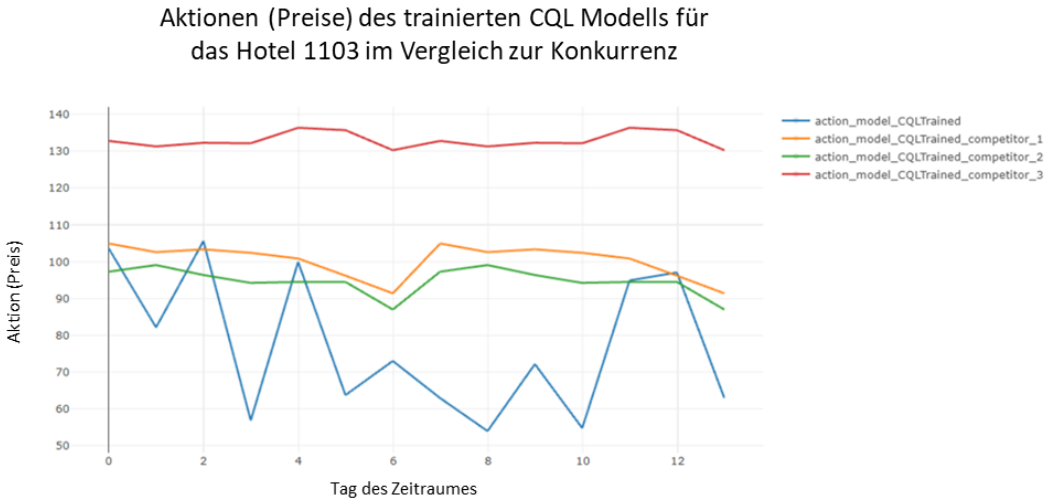


Abbildung 30: Aktionen des CQL Modells für das Hotel 1103 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage

In **Abbildung 30: Aktionen des CQL Modells für das Hotel 1103 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage** wird ersichtlich, dass das Model sich mit den Preisen für dieses Hotel zu einem grossen Teil im vom Touristenkunden (50-100 CHF) erwarteten Preisrahmen aufhält. Das heisst es scheint einen Fokus auf den Touristen zu geben. Dieser Effekt entsteht sehr wahrscheinlich aus dem Grund, dass die Features des Hotels 1103 für den Businesskunden für dieses Hotel eher tief sind.

Um diese Strategie jedoch optimaler zu spielen, müssen die Werte so nahe wie möglich an die Preise der Konkurrenz gesetzt werden, um mit den Touristen Features den grösstmöglichen Gewinn zu erzielen.

Grundsätzlich ist aber eine Tendenz zu beobachten, dass das CQL Modell bessere Resultate erzielt als die Referenzdaten, auf welchen es trainiert wurde. Das heisst das Modell scheint anhand der Rewards der Offline RL Daten die «guten» Aktionen erkennen zu können und diese als Strategie zu erlernen.

Analyse der Nebensaison (Mo. 10.11 2020 –So. 22.11.2020)
Bei der Auswertung des Zeitraums im November zeigt sich ein ähnliches Resultat wie bereits zuvor. Der Min-Konkurrenzagent erzielt erneut eine gute Performance bei einer tiefen Standardabweichung.

Agent	Mean Total Reward (Std)	Max Total Reward	Min Total Reward
Referenz	-255.88 (467.76)	205.00	-1006.00
Fixpreis	455.00 (972.11)	980.00	-1120.00
Min-Konkurrenz	790.16 (352.41)	980.49	219.19
CQL	1048.07 (745.99)	2471.17	353.56

Tabelle 22: Resultate der Agenten bei der Evaluation des Zeitraums 10.11.2020 - 22.11.2020 für das Nachfragemodell mit konstanter Nachfrage

In **Tabelle 22: Resultate der Agenten bei der Evaluation des Zeitraums 10.11.2020 - 22.11.2020 für das Nachfragemodell mit konstanter Nachfrage** ist aber auch ersichtlich, dass das CQL Modell im Schnitt der beste Total Reward erreicht und dabei relativ hohe Werte beim besten, wie auch beim schlechtesten Hotel aufzeigt.

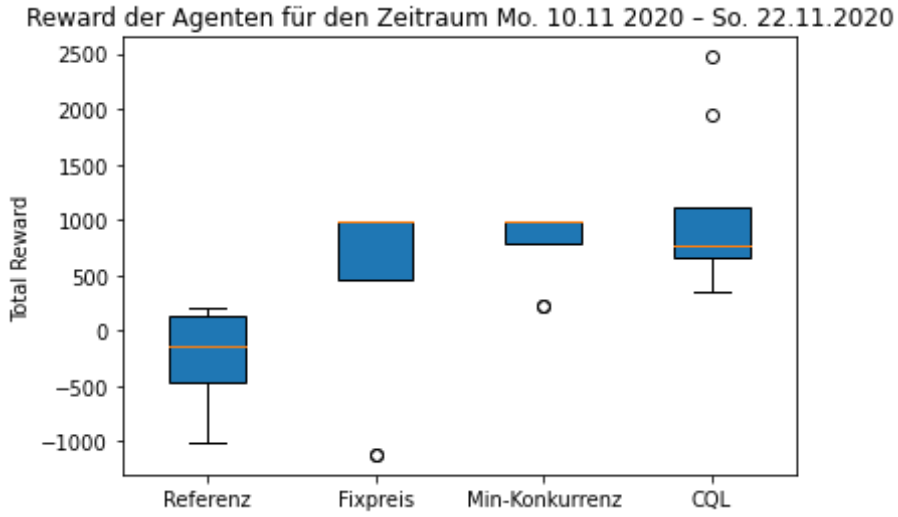


Abbildung 31: Boxplot des Total Rewards der Modelle für den Zeitraum Mo. 10.11 2020 – So. 22.11.2020

Auch in der **Abbildung 31: Boxplot des Total Rewards der Modelle für den Zeitraum Mo. 10.11 2020 – So. 22.11.2020** ist zu erkennen, dass das CQL Modell die Referenz schlägt. Im Vergleich zum Zeitraum im August ist jedoch sichtbar, dass der erzielte Reward weniger variiert. Der Conservative Q-Learning Algorithmus kann mit der vorliegenden Modellierung in einzelnen Fällen den Fixpreis und den Min-Konkurrenzagenten schlagen.

Bei der Betrachtung der Preissetzung des CQL Agenten für das Hotel mit den besten Resultaten (Hotel 1709) zeigt sich erneut, dass grundsätzlich auf den Geschäftskunden abgezielt wird.

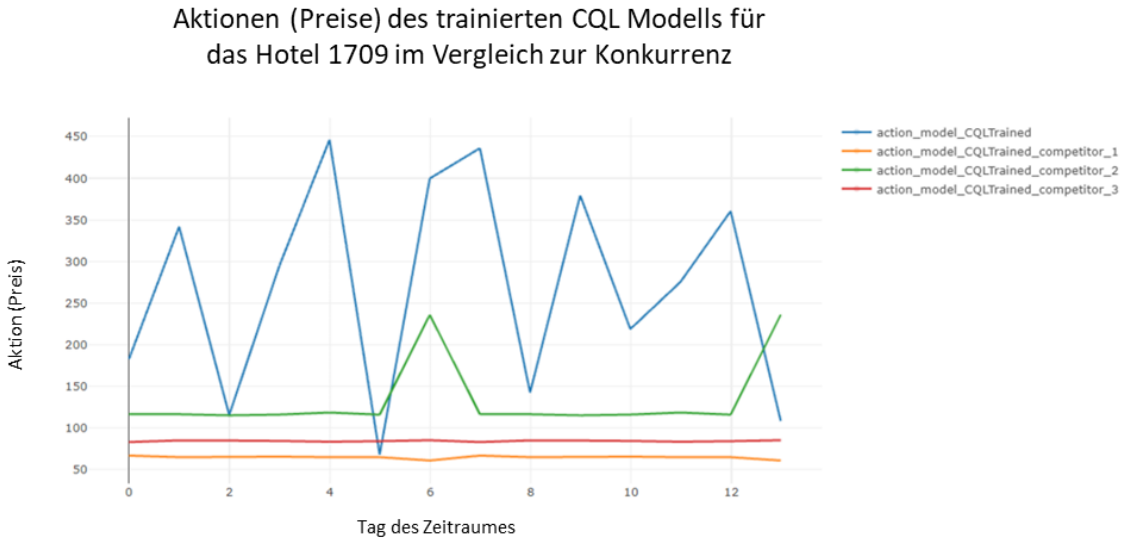


Abbildung 32: Aktionen des CQL Modells für das Hotel 1709 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage

Es wird in **Abbildung 32: Aktionen des CQL Modells für das Hotel 1709 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage** ersichtlich, dass das Modell eine Tendenz dazu hat sehr hohe Preise zu setzen.

Es ist jedoch sehr spannend zu beobachten, dass es trotzdem zu Ausschlägen gegen unten kommt und obwohl die Preise der Konkurrenz relativ konstant sind, eine grosse Varianz in der Preissetzung gibt. Zusätzlich ist ähnlich wie bereits bei den August Monaten zu beobachten, dass zwar ein hoher Preis gesetzt wird, aber nicht den maximal möglichen von 500 CHF erreicht wird. Dies wäre für ein Businesshotel die beste Strategie.

Für das Hotel mit der schlechtesten Performance zeigt sich erneut, dass hier eher tiefe Preise gesetzt werden und entweder der Tourist / den Studenten als Zielkunde gewählt wird.

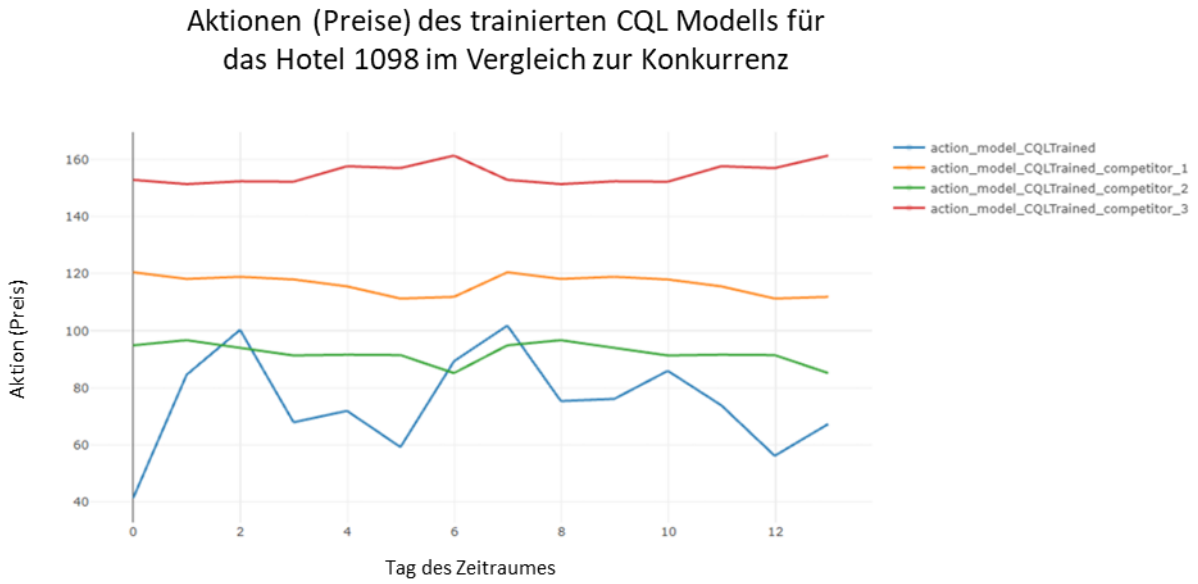


Abbildung 33: Aktionen des CQL Modells für das Hotel 1098 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage

Es wird aber auch in **Abbildung 33: Aktionen des CQL Modells für das Hotel 1098 den Zeitraum 17.08.2020 - 30.08.2020 für das Nachfragemodell mit konstanter Nachfrage** ersichtlich, dass die Preise der Konkurrenz noch nicht perfekt vorhergesagt werden können und dementsprechend einzelne Preise oberhalb des zweiten Konkurrenten (grün) gesetzt werden.

Fazit

In den Resultaten zeigt sich, dass der Conservative Q-Learning Agent im Mittel 82.49 Reward (Gewinn) mehr pro Tag als der Trainingsgrundlage entsprechenden Referenzagent erzielt. Es ist aber in vereinzelten Fällen auch klar ersichtlich, dass einfachere Agenten wie der Min-Konkurrenzagent bessere und stabilere Ergebnisse erzielen. Dennoch kann die Annahme getroffen werden, dass das vorliegende Modell etwas über das **Konstantes Kundemodell** gelernt hat. Zusätzlich ist die Tendenz einer Gewinnmaximierung sichtbar. Die Erklärbarkeit der Resultate hingegen kann nicht vollständig sichergestellt werden. So sind aufgrund des Neuronalen Netzes des CQL Agenten die Entscheidungen schwer nachvollziehbar und es ist eine grosse Schwankung in den gesetzten Preisen feststellbar.

Mit weiterem Training und Optimierung der darunterliegenden Modelle könnten dennoch durchaus Strategien erlernt werden, welche nahe an die optimale Strategie für den Geschäfts-/ Touristen oder Studentenkunden kommt. Mögliche Hindernisse dabei könnten sein, dass aufgrund des Offline Reinforcement Learning Setups in den Daten keine High-Reward Szenarien vorhanden sind und somit diese Strategie nicht gefunden wird. In einem Online-Setting wäre dies unter Umständen aufgrund der Exploration möglich.

Diese Resultate haben grundsätzlich eine nur sehr bedingte Aussagekraft. Es ist klar ersichtlich, dass der RL Algorithmus das Kundenmodell lernt, welches aber nicht der Realität entspricht. Das heisst für andere Kunden- oder Nachfragemodelle können diese Resultate mit hoher Wahrscheinlichkeit nicht reproduziert werden. Dies zeigt jedoch auch die Wichtigkeit einer guten Modellierung eines Nachfrage- und Kundenmodells auf, welches die Grundlage für das Lernen einer sinnvollen Preisstrategie bildet.

Reflexion und Ausblick

Die Erkenntnisse der Realisierung und der Evaluation werden in diesem Kapitel reflektiert. Zusätzlich werden einige **Limitationen** dargelegt und ein **Ausblick** gemacht, welche zusätzlichen Verbesserungen und Erweiterungen bei der Weiterführung der vorliegenden Arbeit möglich sind.

Limitationen

Wie bereits in den Kapiteln **Realisierung** und **Evaluation und Validation** beschrieben, ist die Generalisierbarkeit der Resultate eingeschränkt.

Einerseits wird die Simulation durch **Vereinfachungen der Problemstellung** abstrahiert. So werden beispielweise längere Buchungszeiträume, Rabatte sowie Stornierungen nicht modelliert.

Andererseits ist die Modellierung der Konkurrenz vereinfacht. So ist die Anzahl auf drei Konkurrenten fixiert, welche anhand der Similaritäten der Beschreibungen definiert werden. Dies kann zur Folge haben, dass ein Hotel aus Nordengland mit einem Hotel aus Südengland konkurriert. Diese Konkurrenzsituation kann aufgrund fehlender Ground-Truth Daten nicht validiert werden. Des Weiteren ist die Konkurrenz in der vorliegenden Implementation sehr statisch. So reagiert diese bisher nicht auf die veränderten Preise des Agenten.

Weiter konnte, wie in der Analyse des **Nachfrage- und Kundenmodells** bereits betont, kein Modell entwickelt werden, welches abhängig vom Preis und den Eigenschaften des Agentenhotels und der Konkurrenz die Anzahl Buchungen des Agentenhotels berechnet.

Zusätzlich wird in der Simulation des Reinforcement Learning Modells ein vereinfachtes Kundenmodell verwendet, welches nicht der Realität entspricht. Ebenfalls wird der RL Algorithmus nur mit einem Random Seed trainiert und validiert, was die Reproduzierbarkeit der Ergebnisse einschränkt. Diese Resultate sind ausserdem aufgrund der geringen Anzahl an Hotels im Reservationsdatensatz mit Vorsicht zu geniessen.

Schlussendlich limitieren zusätzliche praktische Einschränkungen die Interpretierbarkeit der Resultate. So wird beispielsweise zwischen den unterschiedlichen Währungen (CHF, GBP, EUR) keine Umrechnungen gemacht.

Reflexion

Im Rahmen der vorliegenden Arbeit wurde die Literatur zu Dynamic Pricing und Reinforcement Learning betrachtet. Nach einer ersten Analyse der Daten konnte die Problemstellung in vier Teilprobleme unterteilt und in einem iterativ-inkrementellen Verfahren bearbeitet werden.

Daraus entstand ein Modell, welches anhand der Kosinus-Ähnlichkeit der Eigenschaften die Konkurrenz bestimmt und den Preis vorhersagt. Die Ergebnisse dieser Modellierung waren sehr ansprechend. Bei der Modellierung der Nachfrage hingegen konnte kein zufriedenstellendes Ergebnis erzielt werden. Dabei ist vor allem der Einbezug der Konkurrenz ein ungeklärtes Problem. Dieses Risiko von unzufriedenstellenden Modellresultaten wurde zu wenig fokussiert. Eine bessere Analyse der vorhandenen Daten und der damit möglichen Modelle zu Beginn des Projektes hätte unter Umständen ein solches Resultat verhindern können. Eine potenzielle Vergrösserung der Datenmenge und der Datenquellen hätten hier zusätzlich Gegensteuer geben können.

Dennoch konnten mit dem Konkurrenzmodell und einem einfachen Kundemodell Experimente mit der entwickelten Reinforcement Learning Umgebung gemacht werden. Dabei konnten bei der Analyse des Trainings des RL Agenten und der Evaluation für zwei Zeitspannen bereits erste positive Ergebnisse festgestellt werden.

Schwierigkeiten während der Arbeit bereiteten vor allem die Modellierung auf der fachlichen Seite, diverse technische Probleme mit der Simulationsumgebung (Nicht genügend Speicherplatz, Abbruch der Simulation, …) und aufgrund der fehlenden Erfahrung die korrekte Evaluierung der Reinforcement Learning Algorithmen.

Schlussendlich ist die persönliche Haupterkenntnis: Ein Reinforcement Learning Algorithmus erzielt nur so repräsentative Resultate, wie es durch die Modellierung der Umgebung möglich ist.

Ausblick

Bei einer Weiterführung der Arbeit sollte der Hauptfokus auf die Verbesserung des Nachfragemodells gelegt werden. Dieses Modell kann in einem weiteren Schritt sehr einfach in die modulare Umgebung integriert und mit unterschiedlichen Agenten validiert werden. Zusätzlich kann die Modellierung der Umgebung mit unter anderem folgenden Punkten erweitert werden:

- Variable Anzahl Konkurrenten
- Grössere Buchungszeitfenster
- Stornierungen

Diese Erweiterungen bringen sehr viel interessantere Szenarien, welche die Möglichkeiten von Reinforcement Learning noch mehr ausreizen können. Diese zusätzliche Komplexität bringt aber auch die Notwendigkeit mit sich, dass für das Agentenhotel eine maximale Zimmeranzahl und die aktuelle Belegung simuliert werden müssen. Ebenfalls sollte das Nachfragemodell Buchungen in die Zukunft mit einer variablen Länge und Stornierungen modellieren. Eine Möglichkeit wäre hierbei, dass die Stornierungen als fixer Prozentsatz repräsentiert werden.

Grundsätzlich würden diese Erweiterungen für den Reinforcement Learning Algorithmus die spannende Herausforderung mit sich bringen, wie man über einen längeren Zeitraum die Auslastung für mehrere Nächte optimiert. Dabei könnte die Wahl zwischen Kurz- und Langaufenthalten oder das frühe oder späte Buchen des Hotels interessante Strategiemöglichkeiten bieten.

Zusätzlich zur Erweiterung der Reinforcement Learning Umgebung könnten folgende zusätzliche Daten spannende Optionen ermöglichen:

- Reservationsdaten von mehr Hotels für einen längeren Zeitraum
- Erzielte Gewinne der Hotels
- Geklickte Hotels im Buchungssystem

Mehr Reservationsdaten würden beispielsweise eine Unterteilung in Trainings- und Validierungshotels und eine bessere Generalisierbarkeit der Resultate erlauben. Zusätzlich könnten durch mehr Reservationen aus dem Jahr 2022 ein Corona-Effekt untersucht werden. Mit Daten zu den Gewinnen hingegen könnte der Reward des Offline-Datensatzes direkt bestimmt werden. Weiter könnten die geklickten Hotels im Buchungssystem oder Daten aus einem Recommendersystem einen besseren Aufschluss über die Konkurrenz geben. Diese zusätzliche Menge an Daten könnten es auch erlauben, dass keine Modellierung der Nachfrage- oder der Konkurrenz mehr notwendig ist und diese direkt als Stichproben aus den Datensätzen entnommen werden.

Ebenfalls bieten sich im Bereich der Reinforcement Learning Algorithmen noch zusätzliche Möglichkeiten indem zum Beispiel ein MARWIL oder ein Behavior-Cloning Agent mit unterschiedlichen Random Seeds trainiert und validiert wird. Durch die Erweiterung des Single-Agent Ansatzes durch zusätzliche RL-Agenten wie die Konkurrenz oder die Kunden können in einem Multi-Agent Ansatz weitere Szenarien analysiert werden. So wären potenzielle Preisabsprachen möglich, welche zu Kartellen führen oder aber es könnte überprüft werden, ob die Agentenhotels gegenseitig den Preis nach unten treiben.

Schlussendlich kann mit der vorliegenden Arbeit und den soeben beschriebenen Erweiterungen der Grundstein gelegt werden, dass damit verbesserte Offline Reinforcement Learning Agenten für das Dynamic Pricing entwickelt werden können, welche vielleicht in Zukunft im realen Einsatz einem Betreiber eines kleinen Gasthauses viel Aufwand ersparen und eine bessere Auslastung seines Hotels ermöglichen.

Literaturverzeichnis

Bayoumi, A. E.-M., Saleh, M., Atiya, A. F., & Aziz, H. A. (2013). Dynamic pricing for hotel revenue management using price multipliers. Journal of Revenue and Pricing Management, 12(3), 271–285. https://doi.org/10.1057/rpm.2012.44

Corgel, J., Lane, J., & Woodworth, M. (2012). Hotel Industry Demand Curves. The Journal of Hospitality Financial Management, 20, 85–95. https://doi.org/10.1080/10913211.2012.10721893

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2019). Deep Reinforcement Learning that Matters. arXiv:1709.06560 [cs, stat]. http://arxiv.org/abs/1709.06560

Hyndman, R. J., & Athanasopoulos, G. (2021). 5.10 Time series cross-validation | Forecasting: Principles and Practice (3rd ed). https://Otexts.com/fpp3/

Kastius, A., & Schlosser, R. (2022). Dynamic pricing under competition using reinforcement learning. Journal of Revenue and Pricing Management, 21(1), 50–63. https://doi.org/10.1057/s41272-021-00285-3

Klein, R., & Steinhardt, C. (Hrsg.). (2008a). Dynamic Pricing. In Revenue Management: Grundlagen und Mathematische Methoden (S. 173–223). Springer. https://doi.org/10.1007/978-3-540-68845-7_5

Klein, R., & Steinhardt, C. (Hrsg.). (2008b). Grundlagen des Revenue Managements. In Revenue Management: Grundlagen und Mathematische Methoden (S. 1–39). Springer. https://doi.org/10.1007/978-3-540-68845-7_1

Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. arXiv:2006.04779 [cs, stat]. http://arxiv.org/abs/2006.04779

Lee, G. (2019). Two Essays on Dynamic Optimal Pricing [Thesis, Georgetown University]. In Georgetown University-Graduate School of Arts & Sciences. https://repository.library.georgetown.edu/handle/10822/1055993

Levine, S. (2020, September 15). Decisions from Data: How Offline Reinforcement Learning Will Change How We Use ML. Medium. https://medium.com/@sergey.levine/decisions-from-data-how-offline-reinforcement-learning-will-change-how-we-use-ml-24d98cb069b0

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. arXiv:2005.01643 [cs, stat]. http://arxiv.org/abs/2005.01643

Maestre, R., Duque, J., Rubio, A., & Arévalo, J. (2018). Reinforcement Learning for Fair Dynamic Pricing. ArXiv:1803.09967 [Cs, Stat]. http://arxiv.org/abs/1803.09967

Martínez-de-Albéniz, V., & Talluri, K. (2011). Dynamic Price Competition with Fixed Capacities. Management Science, 57(6), 1078–1093. https://doi.org/10.1287/mnsc.1110.1337

Michalos, A. C. (Hrsg.). (2014). Veblen Effects. In Encyclopedia of Quality of Life and Well-Being Research (S. 6916–6916). Springer Netherlands. https://doi.org/10.1007/978-94-007-0753-5_104400

Schlosser, R., & Boissier, M. (2018). Dynamic Pricing under Competition on Online Marketplaces: A Data-Driven Approach. 705–714. https://doi.org/10.1145/3219819.3219833

Schlosser, R., & Richly, K. (2019). Dynamic pricing under competition with data-driven price anticipations and endogenous reference price effects. Journal of Revenue and Pricing Management, 18(6), 451–464. https://doi.org/10.1057/s41272-019-00206-5

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (Second edition). The MIT Press.

Wang, Q., Xiong, J., Han, L., sun, peng, Liu, H., & Zhang, T. (2018). Exponentially Weighted Imitation Learning for Batched Historical Data. Advances in Neural Information Processing Systems, 31. https://papers.nips.cc/paper/2018/hash/4aec1b3435c52abbdf8334ea0e7141e0-Abstract.html

Zhang, Q., Qiu, L., Wu, H., Wang, J., & Luo, H. (2019). Deep Learning Based Dynamic Pricing Model for Hotel Revenue Management. 370–375. https://doi.org/10.1109/ICDMW.2019.00061

Abbildungsverzeichnis

Tabellenverzeichnis

Anhänge

Anhang A: Projektmanagement

Die vorliegende Arbeit beinhaltet einen hohen explorativen Anteil. Folglich wird sie in einem iterativ inkrementellen Vorgehen umgesetzt. Dabei sollen die **Projektziele** erreicht und die **Anforderungen** erfüllt werden. Weitere Elemente des Projektmanagements sind die **Stakeholder**, wie auch der **Rahmenplan** und das **Risikomanagement**.

Projektziele

Im Rahmen der vorliegenden Arbeit sollen folgende Ziele erreicht werden.

Ziel	Schlüsselresultate
Ein Single-Agent Reinforcement Learning Algorithmus ist implementiert.	• Eine Reinforcement Learning Umgebung ist technisch umgesetzt. • Die Umgebung bietet Schnittstellen zur Erweiterung an.
Die gelernten Preisstrategien sind überprüft und mit den historischen Daten verglichen.	• Eine Auswertung der Preisstrategie auf zwei Zeiträumen.

Tabelle 23: Übersicht der Projektziele

Anforderungen

Die Anforderungen werden zum grössten Teil der Aufgabenstellung entnommen oder entstanden in Gesprächen mit dem Auftraggeber.

ID	Beschreibung	Quelle
1	Ein Single-Agent Reinforcement Learning Ansatz soll eine Preisstrategie erlernen.	Aufgabenstellung
2	Der RL Agent soll modelliert, implementiert und evaluiert werden.	Aufgabenstellung
3	Die Konkurrenz soll modelliert, implementiert und evaluiert werden.	Aufgabenstellung
4	Die Nachfrage soll modelliert, implementiert und evaluiert werden.	Aufgabenstellung
5	Die Nachfrage und Konkurrenz muss parametrisierbar sein.	Aufgabenstellung
6	Die Nachfrage- und Konkurrenzmodelle werden mit XGBoost umgesetzt.	Daniel Pfäffli
7	Die Preisstrategien werden auf zwei verschiedenen Zeiträumen ausgewertet und mit historischen Daten verglichen.	Aufgabenstellung
8	Die Arbeit enthält einen Ausblick für die Erweiterung für grössere Buchungszeitfenster (>1 Tag) und Einbezug von Stornierungen.	Aufgabenstellung
9	Folgende Technologien müssen verwendet werden: • Git • Docker • Python	Aufgabenstellung

Tabelle 24: Anforderungsübersicht

Stakeholder

Im Projekt sind verschiedene Interessensgruppen involviert. Das Projekt wird von Tobias Heller in Zusammenarbeit mit dem ABIZ Team der Hochschule Luzern (HSLU) durchgeführt. Hauptverantwortlicher der HSLU ist dabei Daniel Pfäffli. Weitere Teammitglieder sind Fabian Gröger und Reza Kakooee, welche bei technischen Hilfestellungen zur Hilfe kommen können.

Weitere Stakeholder beinhalten den Experten Tobias Baltensperger sowie das Unternehmen Room Price Genie (RPG), welches zusammen mit der Hochschule Luzern ein Forschungsprojekt zum Thema Dynamic Pricing durchführen.

Risikomanagement

Das Risikomanagement wird in einer separaten Excel-Liste namens «BAA_Risikomanagement.xlsx» geführt und ist dieser Dokumentation beigelegt. Damit die Dokumentation übersichtlicher bleibt, wird darauf verzichtet, sie direkt in diese Dokumentation einzufügen.

Rahmenplan

Phasen

Das Projekt wird in drei übergreifende Phasen eingeteilt. Zu Beginn findet die einwöchige Initialisierungsphase statt. Anschliessend wird die Konzeptions- und Realisierungsphase in einem iterativen-explorativen Verfahren durchgeführt.

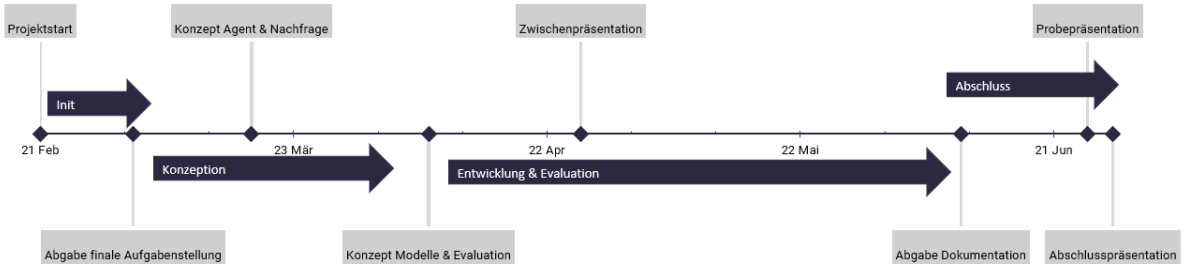


Abbildung 34: Meilensteinplan mit Phasen

Schlussendlich findet die Schlussphase statt, in welcher der Abschluss des Projekts und die Abnahme des Lösungsvorschlages umgesetzt wird.

Detailplan

Meilenstein	**Termin**	**Artefakte**
Projektstart	21.02.2022	-
Finale Aufgabenstellung	04.03.2022	Aufgabenstellung
Konzept Agent & Nachfrage	18.03.2022	Konzeptdokumentation
Konzept Modelle & Evaluation	08.04.2022	Konzeptdokumentation
Zwischenpräsentation	*26.04.2022*	*Zwischenpräsentation*
Abgabe Dokumentation	10.06.2022	Dokumentation
Probepräsentation	25.06.2022	-
Abschlusspräsentation	28.06.2022	Abschlusspräsentation

Tabelle 25: Rahmenplan

Arbeitsjournal

Im Rahmen der Arbeit wird ein Arbeitsjournal geführt. Ein Arbeitsjournaleintrag umfasst Datum, Anzahl Stunden und Arbeitsschritt/Thema. Das Journal ist in einem separaten Dokument «BAA_Arbeitsjournal.xlsx« erfasst und ist dieser Dokumentation beigelegt.

Protokolle

Protokoll BAA Kick-Off

Übersicht

Anlass	BAA Kick-Off
Ort, Datum	MS Teams, 23.02.2022
Uhrzeit	07:00 Uhr – 08:30 Uhr
Teilnehmende	- Pfäffli Daniel (HSLU, Betreuer; DPF) - Heller Tobias (HSLU, Projektmitglied; THE)
Abwesenheiten	Keine
Traktanden	1. Begrüssung 2. Vorstellung Projektauftrag 3. Übersicht Datensätze 4. Übersicht Dynamic Pricing

Inhalt

Traktandum	**Inhalt**
1. Begrüssung	Kurze Begrüssung und Vorstellungsrunde
2. Vorstellung Projektauftrag	Vorstellung des Projektauftrags durch DPF auf Joint Create. THE stellt Frage bezüglich Study-Doc. DPF erklärt das Study-Doc zur Dokumentation aller Experimente dient und als Anhang der Projektdokumentation angefügt werden soll. THE soll sich konkrete Gedanken zu Vorhersage eines Tages oder Zeitraums machen. Wahl von Single-Agent und/oder Multi-Agent Ansatz.
3. Übersicht Datensätze	DPF erklärt die verschiedenen Datensätze. Reservationsdatensatz basiert auf echten Reservationen, diese Hotels sollten simuliert werden. Competitor-Daten beinhalten Metadaten zu Konkurrenten der Hotels. Die Datenquelle sind dabei aus Booking.com gecrawlt. Für diese Hotels besteht zusätzlich ein Datensatz welches der Hotelpreis pro Zimmerkategorie darstellt. Diese Daten sind hauptsächlich aus der UK. Eine mögliche Definition der Konkurrenten könnte beispielsweise aus dem medianen Preis und der örtlichen Nähe erzeugt werden. Zusätzliche mögliche Daten: Reviews
4. Übersicht Dynamic Pricing	DPF erläutert diverse Ansätze wie Dynamic Pricing umgesetzt werden kann. Es wurden folgende Ansätze erklärt: - «Manueller» Ansatz - Game Theory - Econometric approaches - Machine Learning (z.B. XGBoost) - Reinforcement Learning In dieser Arbeit wird laut DPF der Reinforcement Learning Ansatz betrachtet, welcher Ähnlichkeiten zu Game Theory Ansätzen aufweist.

Aufgaben

Aufgaben	**Verantwortlich**	**Bis wann**
Erste Literaturrecherche Fair Dynamic Pricing	Tobias Heller	03.03.2022
Überprüfung und erste Einsicht des Datensätze	Tobias Heller	03.03.2022
Erste konzeptionelle Übersicht über Projekt	Tobias Heller	03.03.2022
Abklärung wie die Booking.com Daten gecrawlt wurden.	Daniel Pfäffli	03.03.2022
Organisation eines Treffens mit RL. Experten	Daniel Pfäffli	03.03.2022
Organisation Termin Zwischenpräsentation	Daniel Pfäffli	SW 07 – SW 10

Protokollführer: Tobias Heller
Datum, Ort: 23.02.2022, Weggis

Anhang B: Aufgabenstellung

Die vorliegende Aufgabenstellung ist eine Kopie des Inhalts aus Jointcreate.[9]

Ausgangslage und Problemstellung

Firmen im Tourismus setzen viel Hoffnung in Dynamic Pricing, um eine bessere Auslastung zu erreichen und damit ihren Gewinn zu verbessern. Im Hotelmanagement von grösseren Hotelbetrieben geschieht dies typischerweise durch einen Angestellten, welcher auf Basis von der aktuellen Auslastung, im Vergleich mit der letzt-jähriger Auslastung, und den Konkurrenz-Preisen eine Preisstrategie entwickeln.

Mit Hilfe von Reinforcement Learning soll ein Model (Agent) eine Preisstrategie erlernen, welches den Gewinn eines Hotels maximiert, und gleichzeitig erklärbar bleibt.

Ziel der Arbeit und erwartete Resultate

In dieser Arbeit soll ein Single-Agent Reinforcement Learning Ansatz für Dynamic Pricing eines Hotels implementiert werden, in welchem erlernte Preisstrategien analysiert werden können. Es soll beantwortet werden, wie plausible und wertvoll die erlernten Preisstrategien sind im Vergleich zu historischen Daten. Für die Arbeit steht ein Hotel-Reservations- und Konkurrenz-Preis-Datensatz zur Verfügung, aus welchen das Konsumenten- und Konkurrenz-Verhalten erlernt werden sollen (offline Reinforcement Learning).

Das RL Setup kann folgende Vereinfachungen beinhalten:

- Buchungszeitfenster 1 Tag
- Keine Stornierungen
- Keine (Gruppen-)Rabatte

Gewünschte Methoden, Vorgehen

Vorgehen:

- Planung, Organisation, Risiko- und Ressourcenmanagement sind Teil der Aufgabe und werden von den Studierenden wahrgenommen
- Während der Arbeit ist ein persönliches Arbeitsjournal zu führen. Ein Arbeitsjournaleintrag umfasst Datum, Anzahl Stunden und Arbeitsschritt/Thema.

Für folgende Teile muss eine Abnahme durch den Betreuer in die Planung aufgenommen werden:

Vor Implementation

- Auswahl des RL-Verfahrens
- Modellierung der Nachfrage und Konkurrenz
- Modellierung des Agenten
- Evaluation der Modelle

[9] Jointcreate Aufgabenstellung: https://home.jointcreate.com/de_ch/ventures/296/

Erwartetes Resultat:

- Ausführliches Study Doc
- Schlussbericht gemäss Vorgaben des Bachelor-Studiengangs Informatik
- Schlusspräsentation nach Abgabe der Arbeit (20 Minuten)
- Implementation und Evaluation eine Konkurrenz-Modelles
- Implementation und Evaluation eines Nachfrage-Modelles
- RL Setup mit Parametrisierung für Nachfrage und Konkurrenz
- Erweiterbarkeit der Umgebungsmodelle
- Evaluation des RL Agenten
- Analyse Preisstrategie für zwei interessante Zeitspannen und Vergleich mit historischen Daten
- Ausblick für grössere Buchungszeitfenster (>1 Tag) und Einbezug von Stornierungen

Study Doc:

Das Führen eines Study Docs soll ein systematisches, reproduzierbares Vorgehen, insb. für die Phase der Modellbildung und des Experimentierens fördern. Es enthält eine Problembeschreibung, einen Vorgehensplan, Ideen, Experimente, sowie deren Resultaten und Interpretationen. Für den Schlussbericht sollen idealerweise grosse Teile direkt aus dem Study Doc übernommen werden können.

Zu verwendende Technologien:

- Git, Python, Docker

Empfohlene Frameworks:

Ray Rllib, Tensorflow

Abgrenzung

Der Algorithmus zum Erlernen des Konkurrenz- und Nachfrage-Verhalten wird nach Absprache mit dem Betreuer festgesetzt. Dafür muss keine Evaluation zwischen verschiedenen Algorithmen gemacht werden.

Die Analyse der Preisstrategie darf auf einzelnen Zeitspannen erfolgen (z.B. eine Woche/ Monat Hochsaison und eine Nebensaison).

Vereinfachungen des Setups, wie z.B. wie weit im Voraus ein Kunde buchen darf, Anzahl Hotels, Attribute eines Hotels, Kundenpräferenzen, können nach Absprache mit dem Betreuer erfolgen.

Kreativität, Varianten, Innovation

RL-Verfahren, Setup, Modellierung, Implementation

Anhang C: Daten

Datenmodell

Die unterschiedlichen Daten werden in einem Datenmodell dargestellt. Dies erlaubt eine erste Übersicht über die verfügbaren Datensätze. Es werden in den folgenden Datensätzen jeweils die wichtigsten Spalten beschrieben.

Konkurrenz[10]

Diese Tabelle zeigt 35'707 Hotels aus dem Vereinigten Königreich. Dabei werden hauptsächlich Informationen zum Ort und Eigenschaften des Hotels repräsentiert. Diese Daten von Booking.com können als Konkurrenz für die Reservationshotels dienen.

Name	Beschreibung	Datentyp	Einheit
hotel_id	Eindeutige Identifikation des Hotels		
latitude	Breitengrad	Number	-
longitude	Längengrad	Number	-
most_popular_facilites	Beliebte Einrichtungen dieses Hotels	String	[«facility1», «facility2», …]
price_median	Medianpreis für dieses Hotel		

Konkurrenzpreise[11]

Der Konkurrenzpreisdatensatz enthält für die Konkurrenzhotels Preisinformationen von Booking.com. Diese wurden alle 3 Tage durch das ABIZ Team gecrawlt. Diese Preise repräsentieren den angezeigten Preis auf der Website und nicht die effektiv bezahlten Preise.

Name	Beschreibung	Datentyp	Einheit
hotel_id	Eindeutige Identifikation des Hotels	Number	-
room_id	Eindeutige Identifikation des Hotelzimmers	Number	-
price	Angezeigter Preis	Number	-
date	Datum der Übernachtung	String	YYYY-MM-DD

[10] Konkurrenzdatensatz: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/blob/main/competitor_dataset.csv
[11] Konkurrenzpreisdatensatz: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/blob/main/hotel_room_prices_clusters.csv

Beschreibung Konkurrenzhotels[12]

Anhand der Beschreibung der Konkurrenzhotels hat das ABIZ Team für jedes Konkurrenzhotel eine Similarität zu gewissen Begriffen berechnet. Diese Similarität besitzt einen kontinuierlichen Wertebereich von 0 bis 1. Zusätzlicher zur Identifikation mit der hotel_id sind Similaritäten zu folgenden Begriffen vorhanden:

luxurious, elegant, relaxing, spacious, modern, restaurant, spa, pool, bar, fitness, leisure, gym, power, breakfast, compact, airport, ski, skiing, beach, view, desk, kitchenette, kettle, friendly, warm, business, hiking, cycling, bicycle, golf, fresh, playground, children, hairdryer, air conditioned, high speed, ski lift, coffee machine, 24 hour

Reservationen[13]

Dieser Datensatz stammt vom Startup Room Price Genie und beinhaltet Buchungen von Kunden für 11 verschiedene Hotels.

Name	Beschreibung	Datentyp	Einheit
hotel_id	Eindeutige Identifikation des Hotels	Number	-
latitude	Breitengrad	Number	-
longitude	Längengrad	Number	-
avg_price	Durchschnittlicher Tagespreis	Number	-
guest_count	Anzahl Gäste	Number	-
night_of_stay	Datum der Übernachtung	String	YYYY-MM-DD
start_date_of_stay	Erstes Datum des Aufenthalts	String	YYYY-MM-DD
end_date_of_stay	Letztes Datum des Aufenthalts	String	YYYY-MM-DD

Hierbei ist wichtig zu beachten, dass die hotel_id des Reservationsdatensatzes keinen Zusammenhang zu den hotel_id des Konkurrenzdatensatzes haben.

Beschreibung Reservationshotels[14]

Analog zu den Beschreibungen der Konkurrenzhotels wurde auch hier durch das ABIZ Team eine Similarität zu gewissen Schlüsselwörtern berechnet. Dabei sind zusätzlich zur hotel_id, hotel_name, address, latitude und longitude die Ähnlichkeit zu folgenden Begriffen vorhanden:

luxurious, elegant, relaxing, spacious, modern, restaurant, spa, pool, bar, fitness, leisure, gym, power, breakfast, compact, airport, ski, skiing, beach, view, desk, kitchenette, kettle, friendly, warm, business, hiking, cycling, bicycle, golf, fresh, playground, children, hairdryer, air conditioned, high speed, ski lift, coffee machine, 24 hour

[12] Beschreibung Konkurrenzhotels: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/blob/main/ft_sim_sent_key_df.csv
[13] Reservationsdatensatz: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/blob/main/reservation_dataset.csv
[14] Beschreibung Reservationshotels: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/blob/main/reservation_hotels_information.csv

Datenverfügbarkeit und Bereinigung

Genauere Details zu Analyse der Datenverfügbarkeit können im Study Doc dem Kapitel **Datenanalyse** entnommen werden.

Anhang D: Modelle und Umgebung

Konkurrenzmodelle

Gemeinsame Features der Konkurrenzmodelle

Folgende Feature Gruppen aus dem Konkurrenz- und Konkurrenzpreise-Datensatz werden für alle drei Modelle erzeugt und verwendet:

Feature Gruppe	Features	Beschreibung	Quelle
Hotelinfos	'hotel_star_rating', 'image_count', 'latitude', 'longitude', 'use_dynamic_pricing', 'room_counts', 'n_rooms', 'n_price_data', 'n_room_data', 'FreeCancellation', 'HalfBoard', 'FullBoard', 'AllInclusive'	Beschreibung des Hotels (Ort, Sterne, …)	Konkurrenz / Konkurrenzpreise
Zeit	'lockdown', 'quarter', 'day_of_week'	Zeit des Preises	Konkurrenzpreise
Preis	'last_price', 'days_since_last_price', 'last_month_price'	Preise desselben Hotels in der Vergangenheit	Konkurrenzpreise

Tabelle 26: Gemeinsame Features der Konkurrenzmodelle

Dabei werden sowohl Informationen zum Hotel wie auch der Zeitpunkt des Preises und Preisinformationen desselben Hotels aus der Vergangenheit definiert.

Features Modell 1

Das Modell 1 verwendet zusätzlich als One-Hot-Vektoren die Werte aus dem «most_popular_facilites» Array. Damit wird versucht herauszufinden, ob es eine Beziehung zwischen Features wie z.B. Gratis-Parkmöglichkeiten und kostenloses Wifi zum Preis gibt.

Feature Gruppe	Features	Beschreibung	Quelle
Beschreibung durch Facilities	'24-hour front desk', 'Air conditioning', 'Airport shuttle', 'Airport shuttle (free)', 'BBQ facilities', 'Baby safety gates', 'Bar', 'Beachfront', 'Body scrub', "Children's playground", 'Daily housekeeping', 'Designated smoking area', 'Facilities for disabled guests', 'Family rooms', 'Fence around pool', 'Fitness centre', 'Free WiFi', 'Free parking', 'Garden', 'Heated pool', 'Heating', 'Hot spring bath', 'Hot tub/jacuzzi', 'Indoor play area', 'Indoor pool', 'Indoor pool (all year)', 'Indoor pool (seasonal)', "Kids' club", "Kids' outdoor play equipment", "Kids' pool", 'Laundry', 'Lift', 'Luggage storage', 'Non-smoking rooms', 'On-site parking', 'Outdoor pool', 'Parking', 'Pets allowed', 'Plunge pool', 'Pool cover', 'Pool with view', 'Pool/beach towels', 'Private beach area', 'Private parking', 'Restaurant', 'Room service', 'Spa and wellness centre', 'Sun loungers or beach chairs', 'Sun umbrellas', 'Swimming pool', 'Swimming pool toys', 'Terrace', 'Water park', 'WiFi'	One Hot Vektoren Features der Hotels	Konkurrenz

Tabelle 27: Zusätzliche Features des Konkurrenzmodells 1

Features Modell 2

Das Modell 2 verwendet anstatt die Most Popular Facilities die Similaritäten von den Schlüsselwörtern zu den Beschreibungen der Hotels als Zusatzinformationen. Dabei werden diese ebenfalls als One Hot Vektoren abgebildet. Dabei wird ein Schwellwert von 0.6 definiert. Folglich werden alle Similaritäten >= 0.6 als 1 und darunter als 0 verwendet. Dieser Schwellwert wurde durch Daniel Pfäffli vorgegeben und hat sich in Experimenten als sinnvoll herausgestellt.

Feature Gruppe	Features	Beschreibung	Quelle
Beschreibung durch Similaritäten	'luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast_x', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour'	One Hot Beschreibung durch Similaritäten zur	Beschreibung Konkurrenzhotels

Tabelle 28: Zusätzliche Features des Konkurrenzmodells 2

Features Modell 3

Das Modell 3 verwendet zusätzlich zu den Similaritäten des Modells 2 noch den letzten Preis der Nachbarn des jeweiligen Hotels. Dabei werden pro Hotel alle Nachbarn einer 1km Distanz gemäss Haversine die Preise vor 3 Tage extrahiert. Diese werden dann aggregiert und der Median, der Minimal und der Maximal Preis gespeichert. Dadurch kann eine Reaktion auf die Preise der direkten Nachbarhotels simuliert werden.

Feature Gruppe	Features	Beschreibung	Quelle
Beschreibung durch Similaritäten	Siehe oben	One Hot Beschreibung durch Similaritäten zur	Beschreibung Konkurrenzhotels
Preis	'price_1km_3_days_ago_median', 'price_1km_3_days_ago_min', 'price_1km_3_days_ago_max'	Historische Preise der direkten Nachbarn	Konkurrenzpreise

Tabelle 29: Zusätzliche Features des Konkurrenzmodells 3

Resultate auf dem Testdatensatz Konkurrenzmodell

Auf dem Dezember Datensatz erhält man für die 10 verfügbaren Tage für die Hotels 35857 und 3314688 folgende Preise:

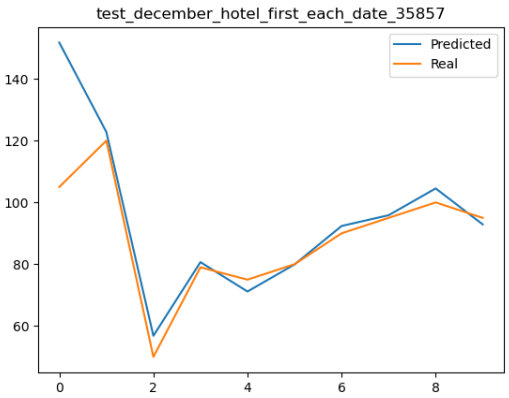


Abbildung 35: Vorhergesagte Preise des Modells für das Hotel 3314688 im Dezember

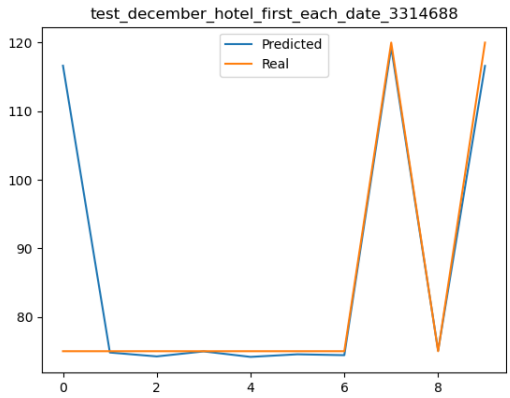


Abbildung 36: Vorhergesagte Preise des Modells für das Hotel 3314688 im Dezember

Datensatz Dezember	R2	Mean Absolute Error	Mean Squared Error
Hotel 35857 / erster Preis pro Tag	0.309	7.184	228.8
Hotel 3314688 / erster Preis pro Tag	0.461	4.866	174.7

Tabelle 30: Performance der Hotels 35857 und 3314688 des Testdatensatz Dezembers

Aufgrund der Graphen sieht die Performance sehr gut aus, jedoch scheint der R2 durch die unzuverlässige Vorhersage am Tag 0 tief zu werden. Dieser grosse Einfluss liegt hauptsächlich daran, dass für beide Hotels nur jeweils für 10 Daten (alle drei Tage) Preisinformationen zur Verfügung stehen.

Wenn man dasselbe für zwei Hotels aus dem Bristol Datensatz macht und für die verfügbaren Tage jeweils den ersten Preis vorhersagt, erhält man folgendes Resultat.

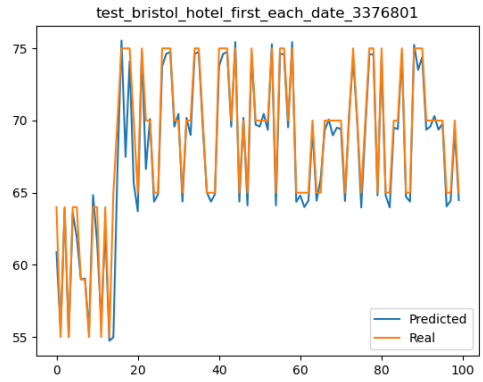


Abbildung 37: 100 erste vorhergesagte Preise des Modells für das Hotel 3376801 in Bristol

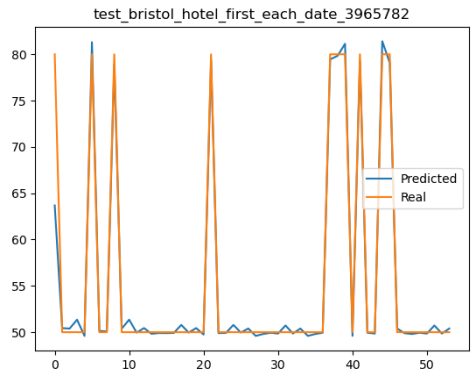


Abbildung 38: 100 erste vorhergesagte Preis des Modells für das Hotel 3965782 in Bristol

Datensatz Bristol	R2	Mean Absolute Error	Mean Squared Error
Hotel 3376801 / erster Preis pro Tag	0.91	0.823	2.402
Hotel 3965782 / erster Preis pro Tag	0.961	18.04	4094.3

Tabelle 31: Performance der Hotels 3376801 und 3965782 des Test-Datensatzes Bristol

Hier ist die Performance auf beiden Hotels sehr gut und es scheint eine starke Vorhersagekraft zu geben.

Um die Qualität des Modells für London genauer zu untersuchen überprüfen, werden ebenfalls zufällig zwei Hotels gewählt und jeweils pro Tag der erste verfügbare Preis vorhergesagt.

Abbildung 39: 100 erste vorhergesagte Preise des Modells für das Hotel 5193388 in London

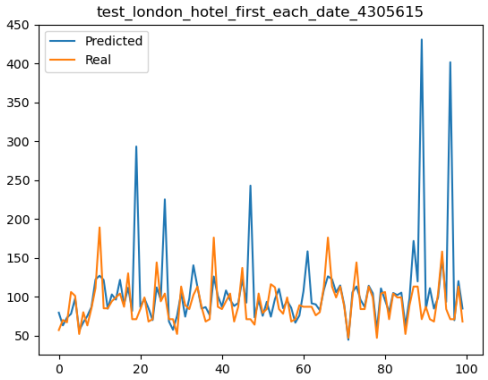


Abbildung 40: 100 erste vorhergesagte Preise des Modells für das Hotel 5193388 in London

Datensatz Dezember	R2	Mean Absolute Error	Mean Squared Error
Hotel 5193388 / erster Preis pro Tag	0.267	18.04	3451
Hotel 4305615 / erster Preis pro Tag	-4.312	18.04	579.8

Tabelle 32: Performance der Hotels 5193388 und 4305615 des Test-Datensatzes London

Hier ist sehr klar zu sehen, dass die Performance auf dem Hotel 5193388 signifikant besser ist als die Performance auf dem Hotel 4305615. Folglich ist das Modell für London nur sehr bedingt einsetzbar.

Kunden- und Nachfragemodelle

Konzepte der Nachfrage- und Kundenmodellierung

Ein mögliches Konzept mit globaler Nachfrage könnte folgendermassen aussehen.

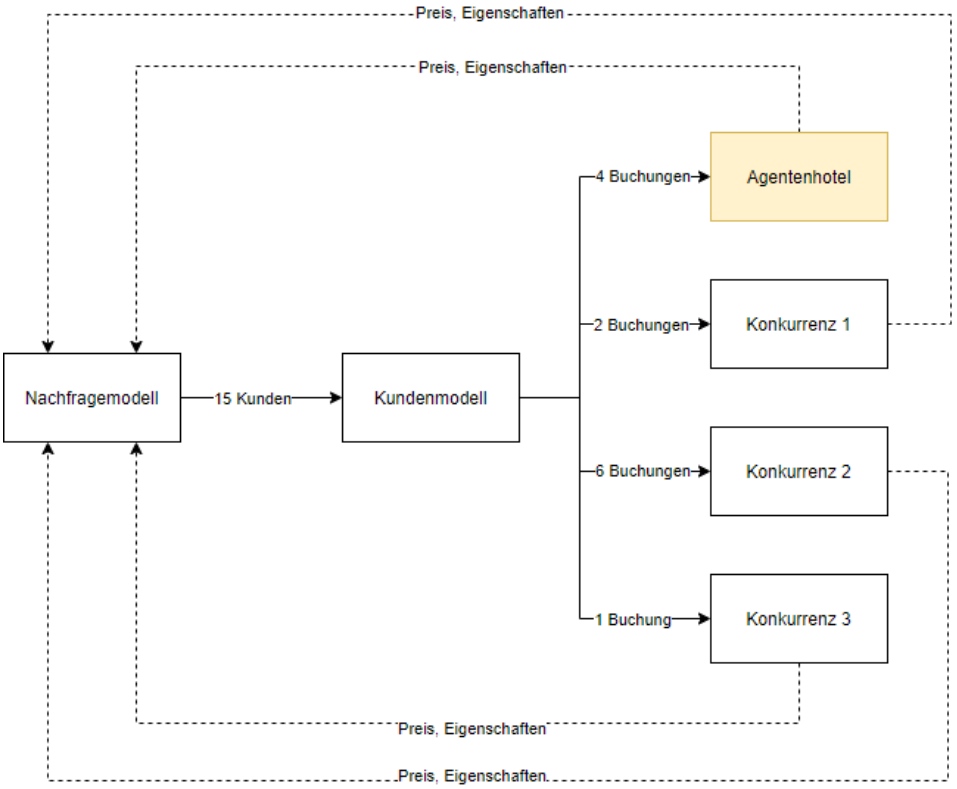


Abbildung 41: Globale Nachfrage und globales Kundenverteilung

Ein Nachfragemodell bestimmt die Anzahl und die Art der Kunden, welche für einen bestimmten Zeitpunkt *t* das Agentenhotel, wie auch die Konkurrenz 1-3 besuchen wollen. Jeder dieser Kunden bestimmt anhand interner Kriterien, welches Angebot er wählt. Dadurch errechnet sich für einen Zeitpunkt *t* die Anzahl Buchungen für das Agentenhotel. Dabei ist jedoch die Schwierigkeit, wie die Informationen zum Preis und den Eigenschaften des Modells für eine globale Nachfrage aggregiert werden können.

Eine andere Möglichkeit ist die Bestimmung der Nachfrage durch lokale Modelle. Dadurch werden pro Hotel (Agent und Konkurrenten) eine Nachfrage generiert, welche dann zu Kunden führt, welche zwischen den Hotels auswählen können.

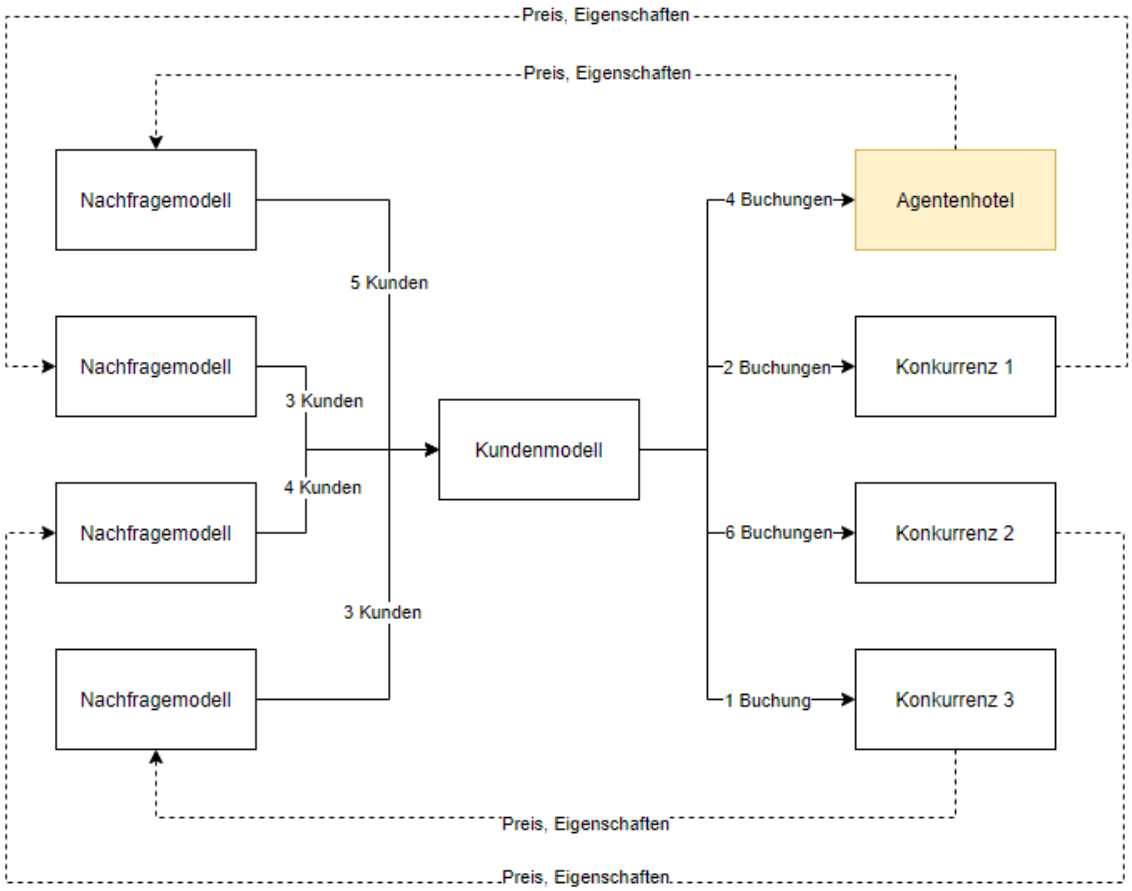


Abbildung 42: Lokale Nachfrage und globale Kundenverteilung

Dies verletzt gewissermassen die Beziehung zwischen der Nachfrage und der effektiven Wahl des Hotels. Dennoch bringt es einige Vorteile, da für die Nachfrage der effektive Preis des jeweiligen Hotels und seine Eigenschaften miteinbezogen werden können.

Eine letzte Möglichkeit besteht aus der Entkopplung zwischen den einzelnen Konkurrenten. Dabei wird für jedes Hotel eine Nachfrage generiert, wobei dann nur mit einer gewissen Wahrscheinlichkeit effektiv das Hotel gewählt wird.

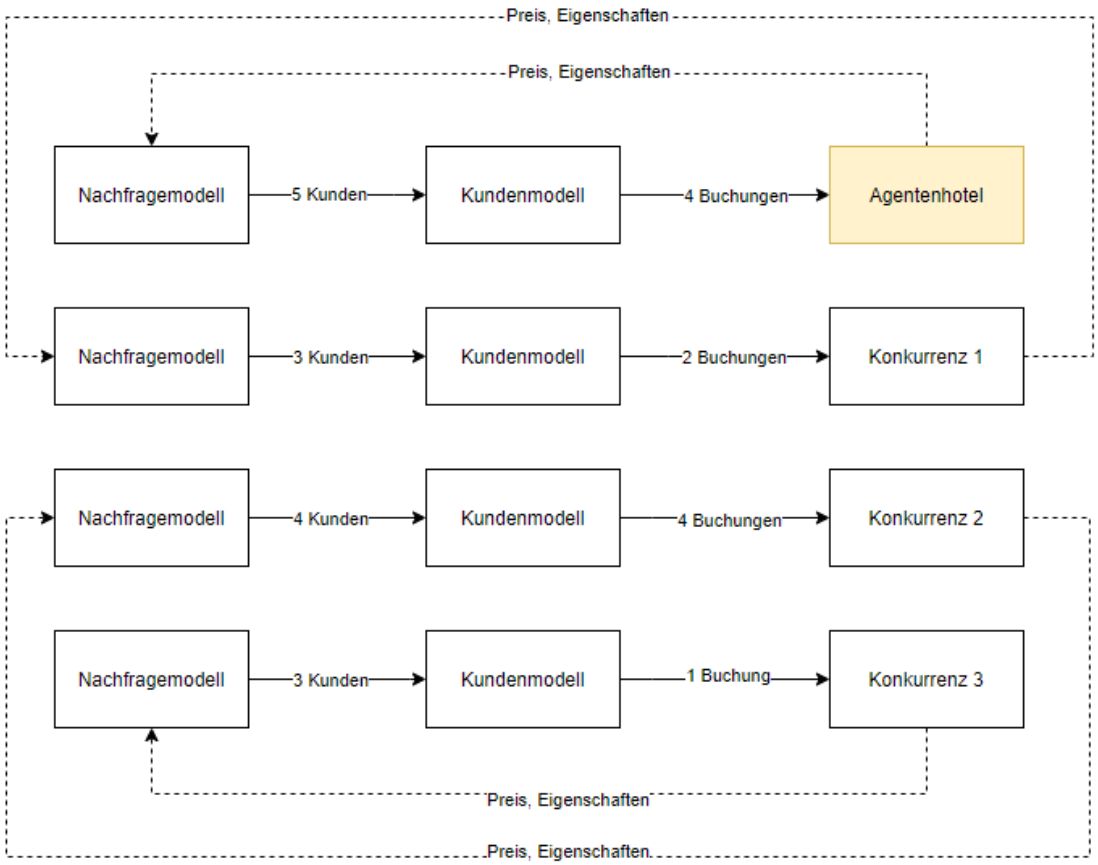


Abbildung 43: Lokales Nachfragemodell und lokale Kundenverteilung

Dies hat wie erwähnt einige Vorteile gegenüber dem vorherigen Modell. Jedoch ist die vorliegende Problematik, dass es keine direkte Konkurrenz mehr zwischen dem Agenten und den anderen Hotels gibt und somit ein grosser Teil der Dynamik entfernt wird.

Mögliche Berechnung des Kundenwerts bei datenbasierten Eigenschaftspräferenzen

Es wäre möglich eine Aufteilung der Kunden in Freizeit (inkl. Wochenenden) und Businesskunden (nur unter der Woche) zu machen. Die Übereinstimmung der Hotels könnte dann durch eine Linearkombination der Features der jeweiligen Hotels bestimmt werden.

$$Featurewert = Featurewichtigkeit\ 1 * Feature\ 1\ Hotel + Featurewichtigkeit\ 2 * Feature\ 2\ Hotel + \ldots$$

Für die Gewichtung zwischen der Preisrangierung und den Features wird für jeden Kunden ein zufälliger Wert zwischen 0.1 und 0.9 exklusiv gewählt.

$$Preisgewichtung = [0.1; 0.9)$$

Der Wert für den Endkunden berechnet sich folgendermassen:

$$Wert\ f\ddot{u}r\ Kunde = Preisgewichtung * Preisrang + (1 - Preisgewichtung) * Featurewert$$

Dadurch kann pro Kunde eine Priorisierung der Angebote gemäss dem Wert erzeugt werden. Anschliessend kann pro Kunde evaluiert werden, welches Hotel gebucht wird und dementsprechend die Anzahl Buchungen für das Agentenhotels berechnet werden.

Reinforcement Learning Umgebung mit Gym

Für die Integration als Gym-Environment muss eine bestimmte Schnittstelle implementiert werden, welche eine Interaktion des Agenten mit einer RL Umgebung ermöglicht. Dabei wird eine Aktion A_t an das Environment übergeben, welche dann ausgeführt und durch das Environment ein Reward R_{t+1} und ein neuer Zustand S_{t+1} zurückgegeben wird.

Dazu muss eine Subklasse von gym.Env definiert und folgende Methoden implementiert werden:

```
class PricingEnv(gym.Env):

  def __init__(self, arg1, arg2):
    super(PricingEnv, self).__init__()
    # Definieren des Action und State/Observation Space (gym.spaces)
    # Beschreibung des Action Space
    self.action_space = ...
    # Beschreibung des State Space
    self.observation_space = ...

  def step(self, action):
    # Führt die Aktion aus und gibt Reward und neuen Zustand zurück
    ...
  def reset(self):
    # Setzt das Environment auf den initialen Zustand
    ...
  def render(self, mode='human', close=False):
    # Anzeigen des Zustands
    ...
```

Die relevanten Methoden sind die `reset` und die `step` Methode. In der `reset` Methode wird das Environment auf den initialen Zustand zurückgesetzt. Im vorliegenden Fall wird dabei zufällig für ein Hotel ein Tag aus den Reservationsdaten initialisiert.

Beim Aufruf der `step` Methode wird die Aktion des Agenten mitgegeben und anhand dieser der Reward berechnet. Im internen Zustand wird die Zeit um einen Tag erhöht und der neue Zustand und der berechnete Reward zurückgegeben.

Modellierung der Action

Der Preis ist dabei eine kontinuierliche Zahl und befindet sich zwischen 0 und 500 Franken pro Nacht. Technisch jedoch erwartet das Environment eine normalisierte Zahl zwischen -1 und +1. Das heisst, es wird ein lineares Mapping von 0 auf -1 und 500 auf +1 gemacht. Diese Normalisierung soll einen zu grossen Einfluss des Preises im Training des neuronalen Netzes vermeiden.

Folglich wird der Action Space im Gym Environment als kontinuierliche Box repräsentiert.

```
self.action_space = gym.spaces.Box(low=-1, high=1, shape=(1,), dtype=np.float32)
```

Modellierung des State

Der State wird bei jedem Zeitschritt und auch beim `reset` zurückgegeben. Da der Conservative Q-Learning Algorithmus nur mit kontinuierlichen Observation Spaces umgeben kann, wird ein Box Space verwendet.

Dabei wird der Tag im Jahr als Zahl zwischen 0 und 365 und das Hotel als Wert zwischen 0 und 8 repräsentiert. Zusätzlich wird der Preis des Vortags zurückgegeben. Dieser wird analog zur Aktion als normalisierter Wert zwischen -1 und 1 repräsentiert. Ebenfalls wird als Beschreibung der Hotels die Similaritäten zu den Begriffen Business, Airport, Desk, Leisure, Relaxing, Swimming Pool, Luxurious und Elegant als Wert zwischen 0 und 1 im State gespeichert.

Dies wird sowohl für den Agenten selbst wie auch für alle Konkurrenten umgesetzt.
Folglich kann der Zustand des Systems als 38-dimensionalen Boxspace repräsentiert werden.

```
self.observation_space = gym.spaces.Box(
    low=np.array([0, 0] + 4 * [-1, 0, 0, 0, 0, 0, 0, 0, 0]),
    high=np.array([366, 8] + 4 * [1, 1, 1, 1, 1, 1, 1, 1, 1]),
    shape=(38,), dtype=np.float32
)
```

Reinforcement Learning Agenten

Weitere Agenten, welche implementiert und integriert sind, befinden sich im folgenden Kapitel. Diese werden aufgrund der Resultate des Nachfragemodells nicht in Experimenten aufgeführt.

MARWIL Agent

Beim Monotonic Advantage Re-Weighted Imitation Learning (MARWIL)[15] Agent handelt es sich um einen Policy-Gradient Algorithmus, welcher ebenfalls für das Offline Reinforcement Learning optimiert wurde und in RLLIB implementiert ist. MARWIL verwendet für das Berechnen der optimalen Policy einen Imitation Learning Algorithmus. Das heisst, der Algorithmus versucht die Behaviour Policy, mit welcher die Daten generiert wurden, zu imitieren. Dabei wird aber anhand des Rewards und einer Advantage Funktion zwischen «guten» und «schlechten» Aktionen unterschieden. (Wang et al., 2018)

Durch ein Sampling aus den Offline-Daten soll eine Policy trainiert werden, welche ebenfalls den **Referenzagent – Reservationsdaten** schlägt.

Behavior Cloning (BC) Agent

Beim Behavior Cloning (BC)[16] Agent handelt sich um eine Anpassung des **MARWIL Agent,** wobei hier der Parameter beta=0 gesetzt wird und ausschliesslich Imitation Learning gemacht wird. Folglich versucht dieser Algorithmus die Behaviour Policy aus der Datengenerierung zu approximieren. Somit wird mit diesem Agenten überprüft, ob es möglich ist, mit Offline Reinforcement Learning eine dem **Referenzagent – Reservationsdaten** ähnliche Preisstrategie zu imitieren.

Dies könnte beispielweise in einem Fall interessant sein, wo alle Daten von einem Experten generiert worden sind, und das Ziel ist dessen Strategie möglichst direkt zu übernehmen.

[15] MARWIL RLLIB Implementation: https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#marwil
[16] Behavior Cloning RLLIB Implementation: https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#behavior-cloning-bc-derived-from-marwil-implementation

Anhang E: Operationalisierungskonzept

Das vorliegende Projekt besteht aus den Teilen Data Engineering, Machine Learning und Software-Engineering. Für die Umsetzung dieser Teilgebiete werden diverse Technologien und Repositories eingesetzt.

Übersicht der verwendeten Technologien

Folgende Technologien werden für das vorliegende Projekt verwendet:

Aufgabe	**Technologie**	**Link**
Versionsverwaltung	Git / GitLab	https://gitlab.enterpriselab.ch/baa-dynamic-pricing
Programmiersprache	Python	https://www.python.org/
Experimentierumgebung	Jupyter Notebook	https://jupyter.org/
ML Framework	XGBoost	https://xgboost.readthedocs.io/en/stable/
RL Framework	RLLIB	https://docs.ray.io/en/latest/rllib/index.html
Virtualisierung	Docker	https://w.docker.com/
Dependency Management	Pip	https://pypi.org/project/pip/
Metrikverfolgung (RL-Training)	Tensorboard	https://www.tensorflow.org/tensorboard
Metrikverfolgung (ML-Training)	MLFlow	https://mlflow.org/
Evaluierungsvisualisierung	MLFlow	https://mlflow.org/

Tabelle 33: Verwendete Technologien

Übersicht der Git Repositories

Für die Versionskontrolle der Daten, Modelle und Resultate wurden drei unterschiedliche Git-Repositories aufgesetzt:

Name	**Aufgabe**	**Link**
Dataset	Verwalten der Daten, Analyse der Daten, Modellieren der Konkurrenz- und Nachfragemodelle	https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
Models	Reinforcement Learning Umgebung	https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models
Results	Resultate aus Modellvalidierungen	https://gitlab.enterpriselab.ch/baa-dynamic-pricing/mlflow-results

Anhang F: Study Doc

Im folgenden Kapitel wird zuerst die **Idee des Study-Docs** erläutert. Zusätzlich wird auf die **Problembeschreibung**, den **Vorgehensplan** und **Erste Ideen** eingegangen. Ebenfalls wird die erste Datenanalyse beschrieben und wie daraus **Konkurrenzmodelle** und **Kundenmodelle** entwickelt werden. Im letzten Kapitel **RL-Experimente** werden die Experimente in der Reinforcement Learning Umgebung, sowie deren Resultate und die Konsequenzen beschrieben.

Die Dokumentation der Parameter und der Umgebung der jeweiligen Experimente sind zu einem Teil in diesem Dokument erfasst. Die Dokumentation der restlichen Parameter und die Sicherstellung der Reproduzierbarkeit werden über den referenzierten Git-Commit umgesetzt.

Grundsätzlich ist das Study Doc als stichwortartige Sammlung von Ideen, Experimenten und Resultaten zu verstehen, welche sich nicht vollständig an die wissenschaftliche Form hält.

Idee des Study-Docs

«Das Führen eines Study Docs soll ein systematisches, reproduzierbares Vorgehen, insb. für die Phase der Modellbildung und des Experimentierens fördern. Es enthält eine Problembeschreibung, einen Vorgehensplan, Ideen, Experimente, sowie deren Resultaten und Interpretationen. Für den Schlussbericht sollen idealerweise grosse Teile direkt aus dem Study Doc übernommen werden können.» Folglich werden zum Teil die stichwortartigen Resultate aus dem Study Doc in Reinform in der Hauptdokumentation zu finden sein.

Problembeschreibung

Um die Aufgabenstellung der Arbeit «Dynamic Pricing mit Reinforcement Learning für die Hotellerie»[17] besser zu verstehen, wird diese in 4 Teilprobleme unterteilt.

Konkurrenz bestimmen und modellieren

Um ein interessantes Dynamic Pricing Modell entwickeln zu können muss für jedes zu simulierende Hotel die Konkurrenz bestimmt und modelliert werden. Dabei soll diese Konkurrenz eine bestimmte Anzahl Hotels definieren, zwischen welchen in der Realität ein Kunde entscheidet.

Dies bringt erhebliche Herausforderungen mit sich, da in den Daten keine einzelnen Kunden und ihr Entscheidungsprozess vorhanden sind. Dadurch muss die Konkurrenz anhand der Daten sinnvoll bestimmt werden. Zusätzlich muss die Preisstrategie der Konkurrenz in unterschiedlichen Situationen (Wochenende, Jahreszeit, Preis der anderen Hotels, …) modelliert werden.

Nachfrage vorhersagen

Damit für das zu simulierende Hotel und seine Konkurrenz eine Bestimmung der Anzahl Buchungen gemacht werden kann, muss zuerst die Nachfrage modelliert werden. Das heisst abhängig von gewissen Eigenschaften der Hotels (Preis, Lage, …) und gewissen zeitlichen Features wie Wochenende oder Saison soll eine Anzahl Kunden zurückgegeben werden.

Kundenverhalten modellieren

Anhand der Nachfrage wird eine gewisse Anzahl Kunden erzeugt, welche anhand gewisser Kriterien eine Buchung vornehmen. Dabei kann der Kunde zwischen dem simulierenden Hotel und seinen Konkurrenten entscheiden.

Grundsätzlich kann davon ausgegangen werden, dass diese Entscheidung vom Preis der Übernachtung abhängig ist. Jedoch ist es auch denkbar, dass der Kunde anhand gewisser Eigenschaften der Hotels nicht unbedingt das preisgünstigste Angebot wählt.

Preisstrategie berechnen und evaluieren

Um eine Preisstrategie mittels eines Reinforcement Learning Modells modellieren zu können, müssen die Komponenten Konkurrenz, Nachfrage und Kunden in ein Reinforcement Learning Setup kombiniert werden. Damit kann ein Algorithmus eine Preisstrategie anhand der historischen Daten lernen.

In einem weiteren Schritt muss diese Preisstrategie analysiert und evaluiert werden. Aufgrund des beschränkten Zeitraums und des finanziellen Risikos, ist es nicht möglich den Algorithmus in der Realität mit einem konkreten Hotel zu testen. Folglich müssen die Algorithmen anhand festdefinierten Metriken wie auch eine qualitative Betrachtung im Rahmen einer Simulation evaluiert und verglichen werden.

[17] Aufgabenstellung JointCreate: https://home.jointcreate.com/de_ch/ventures/296/

Vereinfachungen der Problemstellung

Eine Modellierung eine Dynamic Pricing Simulation kann beliebig komplex umgesetzt werden. Verschiedenste Komponenten wie Konkurrenz, Kundenverhalten, Stornierungen, Buchungszeiträume, ... können in verschiedensten Realitätsgraden konzipiert werden.

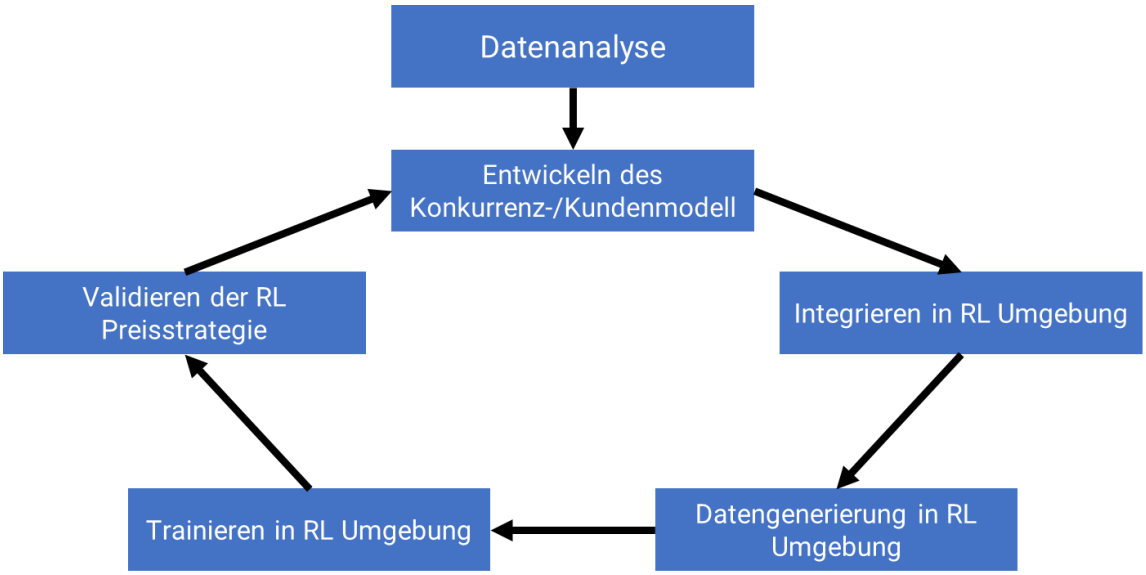
Um den Scope der Arbeit einzugrenzen und die Komplexität der Modellierung zu reduzieren, werden folglich in der Aufgabenstellung folgende Vereinfachungen definiert:

- Das Buchungszeitfenster beträgt einen Tag (folgende Nacht)
- Gebuchte Räume können nicht storniert werden
- Es werden keine Rabatte gewährt

Diese Annahmen erlauben es im Rahmen dieser Arbeit die vereinfachte Problemstellung zu simulieren. Im Rahmen des **Ausblicks** werden mögliche Erweiterung durch grössere Zeiträume und Einbezug von Stornierungen besprochen.

Vorgehensplan

Beim vorliegenden Projekt handelt es sich um ein Innovationsprojekt, welches aus einem grossen Teil an Datenanalyse, Datenaufbereitung und Machine Learning beinhaltet. Beim vorliegenden Projekt sind die Anforderungen und die Machbarkeit des Projektes nur sehr bedingt klar. Deshalb wurde ein exploratives-iteratives Verfahren nach dem Plan-Do-Check-Act Zyklus gewählt. Dabei wurde folgendes Vorgehen entwickelt.



Die Validierung des aktuellen Konzepts wird über bestimmte Metriken (siehe **XXX**), die qualitative Analyse der Preisstrategie und der Besprechung der Ergebnisse mit den Projektmitgliedern sichergestellt.

Dabei wurde in wöchentlichen Meetings mit dem Projektverantwortlichen wurde der Projektstand laufend geprüft und der Informationsfluss sichergestellt. Die Zwischenpräsentation in der Mitte des Projektes ermöglicht es zusätzlich den Projektfortschritt an den Projektverantwortlichen und den Experten mitzuteilen. Dies ermöglicht eine zusätzliche Standortbestimmung und erlaubt Neuausrichtungen im Projekt.

Die Ergebnisse der jeweiligen Experimente werden im Study Doc (siehe **XXX**) dokumentiert. Die Risikoanalyse (siehe **XXX**) und der Meilensteinplan (siehe xxx) kann ebenfalls dem Anhang entnommen werden.

Erste Ideen

In diesem Abschnitt werden die ersten Ideen und Gedanken sowie stichwortartige Diskussionen mit den Forschenden des ABIZ Teams beschrieben.

Offene Fragen

-Was soll alles analysiert werden?
- Wie sollen historische Daten verwendet werden für Reservierungen? -> Man weiss ja die effektive Nachfrage nicht!
- Wie wird die Erklärbarkeit gemessen? (User fragen?, Variablen zeigen?)

Single-Agent vs. Multi Agent

Single Agent! -> Sonst wird es sehr schwierig zu erklären.
- Fixes Ziel: Vorhersage für den nächsten Tag
- Optional: Einbezug von Reservierung im Voraus

Nachfrage

- Nachfrage / Kunden werden statisch definiert
- bietet aber Interface für Dynamik

Evaluierung

- Es wird eine klar definierte Metrik der Effektivität der Preisstrategie definiert und mit dem Auftraggeber besprochen
- Es werden mindestens drei Modelle verglichen (fixer Preis, dynamischer Preis anhand einfacher Regeln, RL Modell)
- Es werden zwei Perioden von maximal einer Woche für ein Hotel für einen Raumtyp evaluiert! Die Wahl der Perioden werden anhand statistischer Merkmale begründet.
- Optional: Hinzufügen von zusätzlichem Raumtyp

Erklärbarkeit optional!

Leisure Time!

Resultate aus Diskussion mit Reza und Daniel

Evaluation

- Overfitting ist Okey
- Kumulativer Reward
- Historical -> How it diverges -> For each Period, what the humans did!

Offline Reinforcement Learning

- Conservative Q-Learning -> RLLIB

Kunden -> Vielleicht aus Reviews!

Modellierung des Hotels

- Hotel aus Bristol (Mit dem häufigsten Zimmer! -> Wo man am meisten Reservationen hat!)
- Echte Werte aus der Website entnehmen (Anzahl Zimmer etc.)
- Modell für Grundkosten (Fixkosten ohne Kunden, linear steigende Kosten pro gebuchtes Zimmer)
- Berechnung der Belegung???
- Cancellation als prozentualer Wert!

Evaluierung

-Zielmetrik: Profit oder Net Margin?
- Vergleich mit anderen Modellen

Modelle

Statischer Preis

- Nimmt Median des ganzen Jahres oder evtl. der Saison
- Keine Anpassung durch Belegung oder Anzahl Zimmer

Dynamisches Modell

- Intelligente Berechnung des Preises anhand von:
  - Belegung
  - Wochenende oder nicht
  - Vielleicht Konkurrenz?
- Room Price Genie Experten konsultieren?

RL Modell

- Simulation durch z.B. Q-Learning Umgebung
- Environment beinhaltet:
  - Konkurrenzpreis
  - Hotelbelegung …

-Muss nachvollziehbar sein!

Konkurrenz

Modellierung durch Median Preis am Vortag. Dieser wird durch die direkte Konkurrenz definiert:

- Latitude
- Longitude
- Evtl. Ort?
- Zimmergrösse
- Sternebewertung

Alternativ:

- Simulieren der Konkurrenz durch ein einfaches ML Modell

Nachfrage

-Wohldefiniertes Interface!
- Am Anfang ähnlich wie im Paper mit solchen User Gruppen

- Andere Möglichkeit, allgemeine Nachfrage simulieren

Framework

- We have an offline Dataset
- Offline Reinforcement Learning is better -> We do not have Environment (We learn from Dataset)
    - We have a dataset!
    - We forget about the
    - Very interesting (State of the Art)
- Online Reinforcement Learning -> Environment (Atari) -> We have a Simulator!
- Reward: Profit
- Action: Price
- States: ?

Steps:

- Predict Consumer Prices!
- Consumer Demand!

Reinforcement: More the Setup of the RL Agent!

Modelling: XGBoost for Consumer Prices!

Evaluation: Compare to historical data!

Datenanalyse

Im vorliegenden Kapitel werden die verfügbaren Datensätze analysiert und bereinigt. Zusätzlich werden offene Fragen geklärt.

Fragen zu Datensets

Datum	Frage	Antwort	Quelle
	Cluster Prices		
02.03.2022	Was sind genau diese Cluster im Clusters Prices Datenset? (Winter & Sommer) (Kategorie der Zimmer, aber wieso pro Saison?)	Gruppiert nach Median Preis, std, mean, … Ähnliche Räume gruppieren -> Normalisieren! 3 Cluster Hoch Preis sehr unterschiedlich -> Lokal stimmt es Eine dieser Clusters ansehen! -> Median NaN -> Perfekt	Daniel Pfäffli
	Konkurrenz		
02.03.2022	Kann man die Hotels aus dem Konkurrenzpreisdatensatz entfernen, welche nicht im Konkurrenzdatensatz sind? (Gewisse Hotels existieren im Customers Datenset nicht)	Gewisse Key Features -> Clusters Sets rauswerfen -> gibt es nicht!	
02.03.2022	Wie wurden genau die Preise gesammelt? Es gibt Hotels mit nur einem Preis und ebenfalls Hotels, welche mehrere Preise pro Tag haben! (Vermutung bei hoher Anzahl: Alle Kombinationen)	Daten alle 3 Tage -> Mehrere Preise für das gleiche Hotels ➔ Median Teilweise nur einmal angeboten Zwei Monaten am Stück vermietet! Konzentrieren auf die mit einigen Preisen pro Woche!	
	Reservationen		

02.03.2022	Sind das unterschiedliche Hotels in Bristol? Eines ist ein paar Autominuten entfernt, die anderen sind mehr oder weniger am gleichen Ort	Zwei verschiedene Hotels	
02.03.2022	Woher kommen diese negativen und gratis Preise bei den Reservierungen?	1709: Cancellation Rate auf 0 -> Aussen vorlassen -Preise = Cancellation Rate (0 oder Minus -> Storniert -> Könnte man ignorieren)	
02.03.2022	Was ist genau der Unterschied zwischen daily_rates_array und dem avg_price?	daily_rates_array = Vergünstigungen nicht dabei -> Nicht so verlässlich	
22.03.2022	Wieso sind immer mindestens drei Tage zwischen den Konkurrenzpreisen?	Crawl Intervall Alle 3 Tage. Wenn jetzt bucht für diese Nacht!	
22.03.2022	Wie würde man sinnvoll mehrere Preise pro Tag pro Raum aggregieren? Ich habe jetzt einfach den neusten genommen.	Minimum Wert ➔ Weil das der letzte Tag ist!	
22.03.2022	Kannst du dir erklären, warum R2 und MAE über die Zeit nicht besser werden?	Schwankung pro Tag, Profitiert nicht von Zeitreihe ➔ Bringt nichts	
22.03.2022	Aufteilung Datenset und Cross Validation	Cross Validation: Sein lassen! Test-Datensatz: Ein Monat!	
22.03.2022	Git Repo Überlauf	Mehr Platz EL, S3	

Erste Einsicht in Daten

Datum: 28.02.2022-02.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: data_exploration/first_look.ipynb

Unterschiede in Hotel Ids der Booking Datensätze

Es gibt in den Prices Clusters zusätzlich 592 Hotels, welche im Konkurrenz Datenset nicht vorhanden sind.
Diese müssen sehr wahrscheinlich entfernt werden.

Orte der Konkurrenz

Ausschliesslich in der UK: Das heisst, es macht wahrscheinlich Sinn, wenn man für die Reservation Daten / zu simulierenden Hotels eines der beiden Bristol Hotels nimmt.

Dabei gibt es auch bereits 151 Hotels in Bristol im Konkurrenzhotelsatz! -> Ohne Bezug auf Longitude and Latitude.

```
competitors[competitors.city_name == "Bristol"]["hotel_id"].size
✓ 0.5s
151
```

Competitor Modell

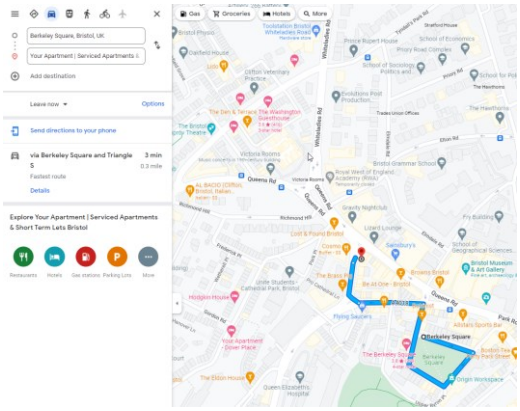
- ➔ Informationen
- ➔ Zeit
- ➔ Etwas über Konkurrenz
- ➔ XGBoost

Reservation Dataset

Man hat dort insgesamt 5 verschiedene Hotels:

ID	Name	Ort	Land	Anzahl Zimmerarten
1795	Greulich Design & Lifestyle Hotel	Zürich	Schweiz	1
1708	The Tryst Beachfront Hotel	Condado	Puerto Rico	1
1706	Hotel Emma	Rotterdam	Niederlande	1
1709, 1527	Your Apartment - Merchant House (Hotel)	Bristol	UK	2
1098, 1099, 1100, 1101, 1102, 1103	Your Apartment – (3 Bed, 2 Bed Upper End, 2 Bed Lower Level, 1 Bed Standard, 1 Bed Premium, Studios)	Bristol	UK	6

1709/1527 und die 1098 – 1103 sind ein bisschen auseinander!



Grundsätzlich macht es sicher Sinn die Hotels aus Bristol zu verwenden. Diese weisen folgende Preise auf:

ID	N	Mean	Std	Min	25	50	75	Max
1709	57590	79.5	79.16	0	0	50	110	1050
1527	51365	105.47	74.35	-209	74	92	118	1050
1098	51101	105.16	71.69	-209	75	94	118	1050
1099	51207	110.26	71.82	-166	77	96	120	1050
1100	48433	106.86	75.77	-209	75	93	117	1050
1101	49571	110.99	71.32	-166	78	95	120	1050
1102	49557	10.32	72.63	-209	75	93	119	1050
1103	48145	105.55	71.40	-131	75	92	117	1050

Beobachtungen

- Die Anzahl Reservationen pro Zimmer / Hotel sind sehr ähnlich
- Bei mehreren Zimmern liegt der Minimal Preis im negativen Bereich!
- Beim Zimmer 1709 Ist der durchschnittliche Price bis auf das 25% Quantil bei 0!
- Der maximale Preis liegt immer bei 1050!

1709: Cancellation Rate auf 0 -> Aussen vorlassen

-Preise = Cancellation Rate (0 oder Minus -> Storniert -> Könnte man ignorieren)

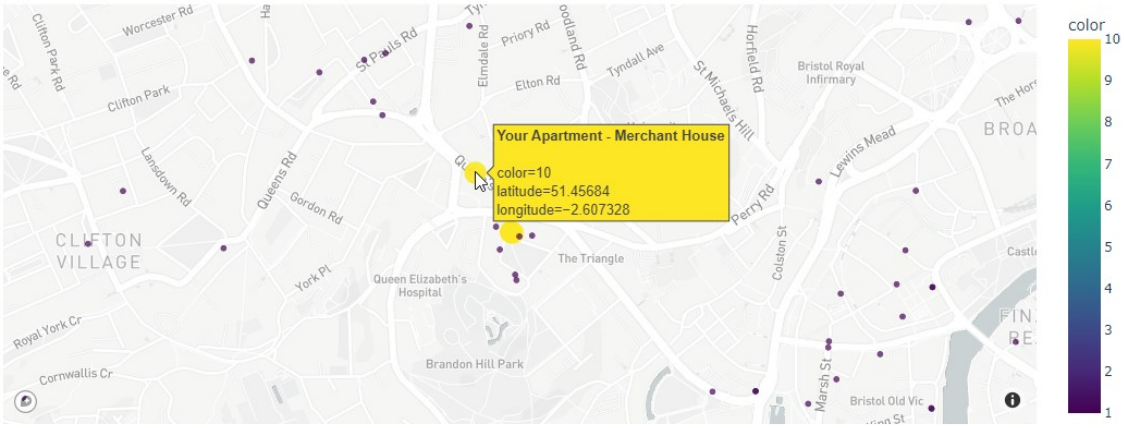
Interessanterweise ist bei den meisten Hotels die Anzahl der Buchungen mit negativen/null Preis zwischen 2 und 7%. Ausser beim 1709 liegt es bei über 30%!

Ebenfalls ist zu beachten, dass pro ReservationID mehrere Zimmer gebucht werden können. Diese haben dann eine andere Room Id.

Zusätzlich wird pro Nacht und Zimmer ein eigener Eintrag gemacht!

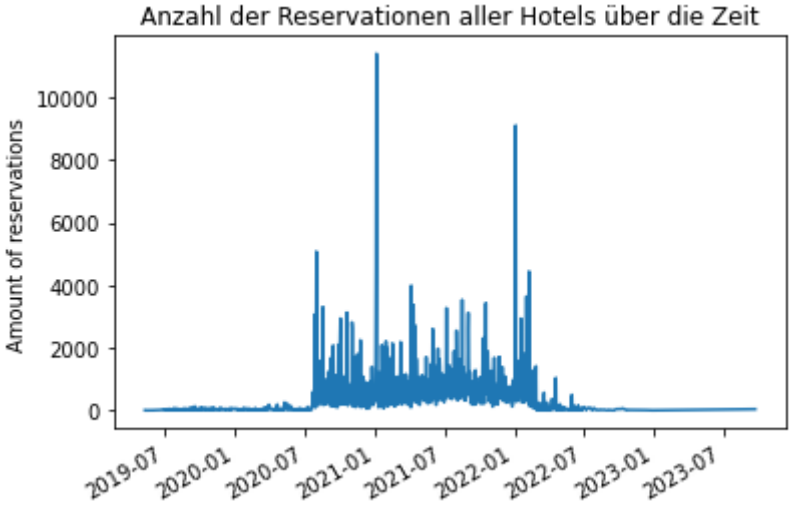
Visualisierung des Reservations und Competitor Datensets

Gelb = Reservations Hotels
Blau = Competitors

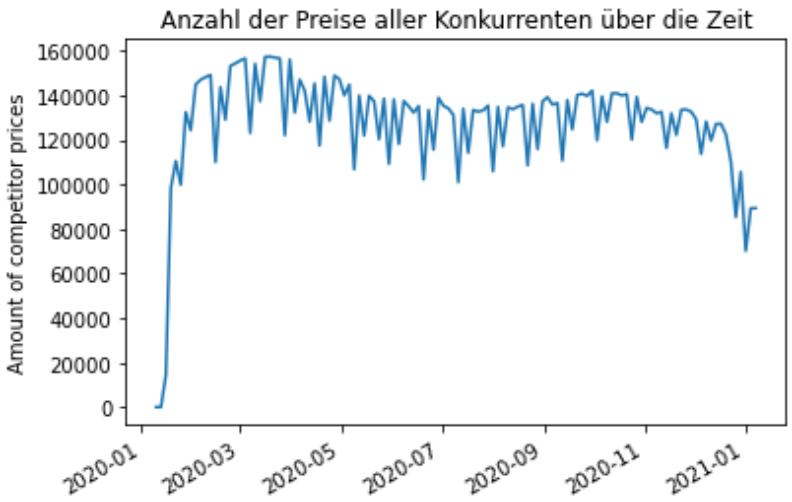


Überschneidungen der Zeiträume

Die Reservationen sind vom Mai 2019 bis September 2023.



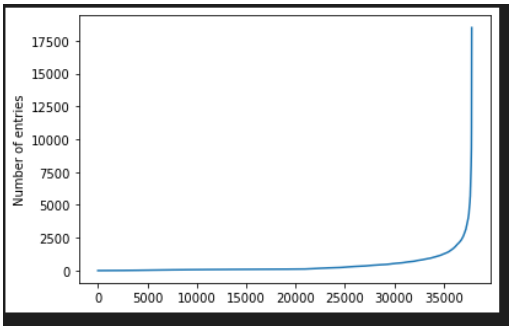
Hingegen die Competitor Prices sind nur vom Januar 2020 bis zum Januar 2021.



Preise pro Competitor

Die Preise pro Competitor variieren sehr stark, so ist es im Minimum nur 1 Preis und manchmal viel mehr!

```
count    37837.000000
mean       410.800222
std        831.232521
min          1.000000
25%         70.000000
50%        109.000000
75%        433.000000
max      18496.000000
Name: room_id, dtype: float64
```



Datensatz bereinigen

Competitor & Competitor Prices

Datum: 14.03.2022/15.03.2022/23.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: data_exploration/cleaning.ipynb

Erste Bereinigung (14.03.2022/15.03.2022 - ecff277b1f001f074d134d08a2c1f7c205c7aec5)

Löschen der Konkurrenzpreise ohne Hotels

Es wurden in einem ersten Schritt Competitor Prices ohne Hotel im Competitor Datenset entfernt, da diese keinen sinnvollen Beitrag leisten können.

Speichern von Hotels mit # Preisen > 4

Damit für die Prediction gute Vorhersagen gemacht werden können, werden nur Hotels und Hotelpreise behalten, bei welchen mind. 4 Preise für ein Hotel zur Verfügung stehen. Dadurch können mindestens dreimal eine Vorhersage mit Last Price gemacht werden. Die Zahl 4 wurde zufällig gewählt.

Die sorgte für folgende Datenreduktion:

Competitors: 37245 -> 35707

Competitor Prices: 15238317 -> 15234938

Zweites Cleaning (22.03.2022)

Doppelte Zeilen entfernen

Im Competitor Prices Cluster Datenset sind teilweise pro Raum, pro Hotel und Datum mehrere Preise aufgeführt. Dies entsteht durch das Intervall des Crawlers. Zusammen mit Daniel Pfäffli und Fabian Gröger wurde entschieden, dass dabei der minimale Preis jeweils behalten werden sollte. Das basiert auf der Annahme, dass der Raum, wenn er verkauft werden würde, dann zum tiefsten Preis verkauft werden würde.

Dies entfernte 8107447 Zeilen, was rund 53.21 % der Daten entspricht.

Konvertierung und Sortierung des Datums

Ebenfalls wurde der String der «date» Spalte in ein datetime Objekt konvertiert. Zusätzlich wurde das Competitor Prices Datenset nach dem Datum mit aufsteigendem Datum sortiert.

Beschränken des Zeitrahmens

Um nur das Jahr 2020 zu betrachten, wurde im Competitor Prices Datenset der Zeitraum der Preise auf 01.01.2020 – 31.12.2020 eingeschränkt.

Dies sorgte erneut für eine Reduktion der Datenmenge um 116804 Zeilen. Wobei vor allem am Ende Daten entfernt worden sind.

Speichern der Daten

Daraufhin wurden die Datensets jeweils im bereinigten Dataset gespeichert!

Reservations

Datum: 30.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: data_exploration/cleaning_reservations.ipynb

Entfernen der Reservationen mit negativem Preis

Zusammen mit Daniel Pfäffli wurde entschieden, dass alle Reservationen mit negativem Preis ignoriert werden können.

Dies sorgte in einem ersten Schritt für eine Reduktion um 68133 Zeilen.

Nicht verwendete Spalten entfernen

Weiter wurden folgende Spalten entfernt: "hotel_desc", "daily_rates_array", "Unnamed: 0", "address"

Sortieren nach Datum

Anschliessend wurde das Datenset nach night_of_stay sortiert.

Unterschiede im Datum erkennen

Datum: 17.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: data_exploration/explore_date_differences.ipynb

Schritte

Es wurde der Tag seit dem letzten Preis aggregiert nach Hotel Id und Room Id betrachtet. Bei beiden Fällen gab es sehr ähnliche Resultate. So gab es Teils für einen Tag mehrere Preise und dann meist mindestens 2 Tage keine Preise.

Resultate

Wenn man nur die von 0 verschiedenen Werte betrachtet und nach Hotel Ids gruppiert, erhält man folgendes Resultat:

```
last_price[last_price.days_since_last_price > 0].days_since_last_price.describe()
✓ 1.4s

count    7.035272e+06
mean     3.730404e+00
std      5.537185e+00
min      3.000000e+00
25%      3.000000e+00
50%      3.000000e+00
75%      3.000000e+00
max      3.540000e+02
Name: days_since_last_price, dtype: float64
```

Erkenntnisse

Es scheint so, als ob die Konkurrenzpreise nur minimal alle drei Tage aufgezeichnet wurden. Dadurch kann z.B. Last Day oder Last Week Price nur sehr umständlich abgebildet werden.

Bestimmung der zwei Wochen Zeiträume

Datum: 17.05.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: data_exploration/two_week_for_evaluation.ipynb

Schritte

Zwei Wochen Intervall mit über 1000 Preisen, welche:

- Tiefste Standardabweichung
- Höchste Standardabweichung

```
df[df.total_prices >= 1000]
```
✓ 0.1s

	night_of_stay	mean_price	total_prices	std_price	min_price	max_price
17	2020-08-17	125.838752	7535	55.676968	34.0	364.0
16	2020-08-03	117.461212	6368	52.621243	23.0	474.0
21	2020-10-12	110.908607	7320	51.233752	5.0	500.0
18	2020-08-31	114.890872	6891	50.081895	34.0	600.0
19	2020-09-14	115.082477	7008	48.966954	35.0	456.0
15	2020-07-20	111.197235	4051	46.532223	37.0	382.0
20	2020-09-28	110.303769	6765	45.564296	32.0	557.0
26	2020-12-21	90.789116	4557	44.472196	21.0	462.0
22	2020-10-26	101.635379	6552	36.823449	17.0	380.0
25	2020-12-07	89.772332	6549	29.900991	33.0	327.0
24	2020-11-23	92.352005	6909	28.235526	32.0	441.0
23	2020-11-09	92.544249	6034	26.391080	38.0	327.0

Somit sind die zwei Wochen mit tiefster Standardabweichung (55.68) vom 09.11.2020 bis am 22.11.2020. Dies macht Sinn, da dann keine Hauptsaison ist.

Jedoch die höchste Standardabweichung (55.68) gibt es in den zwei Wochen vom 17.08.2020 bis am 30.08.2020, wo wahrscheinlich sehr viele Touristen unterwegs sind, und Hochsaison herrscht.

Konkurrenzmodelle

Im folgenden Kapitel wird die Entwicklung und die Evaluation des Konkurrenzmodells beschrieben.

XGBoost

Datum: 09.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_model/xg_boost.ipynb

Installation

sudo pip install xgboost

1. Versuch: Nehme nur Hotels aus Bristol und sage zeitunabhängig den Median Preis voraus

Beschreibung

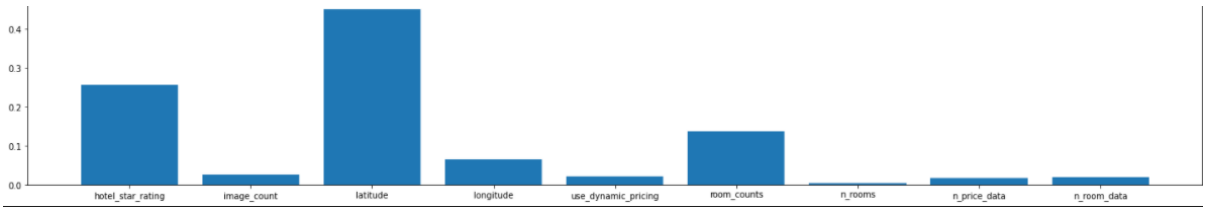
In diesem Versuch wurde nur das Konkurrenz Datenset für den Ort Bristol verwendet. Dabei wurden folgende Features definiert und damit den price_median für diese Hotels vorhergesagt.

Features: 'hotel_star_rating', 'image_count', 'latitude', 'longitude', 'use_dynamic_pricing', 'room_counts', 'n_rooms', 'n_price_data', 'n_room_data'

Resultate

Bei einer Cross Validation wurde ein MAE von 101.743 erzielt mit einer sehr hohen Standardabweichung von 86.098. Dies ist nicht sinnvoll, da der Median Preis sowieso nur bei etwa 145.9 liegt.

Ebenfalls gab die Feature Wichtigkeit Auswertung beim Training auf den ganzen Trainingsdaten nicht sinnvolle Resultate. So war das wichtigste Feature der Breitengrad des Hotels. Was nur sehr bedingt sinnvoll ist!



Fazit

Dies gibt nicht sinnvolle Resultate. Das liegt sehr wahrscheinlich an zwei Faktoren:

1. Zu wenig Daten (nur 151 Zeilen)
2. Nicht sinnvolle Features

Folglich können diese anderen Ansätze versucht werden:

1. Zusätzliche Features generieren anhand des Competitor Datensets (z.B. Most_popular_facilities)
2. Einbezug des Competitor Prices Datensets
3. Erweiterung auf ausserhalb von Bristol

2. Versuch: Zusätzliche Features aus dem Competitor Datenset erzeugen

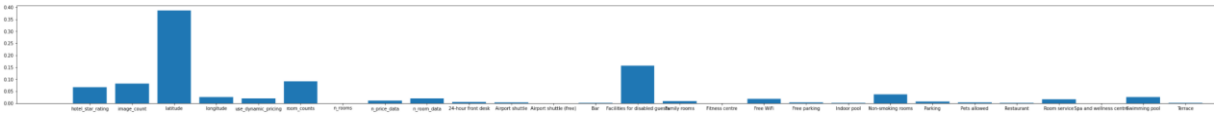
Beschreibung

In diesem Versuch werden analog zum 1. Versuch oben nur das Competitor Datenset verwendet. Es werden aber noch zusätzliche Features erzeugt. Dabei wurden aus dem most_popular_facilities Feature one-hot Features erzeugt.

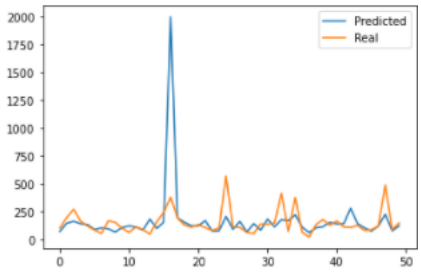
Features: 'hotel_star_rating', 'image_count', 'latitude', 'longitude', 'use_dynamic_pricing', 'room_counts', 'n_rooms', 'n_price_data', 'n_room_data', '24-hour front desk', 'Airport shuttle', 'Airport shuttle (free)', 'Bar', 'Facilities for disabled guests', 'Family rooms', 'Fitness centre', 'Free WiFi', 'Free parking', 'Indoor pool', 'Non-smoking rooms', 'Parking', 'Pets allowed', 'Restaurant', 'Room service', 'Spa and wellness centre', 'Swimming pool', 'Terrace'

Resultate

Auch hier konnte mit Cross Validation nur ein MAE von 101.54 mit einer Standardabweichung von 85.453 erzielt werden. Ebenfalls lag der R2 Wert bei -10.636, was nicht für eine gute Performance spricht. Die Signifikanz der Features war ebenfalls nicht viel vielversprechender. So blieb der Breitengrad das wichtigste Feature, wobei jedoch nun zusätzlich relevant war, ob es Infrastruktur für Behinderte gibt.



Wenn eine Prediction für den Testdatensatz gemacht wird, dann erhält man folgende Resultate:



Hier wird ein R2 von -3.96 erzielt und der MSE liegt bei 31244.97. Das heisst, das Modell ist schlechter, als wenn man einfach der Mittelwert nehmen würde.

Fazit

Das Modell performt nur sehr gering besser als das Modell, welches weniger Variablen als Datengrundlage nimmt. Aus diesem Grund wird in einem nächsten Experiment das Konkurrenz Datenset miteinbezogen.

3. Versuch: Preise mit Hilfe des Competitor Cluster Prices Vorhersagen (ohne Zeitkomponente)

Beschreibung

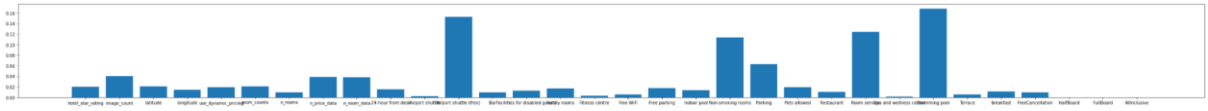
In diesem Versuch wird versucht der durchschnittliche Preis einer Übernachtung in Bristol vorherzusagen. Als Features werden zusätzlich zu den Features aus dem zweiten Versuch noch mehr Features miteinbezogen. In einem ersten Schritt wird der zeitliche Aspekt noch weggelassen und es wird nur versucht, ob zumindest bessere Vorhersagen gemacht werden durch die zusätzlichen Daten. Hier wird jedoch jeweils der effektive Preis aus dem Konkurrenzpreise Datenset als y verwendet, anstatt der Median Preis.

Zusätzliche Features: 'FreeCancellation', 'HalfBoard', 'FullBoard', 'AllInclusive', 'cluster_winter', 'cluster_summer'

Resultate

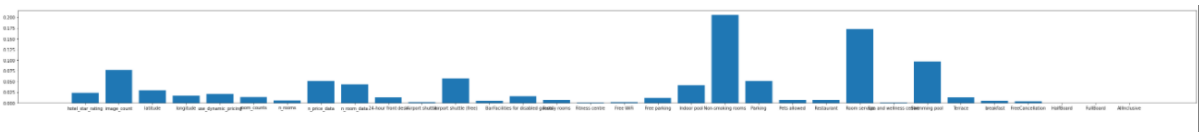
Hiermit konnte bei einem Trainingsset von 20% der Daten ein R2 Wert von ca. 0.559 bei der Cross Validation erzielt werden. Ebenfalls ist der MAE einiges kleiner, so ist im Mittel die Abweichung nur noch bei etwa 35.786 mit einer Abweichung von 1.51. Damit kann schon einigermassen zuverlässig der Preis vorhergesagt werden.
Das Problem ist hierbei aber, dass die signifikantesten Features unter anderem cluster_winter und cluster_summer sind.

Dies ist logisch, da diese unter anderem aufgrund des Preises definiert wurden. Dadurch gibt es eine natürliche Korrelation. Die Werte auf dem Test-Datensatz sind die folgenden: R2: 0.67 und MAE: 35.58.Die Performance bleibt jedoch auch gut, auch wenn man diese beiden Spalten entfernt, man erhält dann ein R2 von 0.555 und einen MAE von 36.235 bei einer Abweichung von 1.563.



Hier ist dann auch spannend zu sehen, dass dann non-smoking-rooms, room-service und swimming-pool wichtiger werden. Ebenfalls ist sichtbar, dass Halb-Pension, Voll-Pension und All-Inclusive keinen Einfluss haben.

Die Performance auf dem Test-Datensatz ist sogar besser und liegt bei: R2: 0.63 und MAE: 36.18.Diese Performance soll nun auf einem grösseren Datenset validiert werden. Dazu wird jetzt 80% der Daten als Trainingsset verwendet. Die Cluster bleiben erneut aussen vor. Bei der Cross Validation wird nun ein R2 von 0.623 und ein MAE von 36.14 mit einer Standardabweichung von 0.724 erzielt. Das heisst, die Performance ist akzeptabel. Auf dem Test-Datensatz wird ein R2 von 0.604 und ein MAE von 35.06 erzielt.



Fazit: Die Performance ist hier einiges besser. Dies liegt vermutlich hauptsächlich an den zusätzlichen Daten, die zur Verfügung gestellt werden.

4. Versuch: Preise mit Hilfe des Competitor Cluster Prices Vorhersagen (ohne Zeitkomponente)

Beschreibung

In diesem Versuch wird versucht der durchschnittliche Preis einer Übernachtung vorherzusagen. Als Features werden zusätzlich zu den Features aus dem dritten Versuch noch zeitliche Features hinzugenommen. Hierbei ist jedoch zu beachten, dass man für die Cross-Validation nun eine zeitliche Abhängigkeit hat.

Zusätzliche Features: 'year', 'month', 'day', 'day_of_year', 'day_of_week', 'quarter', 'is_weekend'

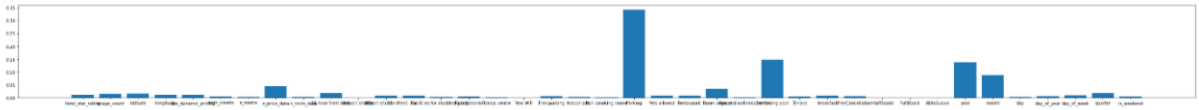
Resultate

Hier scheint es teils gute/ teils schlechte Resultate zu geben auf 80% Trainingsdaten, da wahrscheinlich zu wenig Daten für ein ähnliches Szenario vorhanden sind, bzw. die Anzahl der Splits (n=30) zu klein ist, bzw. das Test-Datensatz dann zu gross ist. Ebenfalls wird hier zwangsläufig die Reihenfolge beibehalten! Dies sorgt für weniger Data-Snooping. Der R2 Wert liegt dabei bei -0.088 mit einer sehr hohen Standardabweichung von 0.860. Der MAE aber ist relativ tief und auf dem Cross Validation Set wird ein Wert von 36.786 mit einer hohen Standardabweichung von 25.376 erreicht.

Tipp: Andere Stadt als Datengrundlage -> Man sieht ein bisschen wie sich das Jahr verhaltet!

Jeden Tag! -> Mehr Features!

Spannend ist auch zu sehen, dass das Jahr und der Monat wichtige Features sind. Jedoch der Tag der Woche ist entgegen den Erwartungen unwichtig!



Auf dem Test-Datensatz wird eine sehr ähnliche Performance von R2 -0.06715 und ein höhere MAE von 74.36 erreicht. Hier ist jedoch hauptsächlich das Problem, dass bei den letzten Datenpunkten gar keine Referenz mehr besteht!

Wenn jetzt die Grösse der Prediction in der Validierung reduziert wird auf 30, auch dann liegt der R2 bei tiefen 0.063. Der MAE senkt sich jedoch auf sehr tiefe 11.173 mit einer Standardabweichung von 2.172.

Fazit

Wenn zeitabhängig vorhergesagt wird, dann ist die Performance sehr schwach. Dies muss in einem weiteren Schritt näher betrachtet werden.

Feedback Daniel Pfäffli: 10.03.2022

Wenn man Prediction über 40 Tage -> Eigene Prediction als Input

Nicht für jeden Tag jedes Hotel ein Preis -> Aufteilen -> Vom letzten Tag fehlt dann evtl.

- ➔ NN -> Masken (NaN auf unrealistischer Preis -> Modell ignoriert es) (Tage nicht für R2 rechnen -> Weil Daten gefehlt haben -> Dies ignorieren)
- ➔ Ansehen wie viel dann effektiv diese NaN Werte haben
- ➔ Last known Price + Anzahl Tage seit letztem Preis / oder zwei letzte Preise
- ➔ Oder kombinieren -> NaN zuerst versuchen! -> Dann last known Price mit 1 -> Dann Kombination!
    - Evtl. Tree ansehen! -> Überprüfen was für Regeln es gibt!

Preise letzter Tag / letzte Woche / Letzter Monat miteinbeziehen

Evtl. Median Preis vom Hotel hinzufügen bzw. Ort

Wie viele Preise gibt es pro Hotel/Tag

Competitor Modell -> Darf Preis von gestern wissen -> Mehrere nicht nur gestern, sondern ein Monat, …

R2, MAE -> Über Zeit betrachten -> Aear under Curve als Vergleich -> Nicht so hohe Standardabweichung

Nach Monat betrachten!

Zeitfenster

- ➔ Corona Flag! -> UK Lockdown Flag
- ➔ Quartal nehmen! / Winter / Summer
- ➔ Wochenend / Jahreszeit
- ➔ Monat eher weniger
- Cleaning
- Mehr Features
- Testen

Neue Features

- ➔ Business oder Freizeit Hotel -> Anhand von
- ➔ Similariät zu Keywords

Konzept

Ein XG Boost Modell für die ganze Konkurrenz!
Zum Training werden jeweils nur die Tage jeweils zuvor verwendet. Dabei wird für die letzten sieben Tagen die Preise vorhergesagt.

Median Hotel nehmen als Prediction, bzw. evtl. eine leichte Varianz um eine Standardabweichung!
Oder Random samplen aus den x Nachbarn!

XGBoost v2

Datum: 14.03.2022 / 15.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_model/xg_boost_2.ipynb

Hier wurden die Punkte aus dem Feedback von Daniel Pfäffli angenommen und getestet. Dabei wurde statt Bristol Liverpool als Datengrundlage genommen. Dies dient dazu, dass kein Datasnooping passiert.

Versuch: Resultate auf Liverpool Set

Beschreibung

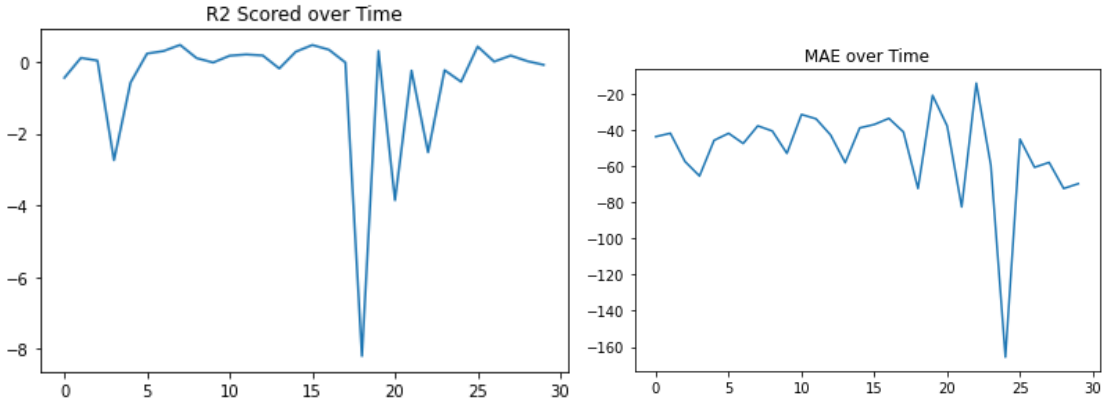
In diesem Versuch wurde sehr ähnlich vorgegangen wie im 4. Versuch oben! Jedoch wurde das Datenset aus Liverpool verwendet und die Anzahl der Time Series Features reduziert und Redundante Flags wie Wochentage und Weekend zusammengenommen.

Features: 'hotel_star_rating', 'image_count', 'latitude', 'longitude', 'use_dynamic_pricing', 'room_counts', 'n_rooms', 'n_price_data', 'n_room_data', '24-hour front desk', 'Airport shuttle', 'BBQ facilities', 'Bar', 'Beachfront', 'Facilities for disabled guests', 'Family rooms', 'Fitness centre', 'Free WiFi', 'Free parking', 'Indoor pool', 'Lift', 'Non-smoking rooms', 'Parking', 'Pets allowed', 'Restaurant', 'Room service', 'Spa and wellness centre', 'Swimming pool', 'Terrace', 'WiFi', 'breakfast', 'FreeCancellation', 'HalfBoard', 'FullBoard', 'AllInclusive', 'quarter', 'day_of_week'

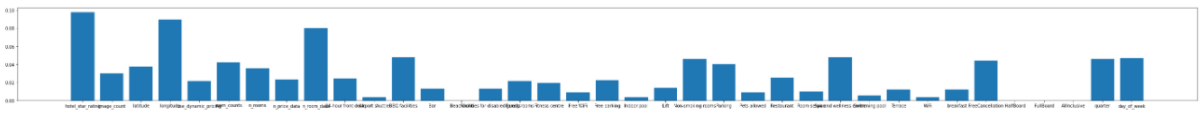
Resultate

Auch in diesem Fall wurde keine gute Performance (Test-Datensatz 0.8, split: 30) erreicht: Auf dem Cross Validation Set wurde eine Performance von R2 von -0.531 und eine MAE von -51.85 mit einer hohen Schwankung von 26.135 erreicht.

Ebenfalls wurde die Performance über die Zeit nicht besser:



Von den signifikanten Features sind der Tag und das Quartal relativ relevant



Fazit

Auch auf den Daten von Liverpool kann ohne zusätzliche Preisinformationen kein zuverlässiges Resultat erzielt werden.

Versuch: Corona Flag

Beschreibung

In einem nächsten Versuch wurde mit einem Corona Flag die Lockdowns in der UK markiert und überprüft, ob so zusätzliche Informationen hinzugekommen sind.

Features

Analog wie oben + Folgendes Lockdown Flag:
Als Dauer wurde folgende Lockdowns gesetzt:

26.03.2020 – 14.08.2020

22.09.2020 – 02.12.2020

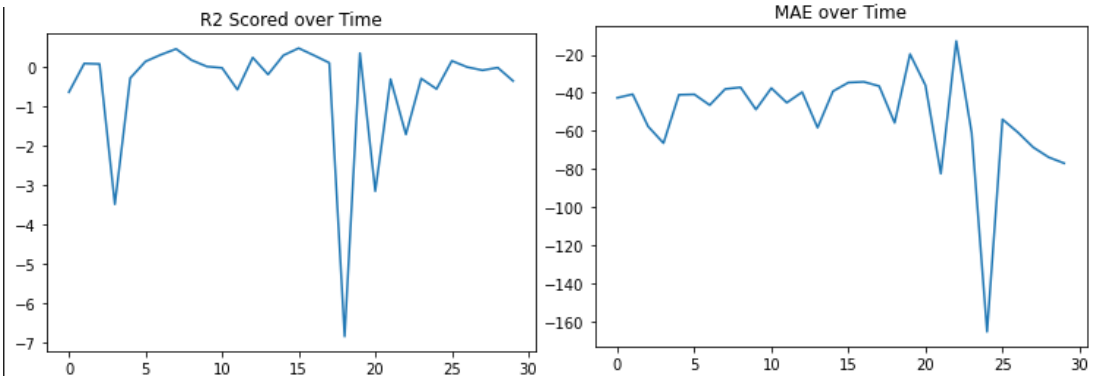
Dabei wurden diese anhand des folgenden Links definierts:
https://www.instituteforgovernment.org.uk/sites/default/files/timeline-lockdown-web.pdf

Dabei wurden vor allem die Einschränkungen in der Bewegungsfreiheit und die Öffnung von kulturellen Stätten betrachtet.

Resultate

Auch in diesem Fall wurde keine gute Performance (Test-Datensatz 0.8, split: 30) erreicht: Auf dem Cross Validation Set wurde eine Performance von R2 von -0.506 und eine MAE von -51.77 mit einer hohen Schwankung von 36.37 erreicht.

Ebenfalls wurde die Performance über die Zeit nicht besser:



Man sieht, dass das Corona Flag zwar relevant ist, aber nicht so signifikant wie andere Features!

Fazit

Es ist spannend zu sehen, dass das Corona Flag nur sehr begrenzt sinnvolle Resultate erzielt!

Versuch: Letzter verfügbarer Preis

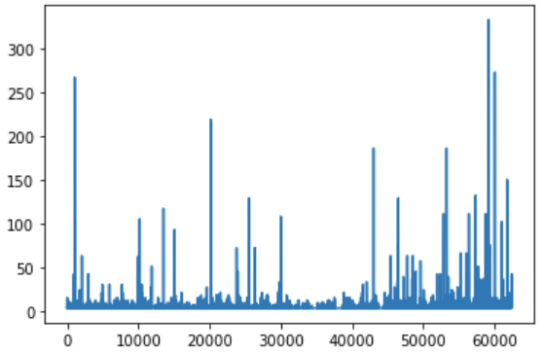
Beschreibung

In diesem Beispiel wird der letzte verfügbare Preis eines Hotels und die Anzahl Tage seit dem letzten Preis als Feature gespeichert.

Features

Wie oben im 1. Versuch und zusätzlich wurde noch last_price und days_since_last_price definiert. Dabei wurde jeweils bei doppelten Preisen für einen Raum innerhalb eines Tages der erste Tag behalten und der andere weggeworfen, da sonst der vorherige Preis auf dem Preis desselben Tages basiert.

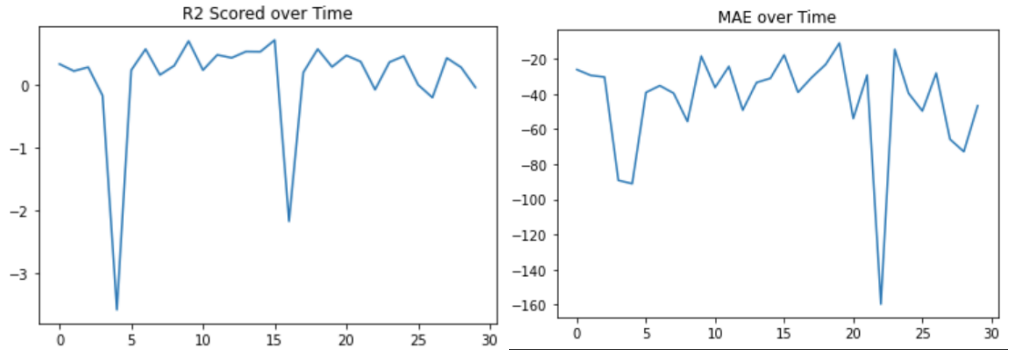
Dabei ist spannend zu sehen, dass es sehr grosse Unterschiede im days_since_last_price gibt:



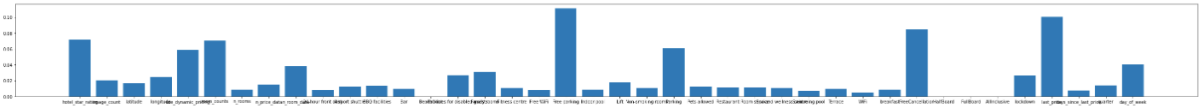
```
count    62392.000000
mean         3.722160
std          4.875331
min          3.000000
25%          3.000000
50%          3.000000
75%          3.000000
max        333.000000
Name: days_since_last_price, dtype: float64
```

Resultate

Es kann hiermit immerhin wieder ein positiver R2 Wert von 0.094 erzielt werden. Ebenfalls liegt der MAE in einem Rahmen von -43.758 bei einer Standardabweichung von 28.952.



Ebenfalls sieht man sehr stark, dass der Preis des letzten Tages signifikant wichtig ist für den Preis.



Jedoch hat auch Free Parking einen sehr starken Einfluss, was eher verwunderlich ist.

Fazit

Durch den Last Price werden schon signifikant zusätzliche Informationen ermöglicht. Damit kann das Modell schon verbessert werden. Jedoch sind die Resultate noch nicht zufriedenstellend. Deshalb wird mit dem Preis des letzten Tages, der letzten Woche und des letzten Monats zusätzliche Informationen bereitgestellt.

Versuch: Preis des Vortrages, der Vorwoche und des Vormonats

Beschreibung

In diesem Fall wird anstatt des letzten Preises den Preis des Vortages verwendet. Falls dieser nicht vorhanden ist, dann wird NaN gesetzt.

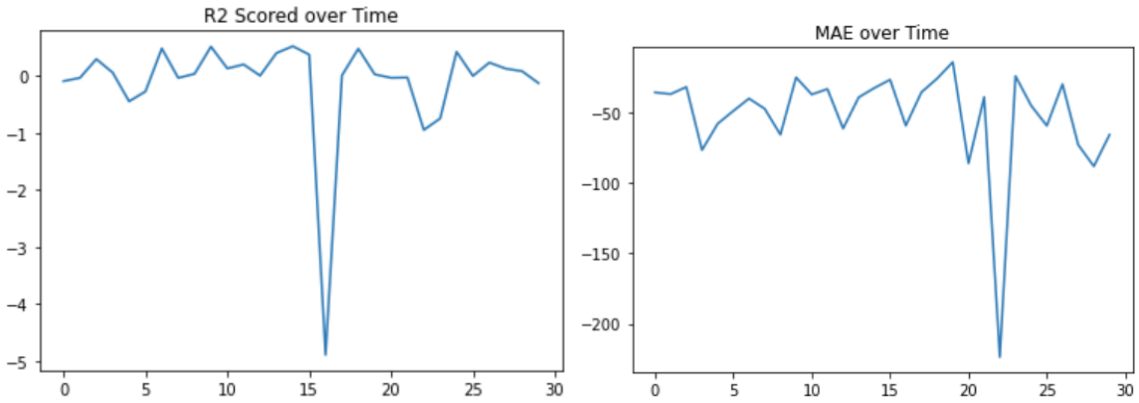
Features

Wie zuvor und zusätzlich ‘last_day_price’ (1d), ‘last_week_price’ (7d), ‘last_month_price’ (30d).

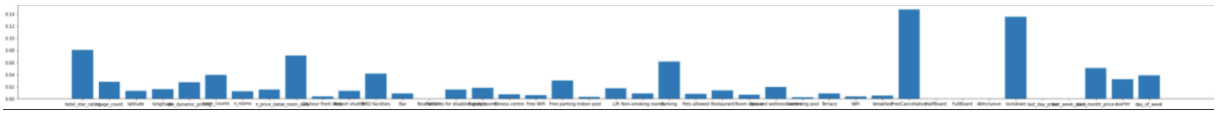
Resultate

Hier war jedoch in der Folge das Problem, dass aggregiert pro Room Id nur immer alle 3 Tage Daten vorhanden waren. Das heisst es wurde meistens für last day und last week NaN gesetzt.

Der Mean R2 ist nicht wirklich zuverlässig-0.103 (0.951). Ebenfalls gibt es einen sehr hohen MAE von -52.519 (36.903). Auch ist keine Verbesserung im R2 und MAE über die Zeit sichtbar.



Man sieht auch, dass die last_day_price und last_week_price keine Verbesserung erreicht. Auch der last_month_price ist nicht das wichtigste Feature.



Schauen, ob es wirklich nicht verfügbar ist!

Fazit

Das Resultat ist nicht so wie erwartet. Ebenfalls ist die Verfügbarkeit der Preise nicht ausreichend. Deshalb müssen in einem nächsten Schritt die Daten noch einmal genauer betrachtet werden.

Feedback Daniel Pfäffli 22.03.2022

Hyperparameter Tuning CVGridSearch,

Min, Max, Median, Konkurrenz 1km-> Vorheriger Preis

1: Ganze Zeitreihe ausser Bristol bis Ende November-> Dann Bristol (Von andern Orten zu unseren Orten) oder nur Dezember (Zeitabhängigkeit), Ein Teil von London (Ort)!

Überprüfen auf Overfitting (Training: R2).

One Hot Encoded: Features! Hotels die kein Dynamic Pricing haben subsamplen beim Training (Max. 0.5).

Attractions: Nearest Attractions!

Ort: Sehr schwierig: In London ein Set von Hotels aussen lassen!

Threshold 0.6: Als One Hot Encoded mit Features!

Bereinigung: Datenset Minimum Preis behalten

Nachfrage

Anzahl Reservationen und Preis!

Aus allen Hotels!

XGBoost v3

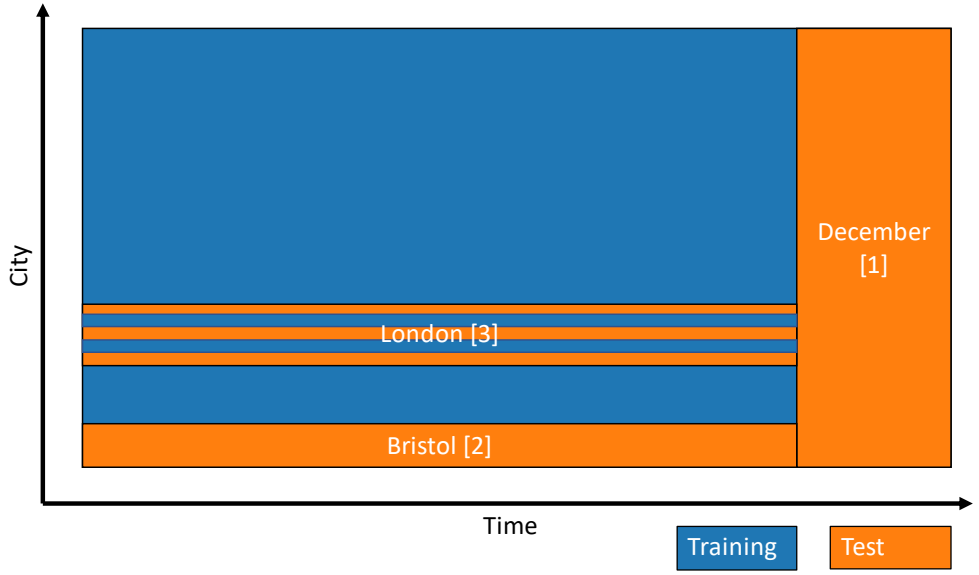
In diesem Fall wurde erneut mehrere Erkenntnisse aus den vorherigen Experimenten umgesetzt. Folgende Punkte wurden hauptsächlich betrachtet:

- Splitting der Daten
  - Dezember für Test-Datensatz
  - Bristol als Test-Datensatz
  - Teil von London als Test-Datensatz (Erklärung siehe unten)
  - Cross-Validation wie gehabt mit unterschiedlichem Split (Reduktion auf 11)
- Verschiebung der Allgemeinen Funktionalität in externe Files
- Zusätzliche Hotelfeatures aufgrund von Text-Ähnlichkeiten
- Last Price aufgrund der Nachbarschaft (Median, Min, Max)
- Gridsearch der Parameter

Splitting der Daten

Datum: 23.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_model/splitting.ipynb

Die Competitor/Competitor Prices Daten wurden auf verschiedenen Ebenen getrennt, um so eine sinnvolle Aussage über die Performance der Modelle machen zu können.



Die Fläche der Balken ist nicht repräsentativ und dient einzig der Visualisierung.

ID	Daten	Beschreibung	Ziel
0	Trainingsdaten	Trainingsdaten, bei welchen in einem Time Series Cross Validation Split mit Grösse 11 die Daten von Jan. – Nov.2020 trainiert werden. Ohne die Testdaten unten!	Rollende Performance beurteilen.
1	Dezember	Testdaten, aus dem Dezember.	Performance in der Zukunft beurteilen.
2	Bristol	Testdaten von Januar bis November, wo der Ort Bristol eingetragen ist.	Performance auf anderen Ort beurteilen

3	London	Daten von Januar bis November 50%: Der Hotels: Testdaten 50%. Der Hotels: Trainingsdaten	Performance: Generalisierung innerhalb eines Ortes beurteilen

Es wurde bewusst keine Überschneidung der Daten erlaubt. Dadurch kann eine bessere Aussage gemacht werden und der Dezember hat keinen Einfluss

Feature: Description

Datum: 23.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: feature_generation/feature_hotel_description.ipynb

Beschreibung

Hier wurde aufgrund der Similaritätsdaten des ABIZ Teams zusätzliche Beschreibungen für das Hotel erzeugt. Das Ursprungsdatenset wurde durch das ABIZ Team zur Verfügung gestellt und zeigt die Similarität zwischen einem Begriff und der Beschreibung eines Hotels dar. Dies soll eine akkurate Beschreibung des Hotels ergeben.

Vorgehen

Für die Umwandlung zu den Features wurden sowohl für das Competitor Prices Datenset, wie auch für das Reservationen Datenset die Similaritäten (0.0 -1.0) in One Hot Vektoren Daten konvertiert. Dazu wurde der Threshold von 0.6 verwendet. Dieser wurde durch das Team vorgegeben und soll als Richtwert dienen, ob dieser Begriff das jeweilige Hotel beschreibt. Folglich sind die Similaritäten >= 0.6 als 1 und alle anderen Werte als 0 encodiert.

Feature: Last Neighbour Prices

Datum: 23.03.2022-25.03.2022 (ca.- 8h)
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: feature_generation/feature_last_neighbours_price.ipynb

Beschreibung

Im vorliegenden Fall wurden zusätzliche Features generiert anhand der Preise des letzten Tages der umliegenden Hotels. Dies hat zwei Gründe:

1. Zusätzliche Preisinformationen für bessere Resultate
2. Simulation, dass Konkurrenz auch auf Konkurrenz reagiert

Dabei wurde die Nachbarschaft relativ einfach definiert und man betrachtet ausschliesslich einen Radius von 1km rund um das Hotel als Nachbarschaft. Dieser Radius wurde zusammen mit Daniel Pfäffli definiert und ist arbiträr. Möglichkeiten zur Verbesserung sind die folgenden:

- Wissenschaftliche Bestimmung des Radius aufgrund von echtem Verhalten in der Realität
- Gridsearch zur Bestimmung des Radius
- Konkurrenz zusätzlich anhand von Similarität bestimmen

Vorgehen

Für jedes Hotel wurde für jeden Tag an dem Preisdaten vorliegen folgende Preise des Vortages der Konkurrenz bestimmt:

- Minimaler Preis
- Median Preis

- Maximaler Preis

Anhand der Längen- und Breitengraden wurde für jedes Hotel die Konkurrenz bestimmt. Dazu muss die Haversine Distanz zwischen den Punkten <= 1km sein. Dies wird Brute Force gerechnet. Dies ist sehr aufwändig, die Nachbarschaften zu berechnen.

Es wurde ein JSON mit den Nachbarschaften berechnet, dies dauerte rund 5 Stunden. Anschliessend mussten auch noch die Preise aller Nachbarn für den Vortag, bzw. die 3 Tage davor berechnet werden. Hier war wieder das Problem, dass die Daten für den Vortag auch bei den Nachbarn nicht vorhanden waren.

Deshalb entschied sich der Autor für den Preis vor 3 Tagen aller Nachbarn und dabei wurde der Minimal, der Median und der Maximalpreis berechnet.

Schwierigkeiten

Hier gab es erneut sehr grosse Schwierigkeiten, da die Datensets sehr gross sind. So war sowohl das Gitlab Repository wie auch die lokale Festplatte der Virtuellen Maschine zu klein. Diese HW Probleme kosteten rund 4 Stunden Arbeit.

Sobald diese Probleme behoben wurden, ging es weiter damit die Preise der Nachbarn effizient zu bestimmen. Dazu musste über das vollständige Competitor Datenset iteriert werden.

Dabei wurde von Zeit zu Zeit einige Verbesserungen für die Performance vorgenommen:

- Wechsel von For-Loop zu Iter-Items
- Versuch zu Parallelisieren (kein Geschwindigkeitsgewinn)
- Setzen eines Multiindex auf Date und Hotel ID
- Verwenden von df.apply statt iteritems .
- Verwenden von mApply -> Parallelisieren!

Trotzdem dauerte die Berechnung aller Zeilen > 7Mio.

Dies stürzte jedoch einmal über das Wochenende ohne eine Bemerkung ab!

Wie es aussieht!

Hyperparameter Optimierung erster Versuch

Datum: 25.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_models/hyper_params_generator.ipynb

Beschreibung

Um bessere Hyperparameter zu erreichen, wird mit einfachen Features einmal die Hyperparameter Optimierung versucht.

https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html

Probleme

Ein Problem gab es mit dem Memory es wurde folgende Fehlermeldung geworfen:

Mar 28 14:00:54 rl-for-dynamic-pricing kernel: [423997.983541] Out of memory: Killed process 80159 (python3) total-vm:29101256kB, anon-rss:26381304kB, file-rss:2588kB, shmem-rss:24kB, UID:1004 pgtables:52456kB oom_score_adj:0

In einem weiteren Versuch ist die VM einfach abgestürzt!

Lösung

Die Anzahl Jobs die parallel laufen wurden auf 1 reduziert, damit das Datenset nicht mehrmals kopiert wird. Zusätzlich wurde die n_estimators auf maximal 150 reduziert, damit erneut die Datenmenge reduziert wird. Ebenfalls wurde die Anzahl Fits reduziert.

Ebenfalls ist aufgefallen, dass obwohl mit nohup gestartet wird das Ausführen des Scripts trotzdem gestoppt wird, wenn man sich ausloggt. Deshalb musste mit screen eine Session gestartet werden.

screen -dmS hyperParam

screen -r

nohup python3 -u /hdd/workspace/bt-rl-dynamic-pricing-dataset/competitor_model/hyper_parameter_tuning.py > hyper_param_1_150.log 2>&1 & disown

Ebenfalls ist es bei einer Erhöhung auf 200 Estimators und 25 depth erneut ausgetimet.

Parameter

Mit diesen Parametern konnte das Hyperparameter Tuning schlussendlich erfolgreich durchgeführt werden.

```
hyperparameter_grid = {
  'n_estimators': [100, 150, 175],
  'max_depth': [10, 15, 20],
  'learning_rate': [0.05, 0.1, 0.15, 0.20],
  'min_child_weight': [3, 4]
  }
```

Features

Folgende Features wurden dabei verwendet:

['hotel_star_rating', 'image_count', 'latitude', 'longitude', 'use_dynamic_pricing', 'room_counts', 'n_rooms', 'n_price_data', 'n_room_data', 'breakfast', 'FreeCancellation', 'HalfBoard', 'FullBoard', 'AllInclusive', 'lockdown', 'last_price', 'days_since_last_price', 'quarter', 'day_of_week', 'last_month_price', '24-hour front desk', 'Bar', 'Facilities for disabled guests', 'Family rooms', 'Free WiFi', 'Free parking', 'Heating', 'Lift', 'Non-smoking rooms', 'Parking', 'Pets allowed', 'Restaurant', 'Terrace']

Beste Hyperparameter

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, Wichtigkeit_type=None,
             interaction_constraints='', learning_rate=0.05, max_delta_step=0,
             max_depth=20, min_child_weight=3, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=1,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='approx', validate_parameters=1, verbosity=None)
```

Resultate mit dem Hyperparameter Model

Datum: 25.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_models/hyper_params_generator.ipynb

Beschreibung

Mit dem Hyperparameter Modell, welches oben beschrieben wurde eine Cross Validation ausgeführt. Dies wurde einerseits einmal mit den Description Features gemacht und einmal ohne.

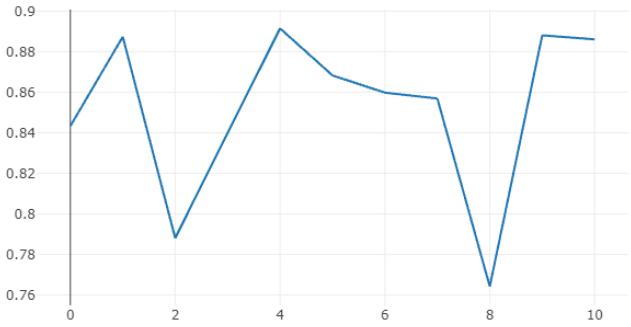
Probleme

Auch hier gab es erneut Out-Of-Memory Fehler. Dies geschah dadurch, dass der Crossvalidator mit n_jobs = -1 gestartet wurde. Dadurch wurde es 16-mal parallel ausgeführt, wodurch auch das Datenset 16 mal kopiert wurde. Dies sorgte für einen Überlauf im RAM. Deshalb wurde die n_jobs Variable auf 1 gesetzt. Dies war dann erfolgreich.

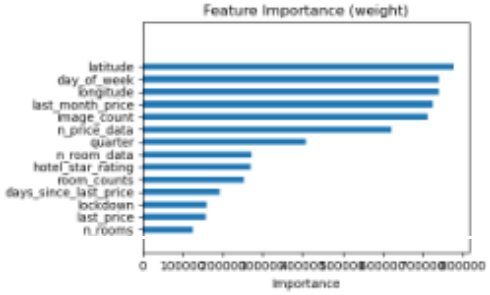
Resultate

Ohne Description

Auf dem Cross-Validation Set konnten folgende R2 Werte über die Zeit erzielt werden:



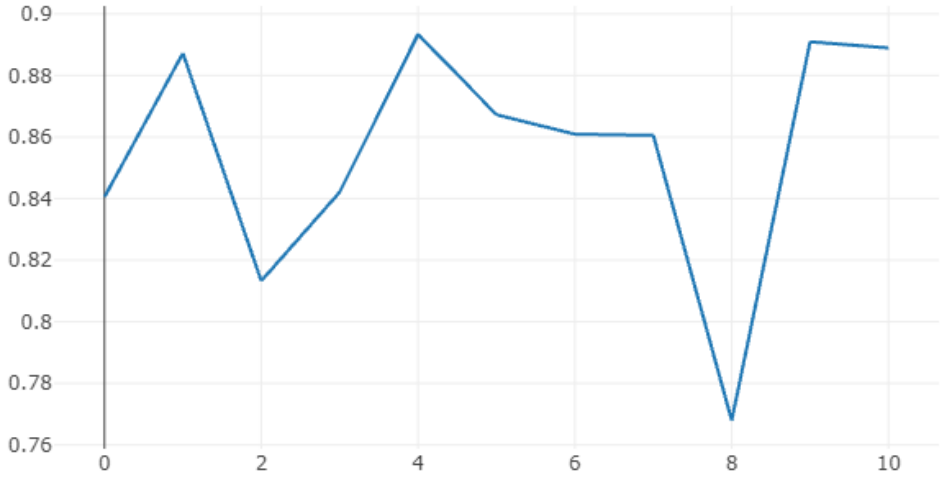
Am Ende wurde ein R2 von 0.886 erreicht, was für eine akzeptable Performance spricht.



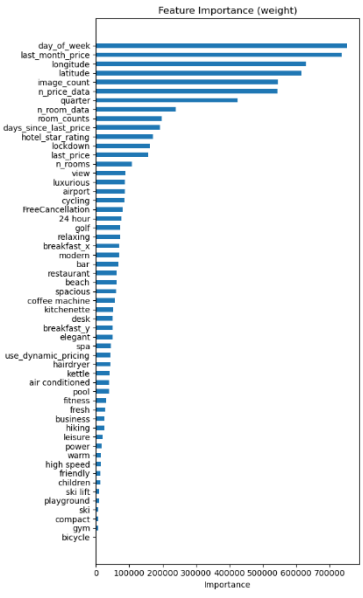
Es ist klar ersichtlich, dass vor allem die latitude und der day of week signifikant wichtig für Bestimmung des Preises ist.

Mit Description

Mit dem Hinzufügen der Description anstatt der Most Popular Facilities kann folgende Performance erreicht werden: Hier wird ebenfalls ein R2 von 0.89 erreicht.

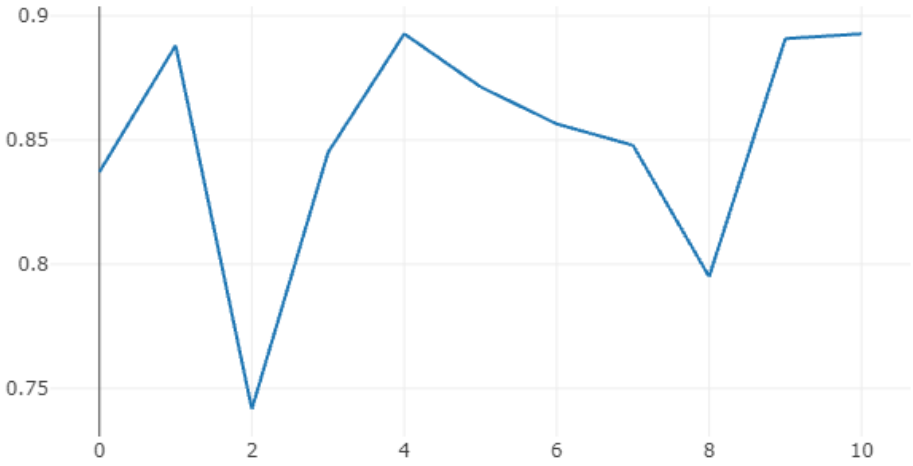


Wichtig sind hier auch hauptsächlich der day_of_week und der last_month_price.

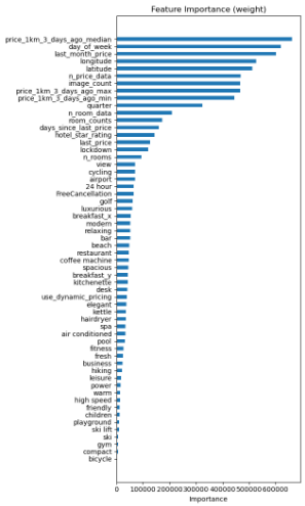


Mit Description und Competitor Last Price

Wenn zusätzlich zur Description auch noch der Competitor Price von 1km Distanz hinzugenommen wird, dann verbessert sich die Performance des R2 erneut minimal auf: 0.893.



Diese Verbesserung wird sehr wahrscheinlich durch die Preise der Konkurrenz ermöglicht:

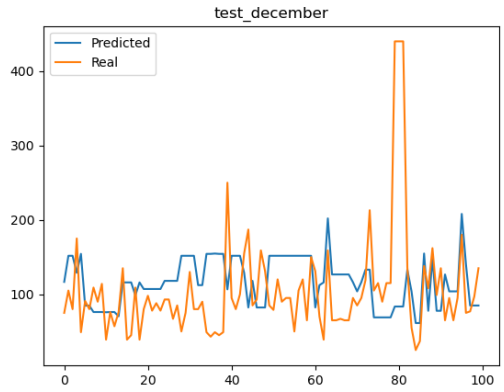


Folgende Features werden nun verwendet: ['price_1km_3_days_ago_median', 'price_1km_3_days_ago_min', 'price_1km_3_days_ago_max', 'luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast_x', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour', 'hotel_star_rating', 'image_count', 'latitude', 'longitude', 'use_dynamic_pricing', 'room_counts', 'n_rooms', 'n_price_data', 'n_room_data', 'breakfast_y', 'FreeCancellation', 'HalfBoard', 'FullBoard', 'AllInclusive', 'lockdown', 'last_price', 'days_since_last_price', 'quarter', 'day_of_week', 'last_month_price']

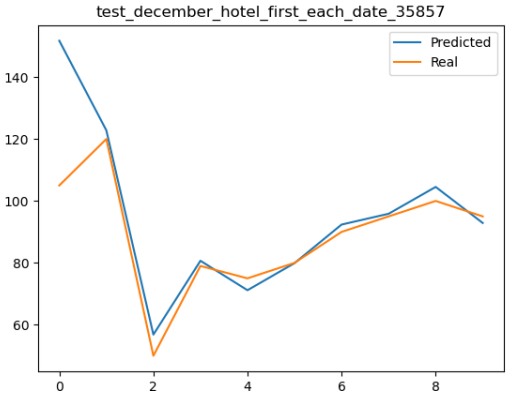
Folglich wurde dieses letzte Modell so als akzeptiert angesehen. Wenn nun die Performance auf den drei Test-Datensatzs angesehen wird, sieht man folgende Resultate:

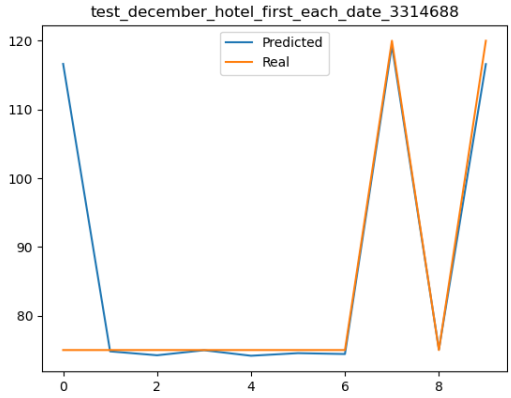
Datenset	R2	Mean Absolute Error	Mean Squared Error
Dezember	0.718	32.13	23759.9
Bristol	0.811	18.04	4094.3
London	0.62	56.31	67429.6

Wenn man die ersten 100 Predictions des Dezember Datensets betrachtet, erhält man folgende Resultate:



Wenn man dasselbe für zwei Hotels aus dem Dezember Datenset macht und für die 10 verfügbaren Tage jeweils den ersten Preis vorhersagt, erhält man folgendes Resultat.

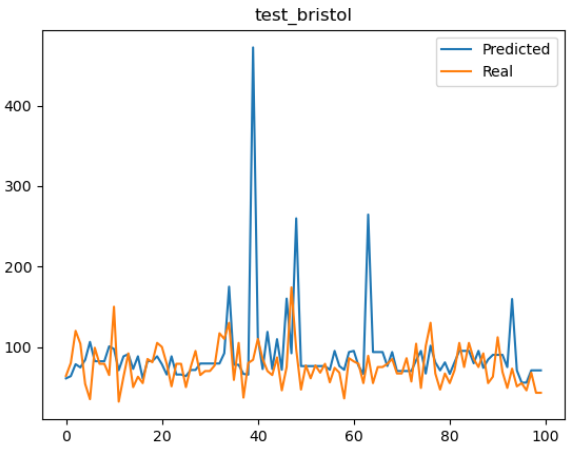




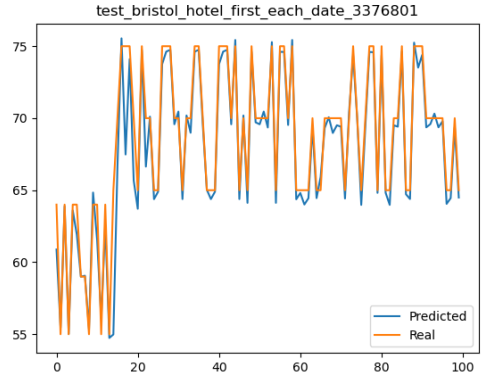
Datenset Dezember	R2	Mean Absolute Error	Mean Squared Error
Hotel 35857 / erster Preis pro Tag	0.309	7.184	228.8
Hotel 3314688 / erster Preis pro Tag	0.461	4.866	174.7

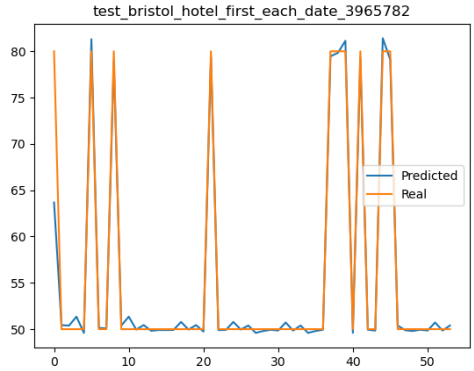
Die Performance ist dabei optisch sehr gut. Der tiefe R2 Wert ist sehr wahrscheinlich durch die Abweichung am Tag 0 zu erklären und der Einfluss dieser Abweichung ist bei N=10 Datenpunkten sehr stark messbar.

Wenn man die ersten 100 Predictions des Bristol Datensets verwendet, dann gibt es folgende Resultate.



Wenn man dasselbe für zwei Hotels aus dem Bristol Datenset macht und für die verfügbaren Tage jeweils den ersten Preis vorhersagt, erhält man folgendes Resultat.

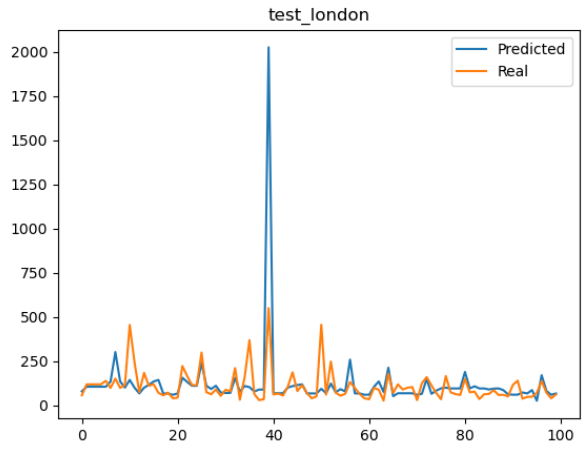




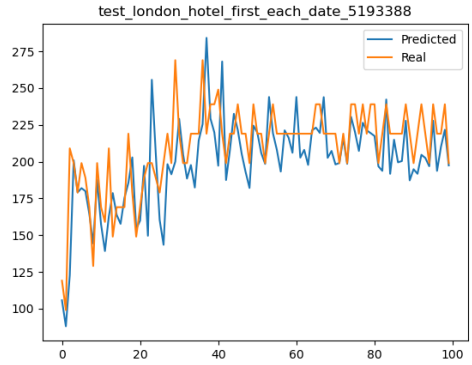
Datenset Bristol	**R2**	**Mean Absolute Error**	**Mean Squared Error**
Hotel 3376801 / erster Preis pro Tag	0.91	0.823	2.402
Hotel 3965782 / erster Preis pro Tag	0.961	18.04	4094.3

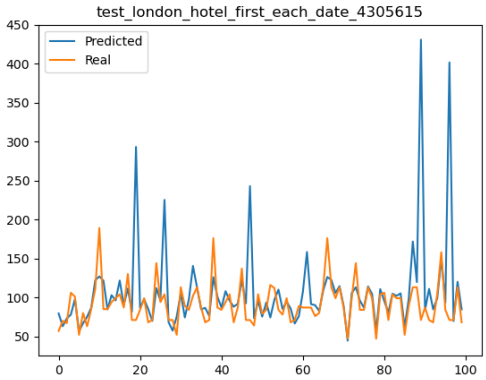
Hier ist die Performance auf beiden Hotels sehr gut und es scheint eine starke Vorhersagekraft zu geben.

Wenn man die ersten 100 Predictions des London Datensets verwendet, dann gibt es folgende Resultate.



Um diese Performance zu überprüfen, wurden ebenfalls zufällig zwei Hotels gewählt und jeweils pro Tag der erste verfügbare Preis vorhergesagt.





Datenset Dezember	**R2**	**Mean Absolute Error**	**Mean Squared Error**
Hotel 5193388 / erster Preis pro Tag	0.267	18.04	3451
Hotel 4305615 / erster Preis pro Tag	-4.312	18.04	579.8

Hier ist sehr klar zu sehen, dass die Performance auf dem Hotel 5193388 signifikant besser ist, als die Performance auf dem Hotel 4305615. Folglich ist das Modell für London nur sehr bedingt einsetzbar.

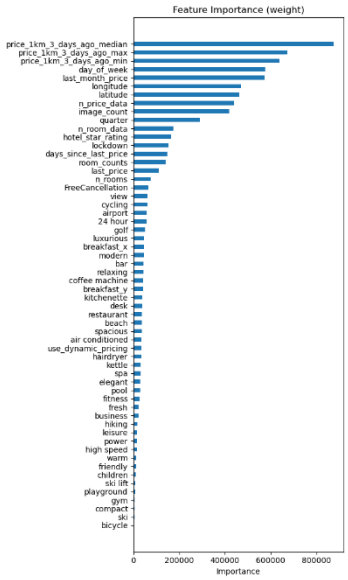
Dies zeigt eine gute Generalisation auf einen anderen Ort (Bristol) und eine relativ gute Generalisierung auf einen anderen Zeitraum (Dezember). Innerhalb des Ortes Londons, sind die Resultate weniger vielversprechend. Dies ist aber nicht so ein grosses Problem, da man hauptsächlich Hotels auf Bristol betrachtet.

Definitives Modell

Datum: 11.04.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_model/competitor_model_1_fit.py

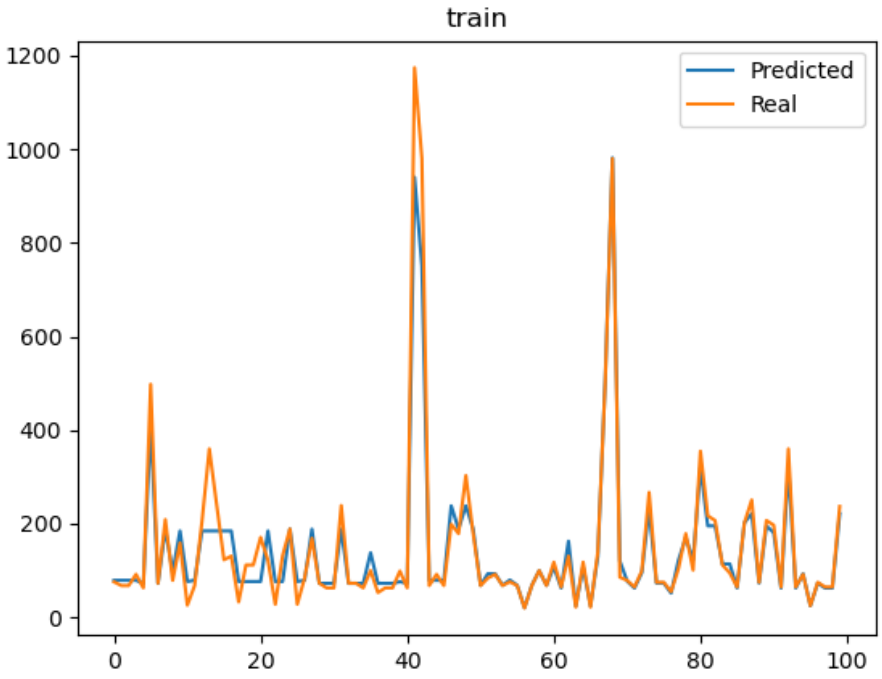
Beschreibung

In einem weiteren Schritt wurde das Modell auf den ganzen Daten trainiert. Dies ergab folgende Feature Wichtigkeiten:



Ebenfalls kann mit dem Vorhersagen auf den Trainingsdaten gezeigt werden, dass es ein gutes Fitting auf die Trainingsdaten gibt. Es wird nämlich ein R2 von 0.964 erreicht.

Ebenfalls kann man sehen, wenn man sich die ersten 100 Zeilen des Trainingssets vorhersagt, dass es gute Vorhersagen macht:



Validierung der Plausibilität

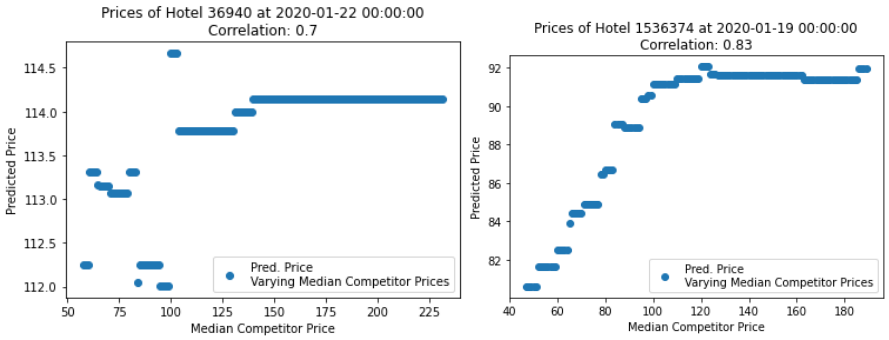
Datum: 12.04.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: competitor_model/validate_competitor_model_1.ipynb

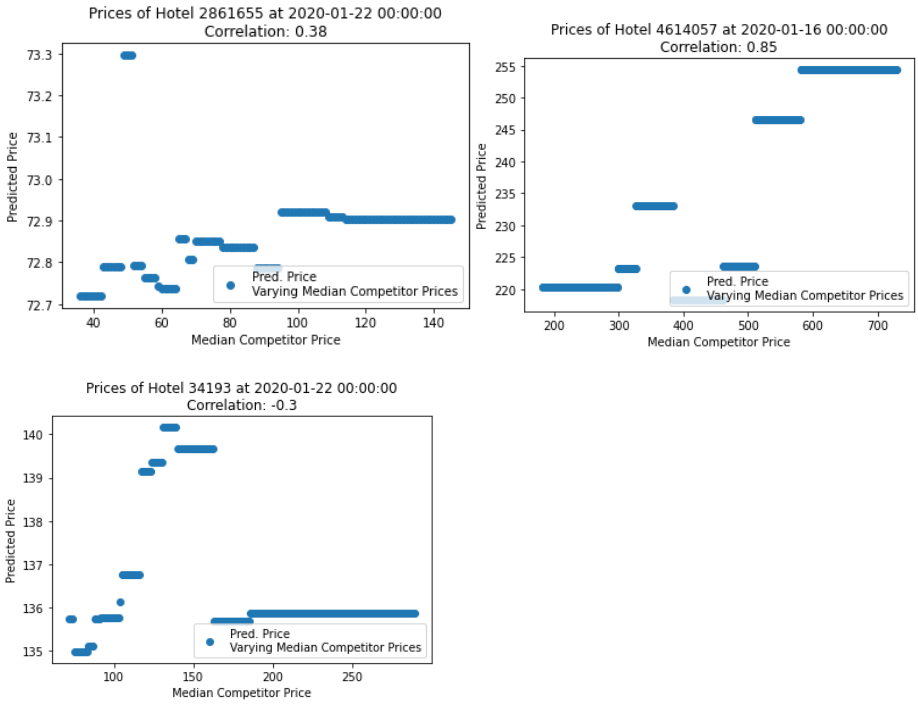
Um auf einfache manuelle Art das Modell zu validieren, wurden für 5 zufällige Datenpunkte alles bis auf den Preis der Konkurrenz ('price_1km_3_days_ago_median') gefixt und dann Vorhersagen gemacht. Hier gibt es mehrere Möglichkeiten wie man sich verhalten kann, entweder erhöht man mit der Konkurrenz der Preis oder senkt den Preis, wenn die Konkurrenz erhöht, um so konkurrenzfähig zu sein.

Der Preis wurde dabei jeweils zwischen 0.5 dem effektiven durchschnittlichen Preis und 2-mal den durchschnittlichen Preis variiert.

Resultate

Hier ist es sehr spannend zu sehen, dass sich jedes Hotel bei ändernden Konkurrenzpreisen anders verhaltet.





Wie erwartet, erhöhen sehr viele Hotels die Preise, wenn die Konkurrenz die Preise erhöht. Siehe Erste 4 Bilder. Jedoch gibt es auch Hotels, die initial den Preis erhöhen und dann bei zu hohen Konkurrenzpreisen den eigenen Preis wieder senken.

Fazit

Dieses unterschiedliche Verhalten macht Sinn und macht das Setup spanend. In einem weiteren Schritt könnte beispielsweise der Einfluss des Preises des vorherigen Tages oder des min oder max Preises der Konkurrenz berechnet werden. Ebenfalls ist es spannend zu sehen, dass der Preis eher tiefer als die Konkurrenz ist.

Kundenmodelle

Im Rahmen der Nachfrage / Kundenmodells wird die Kundennachfrage an einem bestimmten Zeitpunkt modelliert.

Feature-basiertes XGBoost Modell

XGBoost

Eine Möglichkeit ist es dabei ebenfalls ein XGBoost Modell zu verwenden und dabei anhand des Reservation Datensets eine gewisse Vorhersage zu treffen.

Datum: 30.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_1.ipynb

Gruppieren der Daten

Als Nachfrage pro Tag und Hotel werden die Reservationen nach date_of_stay und hotel_id gruppiert, gezählt und mit der jeweiligen Anzahl Personen in diesem Zimmer multipliziert.

Features

Description

Anhand der Hotel Beschreibung wurde anschliessend erneut die Keywords erzeugt. Damit kann dann eine Aussage über ein Hotel gemacht werden.

Zeit

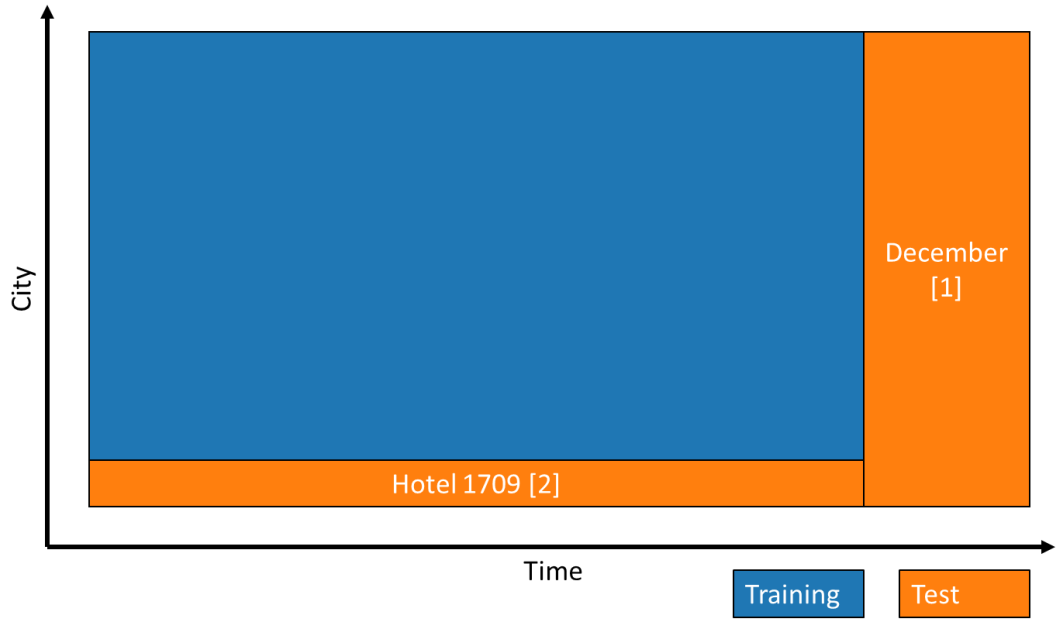
Um die Zeit abzubilden und Vorhersagen für einen bestimmten Tag zu machen, wurden folgende Features erzeugt:

«month», «quarter», «day_of_week»

Splitting der Daten

Datum: 31.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/splitting.ipynb

Im Fall des Demand Modells entschied ich mich erneut für zwei Splittings. So habe ich ein Hotel rausgenommen und als zusätzliches Test-Datensatz nur den Dezember.



Die Fläche der Balken ist nicht repräsentativ und dient einzig der Visualisierung.

ID	Daten	Beschreibung	Ziel
0	Trainingsdaten	Trainingsdaten, bei welchen in einem Time Series Cross Validation Split mit Grösse 11 die Daten vor Dezember 2020 trainiert werden. Ohne die Testdaten unten!	Rollende Performance beurteilen.
1	Dezember	Testdaten > Dezember	Performance in der Zukunft beurteilen.
2	Hotel	Testdaten vor Dezember 2020 für das Hotelzimmer 1709	Performance auf anderen Zimmertyp beurteilen

Es wurde bewusst keine Überschneidung der Daten erlaubt. Dadurch kann eine bessere Aussage gemacht werden.

Hyperparameter Tuning

Datum: 31.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/run_hyperparameters.py

In diesem Fall ist das Hyperparameter Tuning signifikant einfacher. Es wurde auf den folgenden Features trainiert auf den vor Dezember Daten trainiert.

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern',

'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym',

'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach',

'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business',

'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground',

'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift',

'coffee machine', '24 hour', 'latitude', 'longitude', 'quarter',

'month', 'year', 'day_of_week']

Es wurden dabei ein RandomSearch gemacht auf den folgenden Parametern.

```
hyperparameter_grid = {
  'n_estimators': [50, 100, 200, 500],
  'max_depth': [3, 5, 10, 15, 30],
  'learning_rate': [0.05, 0.1, 0.15, 0.20],
  'min_child_weight': [1, 2, 3, 4]
  }
```

Dies erzeugte das beste Resultat:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
    colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
    gamma=0, gpu_id=-1, importance_type=None,
    interaction_constraints='', learning_rate=0.05, max_delta_step=0,
    max_depth=30, min_child_weight=4, missing=nan,
    monotone_constraints='()', n_estimators=100, n_jobs=8,
    num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
    reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
    validate_parameters=1, verbosity=None)
```

Cross-Validation Modell

Datum: 31.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_1.ipynb

Aufbau

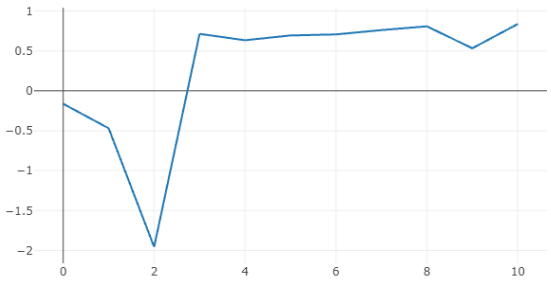
Das Train Datenset wurde mit den zuvor gesetzten Hyperparametern trainiert.

Folgende Features wurden verwendet:

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour', 'latitude', 'longitude', 'quarter', 'month', 'year', 'day_of_week']

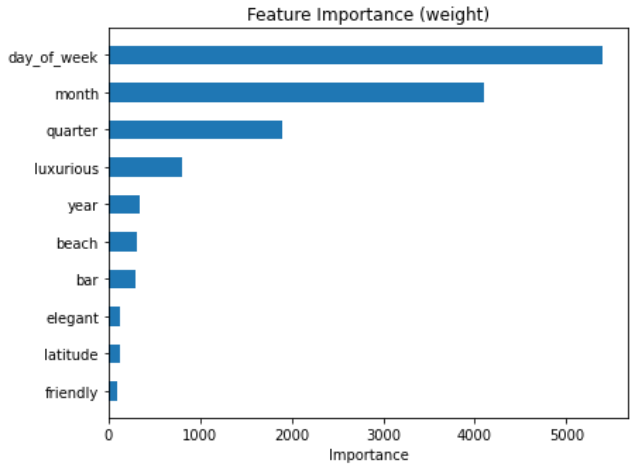
Resultate

Dabei wurden folgende R2 auf dem Cross Validation Set erreicht:



Es ist sehr klar sichtbar, dass nach einer gewissen Zeit die Nachfrage sehr gut vorhergesehen werden kann und am Ende ein R2 von 0.836 erreicht wird. Auch der MAE konvergiert immer mehr gegen 0, was ein gutes Zeichen ist.

Die Feature Wichtigkeit zeigt sehr stark, dass die Nachfrage abhängig vom day of week, month und quarter ist.

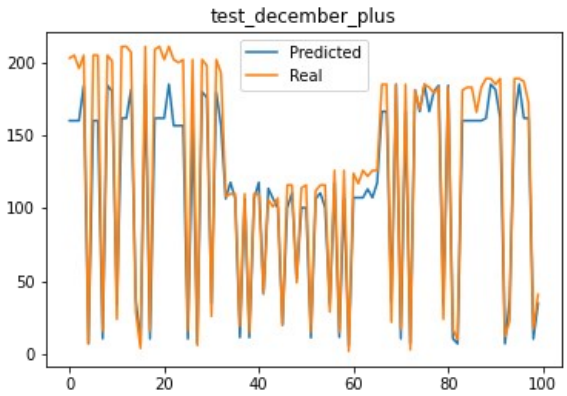


Folglich wurde dieses Modell so als akzeptiert angesehen. Wenn nun die Performance auf den beiden Test-Datensatzs angesehen wird, sieht man folgende Resultate:

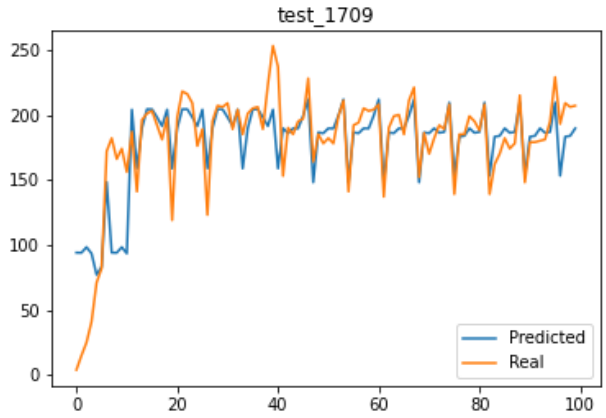
Datenset	**R2**	**Mean Absolute Error**	**Mean Squared Error**
Dezember	0.837	18.54	643.6
1709	0.692	14.13	549.1

Dies zeigt sehr gut, dass das Modell auch ziemlich gut auf andere Zeiträume und auch gut, aber weniger gut auf andere Hotels generalisiert.

Wenn man die ersten 100 Predictions des Dezember Datensets betrachtet, erhält man folgende Resultate:



Wenn man die ersten 100 Predictions des 1709 Datensets verwendet, dann gibt es folgende Resultate.

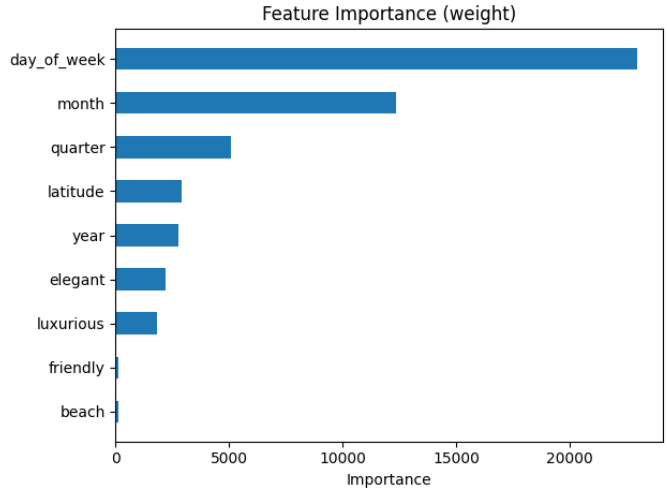


Definitives Modell

Datum: 31.03.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_1_fit.py

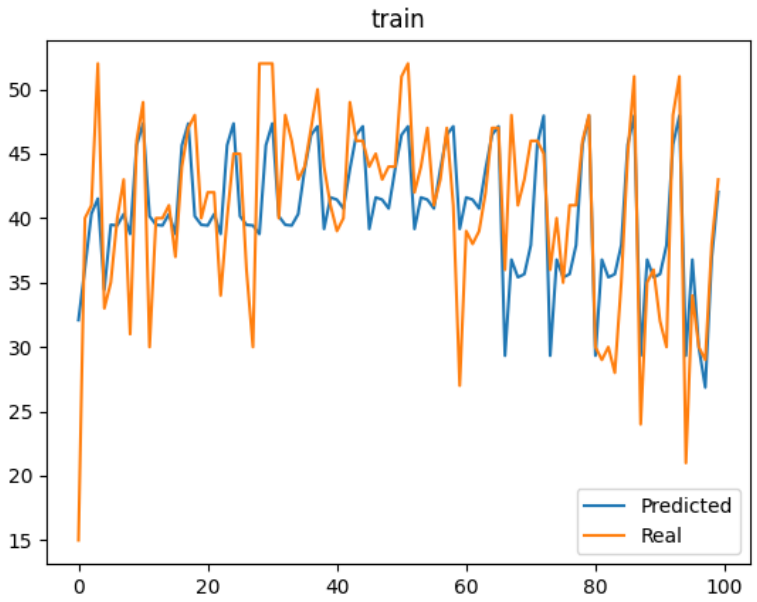
Beschreibung

In einem weiteren Schritt wurde das Modell auf dem ganzen Daten gefittet. Dies ergab folgende Feature Wichtigkeiten:



Ebenfalls kann mit dem Vorhersagen der Trainingsdaten gezeigt werden, dass es ein gutes Fitting auf die Trainingsdaten gibt, aber kein Overfitting. Es wird nämlich ein R2 von 0.892 erreicht.

Ebenfalls kann man sehen, wenn man sich die ersten 100 Zeilen des Trainingssets vorhersagt, dass es gute Vorhersagen macht:



Feature-basiertes XGBoost Modell mit Einbezug des Median-Preises

Aufgrund des Feedbacks von Daniel Pfäffli vom 06.04.2022 macht es Sinn, dass das Modell erneut trainiert wird und der Median Preis dieses Hotels für einen Tag noch einbezogen wird.

Hyperparameter Tuning

Datum: 06.04.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/run_hyperparameters.py

In diesem Fall ist das Hyperparameter Tuning signifikant einfacher. Es wurde auf den folgenden Features trainiert auf den vor Dezember Daten trainiert.

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern',

'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym',

'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach',

'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business',

'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground',

'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift',

'coffee machine', '24 hour', 'latitude', 'longitude', 'quarter',

'month', 'year', 'day_of_week', 'median_price']

Es wurden dabei ein RandomSearch gemacht auf den folgenden Parametern.

```
hyperparameter_grid = {
  'n_estimators': [50, 100, 200, 500],
  'max_depth': [3, 5, 10, 15, 30],
  'learning_rate': [0.05, 0.1, 0.15, 0.20],
  'min_child_weight': [1, 2, 3, 4]
  }
```

Dies erzeugte das beste Resultat:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
       gamma=0, gpu_id=-1, Wichtigkeit_type=None,
       interaction_constraints='', learning_rate=0.15, max_delta_step=0,
       max_depth=5, min_child_weight=1, missing=nan,
       monotone_constraints='()', n_estimators=50, n_jobs=8,
       num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
       validate_parameters=1, verbosity=None)
```

Cross-Validation Modell

Datum: 06.04.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_2.ipynb

Aufbau

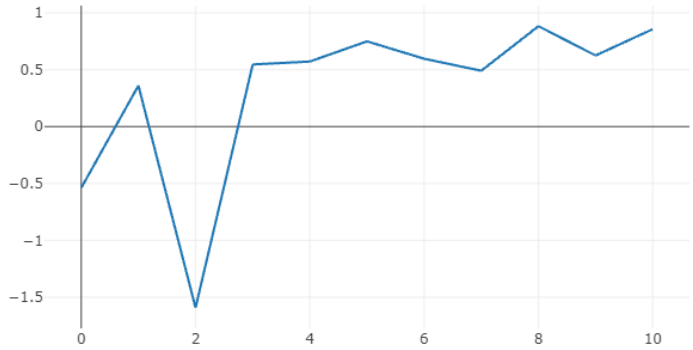
Das Train Datenset wurde mit den zuvor gesetzten Hyperparametern trainiert.

Folgende Features wurden verwendet:

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour', 'latitude', 'longitude', 'quarter', 'month', 'year', 'day_of_week', 'median_price']

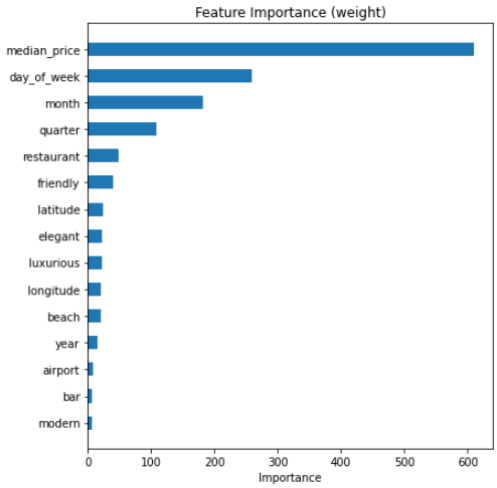
Resultate

Dabei wurden folgende R2 auf dem Cross Validation Set erreicht:



Es ist sehr klar sichtbar, dass nach einer gewissen Zeit die Nachfrage sehr gut vorhergesehen werden kann und am Ende ein R2 von 0.856 erreicht wird. Auch der MAE konvergiert immer mehr gegen 0, was ein gutes Zeichen ist.

Die Feature Wichtigkeit zeigt sehr stark, dass die Nachfrage abhängig vom median Preis, day of week, month und quarter ist.

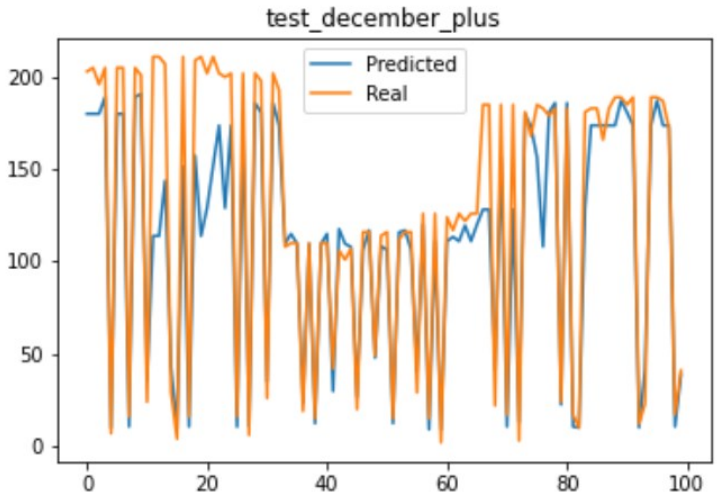


Folglich wurde dieses Modell so als akzeptiert angesehen. Wenn nun die Performance auf den beiden Test-Datensatzs angesehen wird, sieht man folgende Resultate:

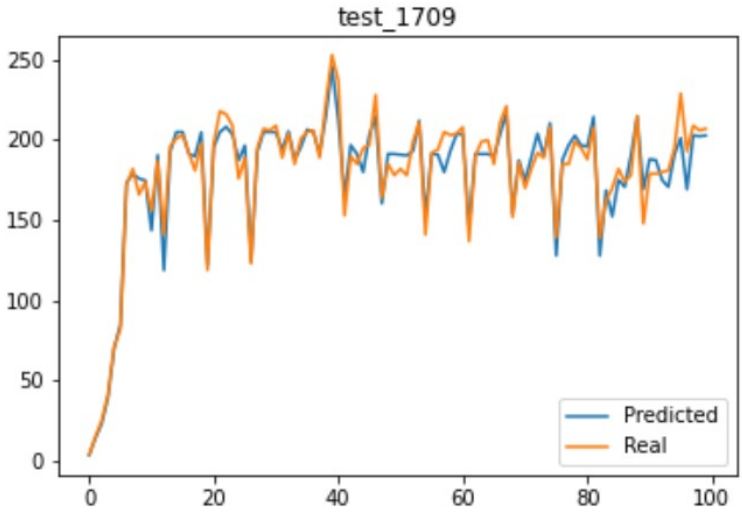
Datenset	**R2**	**Mean Absolute Error**	**Mean Squared Error**
Dezember	0.756	21.8	962.8
1709	0.953	6.998	84

Hier sieht man im Gegensatz zum Versuch 1, dass das Modell viel besser auf andere Hotels generalisiert und schlechter auf andere Zeiträume.

Wenn man die ersten 100 Predictions des Dezember Datensets betrachtet, erhält man folgende Resultate:



Wenn man die ersten 100 Predictions des 1709 Datensets verwendet, dann gibt es folgende Resultate.

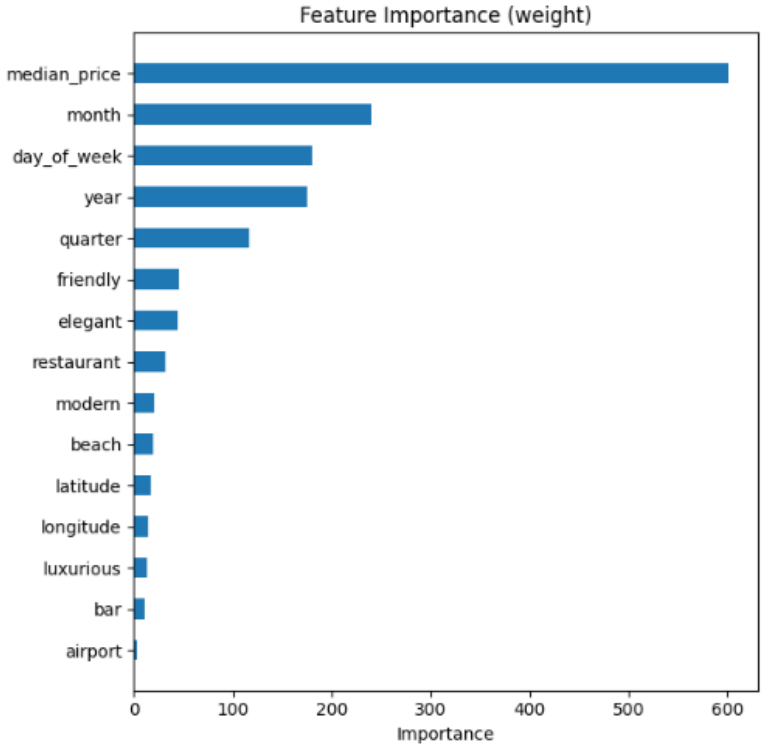

test_1709
250
200
150
100
50
0
0
20
40
60
80
100
Predicted
Real

Definitives Modell

Datum: 06.04.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_2_fit.py

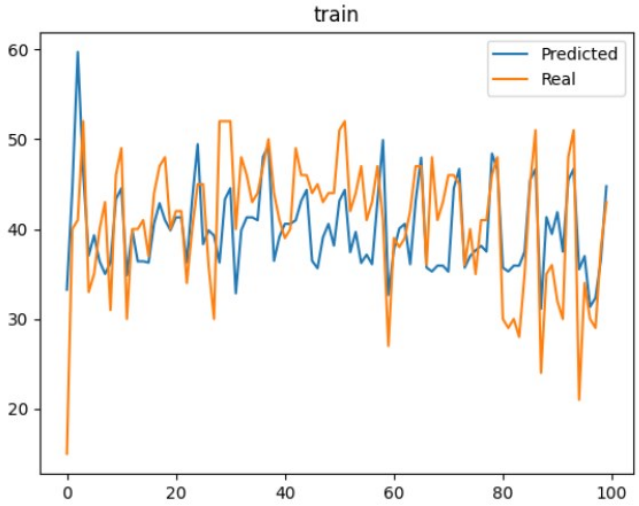
Beschreibung

In einem weiteren Schritt wurde das Modell auf dem ganzen Daten gefittet. Dies ergab folgende Feature Wichtigkeiten:



Ebenfalls kann mit dem Vorhersagen der Trainingsdaten gezeigt werden, dass es ein gutes Fitting auf die Trainingsdaten gibt. Es wird nämlich ein R2 von 0.933 erreicht.

Ebenfalls kann man sehen, wenn man sich die ersten 100 Zeilen des Trainingssets vorhersagt, dass es gute Vorhersagen macht:



Validierung der Plausibilität

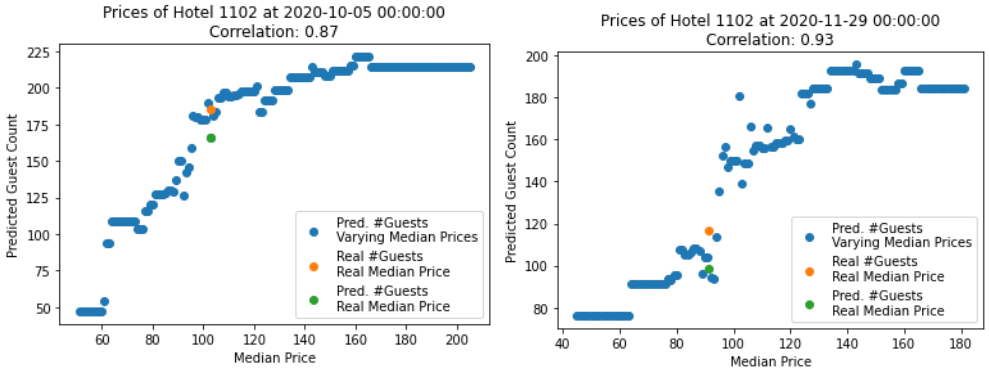
Datum: 06.04.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/validate_demand_model_2.ipynb

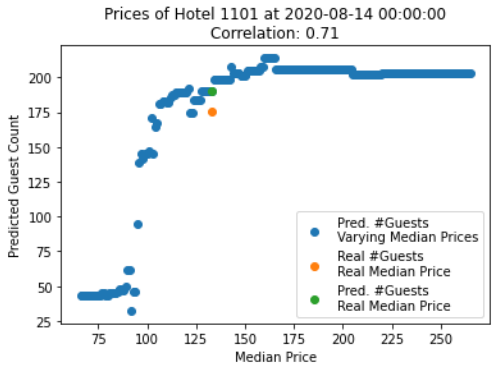
Um auf einfache manuelle Art das Modell zu validieren, wurden für 5 zufällige Datenpunkte alles bis auf den Preis gefixt und dann Vorhersagen gemacht. Denn rein theoretisch würde es Sinn machen, dass die Nachfrage sinkt, wenn der Preis steigt.

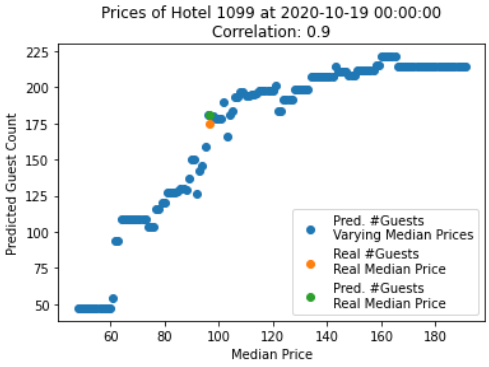
Der Preis wurde dabei jeweils zwischen 0.5 dem effektiven Median Preis und 2-mal den Medianpreis variiert.

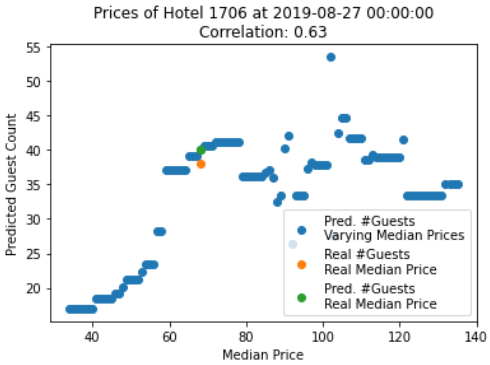
Resultate

Diese These kann insofern nicht validiert werden, da bei allen Versuchen eine positive Korrelation zwischen dem Medianpreis und der Anzahl Gäste erzielt wird. Das heisst, je höher der Preis ist, desto höher sind die Anzahl Gäste.









Fazit

Dieser Effekt gibt es sehr wahrscheinlich daraus, dass im Reservation Datenset die Preise jeweils anhand der Nachfrage definiert wurden und nicht umgekehrt. Das heisst wenn es eine grosse Nachfrage gab, dann wurden die Preise erhöht, da die Leute sowieso einen Raum buchen. Deshalb ist die Berechnung der Nachfrage anhand des Preises mit diesem Datenset nur sehr bedingt möglich.

Weiteres Vorgehen

In einem weiteren Schritt wird zuerst einmal das Modell aus dem ersten Versuch verwendet. Damit wird die allgemeine Nachfrage für ein bestimmtes Hotel simuliert, unabhängig vom aktuellen Preis. In einem weiteren Schritt kann dann eine Anpassung der Nachfrage aufgrund der Preiselastizität und des Preises erreicht werden.

Dies erfordert aber einen grossen Anteil an Expertenwissen, was im aktuellen Moment nicht vorhanden ist.

Regressionsbasiertes Kundenmodell mit Kundengruppen

Datum: 28.04.2022

Ziel: Um eine kleinere Abhängigkeit vom Kundemodell zu haben wird das Kundenmodell verbessert

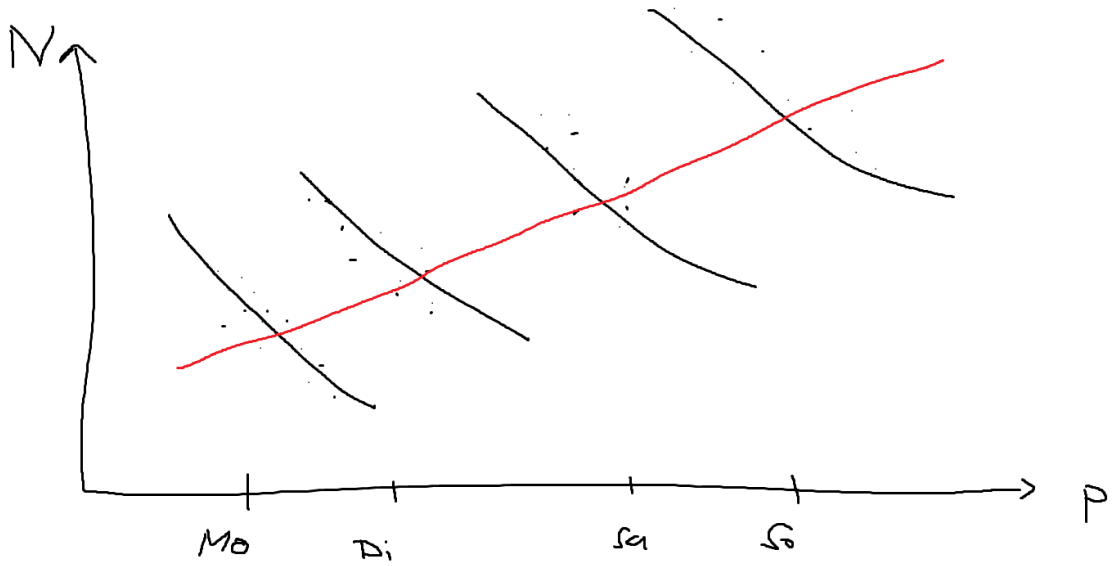
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/blob/main/demand_model/price_sensitivity_demand_model.ipynb

Beschreibung

Beschreibung Tobias Baltensperger – 28.04.2022

mein Vorschlag wäre, dass du versuchst von den bestehenden Daten die Parametrisierung deines Kundenmodells abzuleiten, und dann deine Ergebnisse anhand gängiger Werte für diese Parameter validierst.

Für die Parametrisierung brauchst du in erster Linie die Preissensitivität. Deine Daten werden dir sagen, dass hohe Preise und hohe Nachfrage positiv korrelieren, was wenig hilfreich ist, um Preissensitivität zu ermitteln. Wir wissen ja aus Economics 101, dass höhere Preise generell die Nachfrage dämpfen. D.h. in einem ersten Schritt musst du die Daten so gruppieren, dass du eine sinnvolle Aussage daraus treffen kannst, z.B. in dem du sie nach Wochentag gruppierst, plus ev Saisonalität, oder zusätzlichen Kriterien. Hier eine Illustration:



N: Nachfrage; P: Preis; Mo-So: Wochentage.

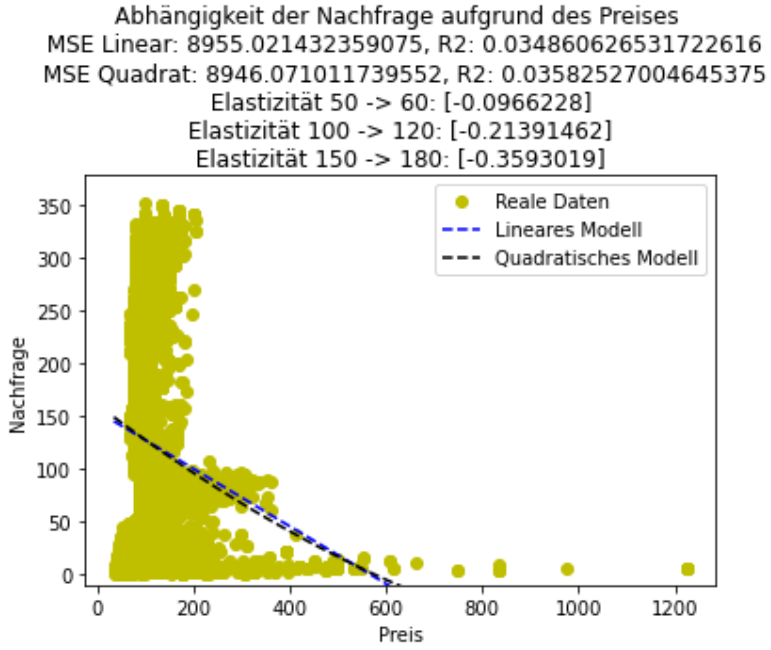
Wenn du alle Daten zusammen anschaust kommt die rote Linie raus. Die Frage ist ob du durch geschicktes Gruppieren der Daten die schwarzen Kurven rauskriegen kannst. Klar, die implizite Annahme die du bei einer solchen Gruppierung triffst ist, dass die Nachfragekurve an Montagen immer gleich ist und sich die Nachfrage nur in Abhängigkeit des Preises ändert. Aber diese Annahme ist sicher sinnvoller als die Annahme, dass die Nachfragekurve über alle Tage hinweg gleich ist (das ist ja die implizite Annahme bei der rote Linie...).

Daraus kannst du dann (hoffentlich) eine Preiselastizität ableiten, diese kannst du mit online Quellen vergleichen um zu sehen ob du nicht komplett daneben liegst.

In einem 2. Schritt kannst du dein Kundenmodell verfeinern. Sagen wir du stellst fest, dass sich die Preissensitivitäten gemäss einem bestimmten Muster stark unterscheiden, zb viel tiefer unter der Woche als am Weekend. In diesem Beispiel könntest du dann 2 Gruppen machen: der unter-der-Woche Besucher, und der Weekend-Besucher. Oder du leitest aus deinem Datenset approximativ ab, wieviele deiner Kunden Businesskunden sind, und wie viele Touristen, in Abhängigkeit des Wochentages. Dann kannst du diesen 2 Gruppen je eine Preissensitivität zuordnen, und eine Zusammensetzung dieser Gruppen in Abhängigkeit des Wochentages bestimmen... hier sind deine Möglichkeiten recht offen.

Vorgehen

Wenn man alle Reservationen betrachtet, dann sieht man, dass es auch hier eine gewisse negative Verbindung gibt:



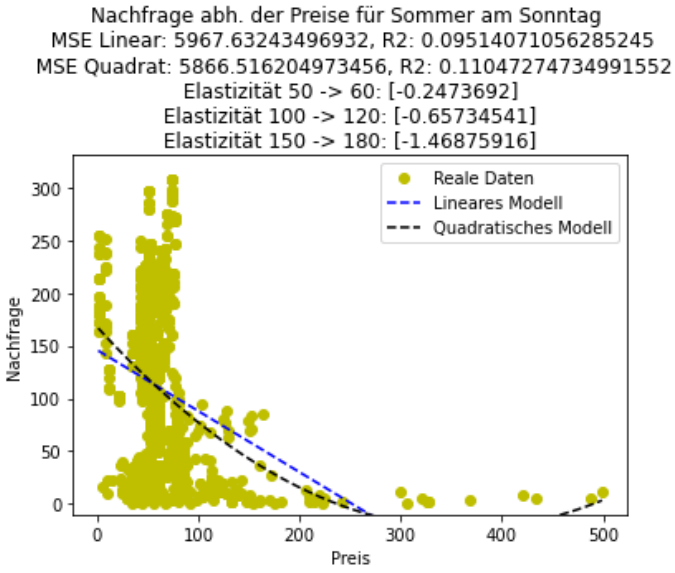
Jedoch ist dies sehr generell und eine Betrachtung nach Tag und Saison macht sicher noch mehr Sinn.

Dazu wurden alle minimalen Preise unter 500 Franken betrachtet und die Anzahl Buchungen am jeweiligen Tag.

Dabei wurden die Buchungen nach Tag und Sommer (Monate Mai, Juni, Juli, August) oder nicht aufgeteilt, um somit 14 verschiedene Preissensivitäten zu berechnen.

Dabei wurde ein Lineares und ein Quadratisches Modell gefittet:

Für die Preise im Sommer für den Sonntag sieht das folgendermassen aus:



Resultate

Dabei wurden folgende Resultate erzielt:

Modell	**Mean MSE (Std)**	**Mean R2 (Std)**
Linear	7593.60 (1178.92)	0.116 (0.026)
Quadratisch	7460.64 (1158.705)	0.131 (0.033)

Dies zeigt, dass das lineare Modell leicht schlechter die Preise vorhersagt. Jedoch ist es deutlich einfacher zu erklären. Dadurch wurde es sich entschieden, dass man das lineare Modell nimmt.

Wenn man die Lineare Preissensitivitäten Sommer, *Nicht Sommer* der Nachfrage betrachtet, erhält man folgende Werte:

	Montag	**Dienstag**	**Mittwoch**	**Donnerstag**	**Freitag**	**Samstag**	**Sonntag**
50 -> 60	-0.319	-0.317	-0.34	-0.284	-0.203	-0.192	-0.247
CHF	*-0.263*	*-0.277*	*-0.308*	*-0.264*	*-0.170*	*-0.160*	*-0.192*
100 -> 120	-0.938	-0.926	-1.041	-0.794	-0.508	-0.476	-0.657
CHF	*-0.714*	*-0.768*	*-0.892*	*-0.717*	*-0.411*	*-0.380*	*-0.475*
150 -> 180	-2.647	-2.588	-3.256	-1.977	-1.023	-0.937	-1.469
CHF	*-1.666*	*-1.869*	*-2.416*	*-1.675*	*-0.775*	*-0.703*	*-0.935*

Dabei ist sehr spannend zu beachten, dass die Nachfrage vor allem im Sommer elastischer ist. Dies ist sehr wahrscheinlich daran geschuldet, dass im Sommer die Leute eher einmal nicht buchen, wenn es zu teuer ist. Jedoch im Winter reisen eher Leute an, welche sowieso kommen werden.

Ebenfalls sind die Preise gegen Ende der Woche weniger elastisch, dies bedeutet, dass am Wochenende sehr wahrscheinlich die Touristen sowieso kommen wollen und unabhängig vom Preis das Zimmer buchen werden.

Grundsätzlich stimmen diese Elastizitäten ungefähr mit den empirischen Werten von Corgel et. al überein, wo Elastizitäten zwischen -0.08 und -1.36 liegen. Ebenfalls bestätigt sich in den vorliegenden Daten, dass es bei höherpreisigen Hotels eine höhere Elastizität gibt, was auch unserem Modell entspricht. (Corgel et al., 2012)

Fazit

Dadurch konnte ein relativ einfaches Modell erzielt werden, welches Preisensivitäten ergibt, welche durchaus vernünftig sind.

Feature-basiertes XGBoost Modells mit Einbezug der Konkurrenzdaten (Reservationsdatensatz)
Aufgrund des Feedbacks von Tobias Baltensperger von 10.05.2022 macht es Sinn, dass das Modell erneut trainiert wird und zusätzlich drei Feature Paare für die Konkurrenz miteinbezogen wird. Als Konkurrenz wurde dabei jeweils alle anderen Hotels (nicht nur verschiedene Zimmer) aus dem Reservationsdatenset verwendet. Beim Test-Datensatz mit einem Hotel wurde das eigene Hotel als Test-Datensatz verwendet.

Ebenfalls wurden die Features nicht mehr als One-Hot Vektoren betrachtet!

Cross-Validation Modell
Datum: 11.05.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_3.ipynb, https://gitlab.enterpriselab.ch/baa-dynamic-pricing/mlflow-results/-/commit/2461a99ad06f8a819cc8f77c59a698b97341e173

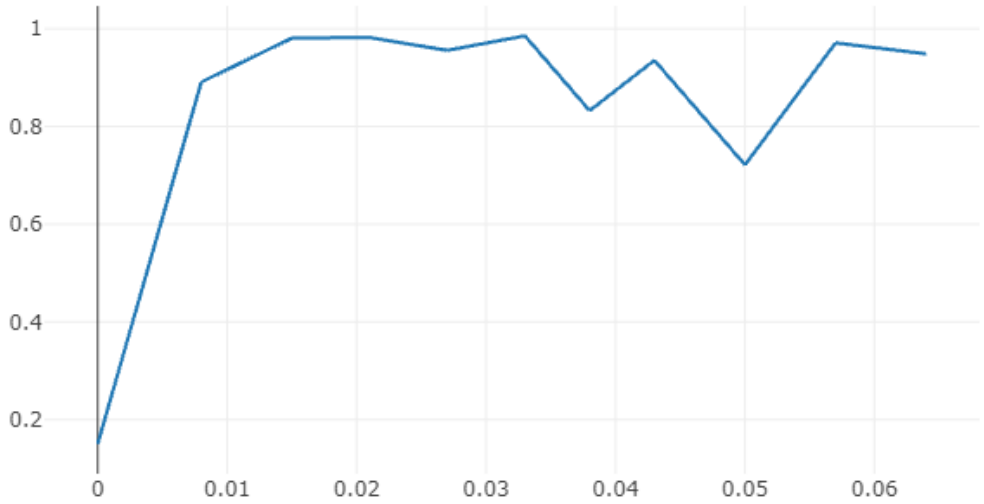
Aufbau
Folgende Features wurden verwendet:

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour', 'latitude', 'longitude', 'median_price', 'min_price', 'max_price', 'luxurious_competitor_1', 'elegant_competitor_1', 'relaxing_competitor_1', 'spacious_competitor_1', 'modern_competitor_1', 'restaurant_competitor_1', 'spa_competitor_1', 'pool_competitor_1', 'bar_competitor_1', 'fitness_competitor_1', 'leisure_competitor_1', 'gym_competitor_1', 'power_competitor_1', 'breakfast_competitor_1', 'compact_competitor_1', 'airport_competitor_1', 'ski_competitor_1', 'skiing_competitor_1', 'beach_competitor_1', 'view_competitor_1', 'desk_competitor_1', 'kitchenette_competitor_1', 'kettle_competitor_1', 'friendly_competitor_1', 'warm_competitor_1', 'business_competitor_1', 'hiking_competitor_1', 'cycling_competitor_1', 'bicycle_competitor_1', 'golf_competitor_1', 'fresh_competitor_1', 'playground_competitor_1', 'children_competitor_1', 'hairdryer_competitor_1', 'air conditioned_competitor_1', 'high speed_competitor_1', 'ski lift_competitor_1', 'coffee machine_competitor_1', '24 hour_competitor_1', 'latitude_competitor_1', 'longitude_competitor_1', 'median_price_competitor_1', 'min_price_competitor_1', 'max_price_competitor_1', 'luxurious_competitor_2', 'elegant_competitor_2', 'relaxing_competitor_2', 'spacious_competitor_2', 'modern_competitor_2', 'restaurant_competitor_2', 'spa_competitor_2', 'pool_competitor_2', 'bar_competitor_2', 'fitness_competitor_2', 'leisure_competitor_2', 'gym_competitor_2', 'power_competitor_2', 'breakfast_competitor_2', 'compact_competitor_2', 'airport_competitor_2', 'ski_competitor_2', 'skiing_competitor_2', 'beach_competitor_2', 'view_competitor_2', 'desk_competitor_2', 'kitchenette_competitor_2', 'kettle_competitor_2', 'friendly_competitor_2', 'warm_competitor_2', 'business_competitor_2', 'hiking_competitor_2', 'cycling_competitor_2', 'bicycle_competitor_2', 'golf_competitor_2', 'fresh_competitor_2', 'playground_competitor_2', 'children_competitor_2', 'hairdryer_competitor_2', 'air conditioned_competitor_2', 'high speed_competitor_2', 'ski lift_competitor_2', 'coffee machine_competitor_2', '24 hour_competitor_2', 'latitude_competitor_2', 'longitude_competitor_2', 'median_price_competitor_2', 'min_price_competitor_2', 'max_price_competitor_2', 'luxurious_competitor_3', 'elegant_competitor_3', 'relaxing_competitor_3', 'spacious_competitor_3', 'modern_competitor_3', 'restaurant_competitor_3', 'spa_competitor_3', 'pool_competitor_3', 'bar_competitor_3', 'fitness_competitor_3', 'leisure_competitor_3', 'gym_competitor_3', 'power_competitor_3',

'breakfast_competitor_3', 'compact_competitor_3', 'airport_competitor_3', 'ski_competitor_3', 'skiing_competitor_3', 'beach_competitor_3', 'view_competitor_3', 'desk_competitor_3', 'kitchenette_competitor_3', 'kettle_competitor_3', 'friendly_competitor_3', 'warm_competitor_3', 'business_competitor_3', 'hiking_competitor_3', 'cycling_competitor_3', 'bicycle_competitor_3', 'golf_competitor_3', 'fresh_competitor_3', 'playground_competitor_3', 'children_competitor_3', 'hairdryer_competitor_3', 'air conditioned_competitor_3', 'high speed_competitor_3', 'ski lift_competitor_3', 'coffee machine_competitor_3', '24 hour_competitor_3', 'latitude_competitor_3', 'longitude_competitor_3', 'median_price_competitor_3', 'min_price_competitor_3', 'max_price_competitor_3', 'quarter', 'month', 'year', 'day_of_week']

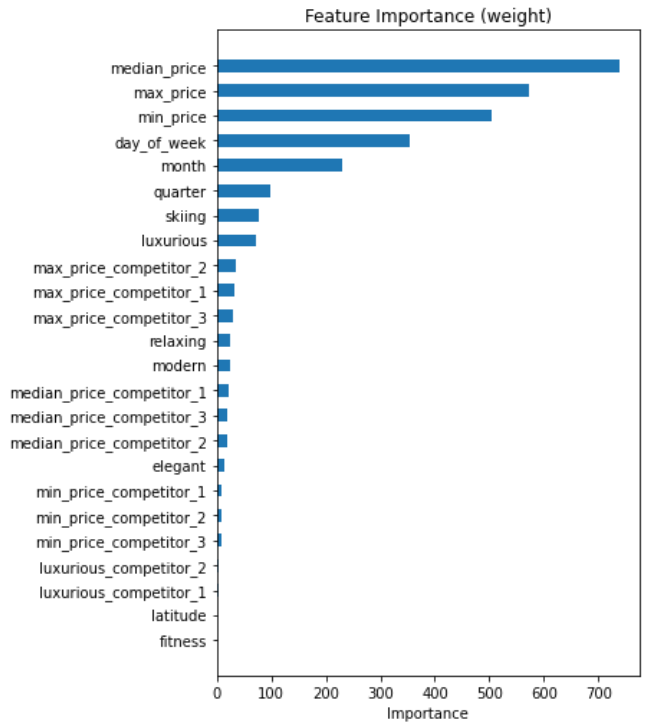
Resultate

Dabei wurde folgender R2 auf dem Cross Validation Set erreicht: [18]



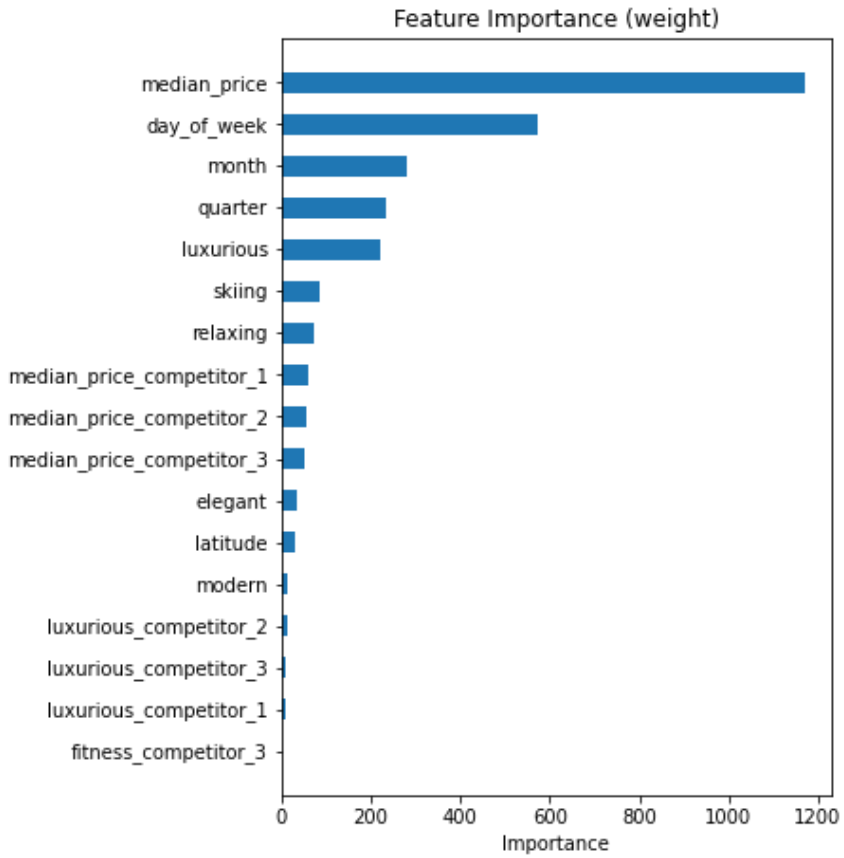
Die Nachfrage kann einigermassen gut vorhergesehen werden. Es gibt jedoch grosse Schwankungen.

[18] Run: http://127.0.0.1:5000/#/experiments/0/runs/971568474da047feb5c7c77c79a54e58

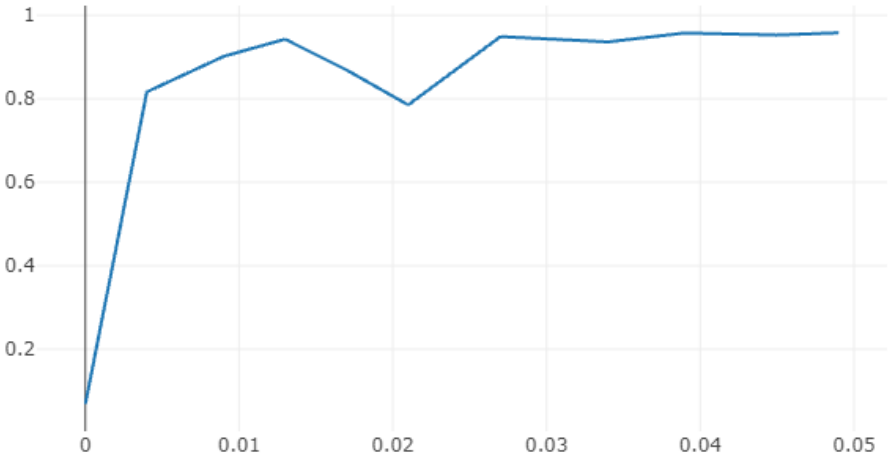


Die Feature Wichtigkeit zeigt sehr stark, dass die Nachfrage abhängig vom medianen Preis, maximalen und minimalen Preis abhängt.

Da der minimale Preis und der maximale Preis zur Inferenzzeit nicht vorhanden sind, werden diese aus den Features entfernt, um damit abschätzen zu können, inwiefern sich die Konkurrenzfeatures mehr aufdrängen.



Die Resultate sind besser und der Median Preis der Konkurrenz ist jetzt wichtiger. Jedoch ist immer noch tief. Wenn man die Performance ansieht, erhält man folgende Ergebnisse:



Die Konkurrenz hat nahezu keinen Einfluss auf die Nachfrage. Das heisst, mit diesem Modell kann nicht die Nachfrage abhängig von der Konkurrenz und dem eigenen Hotel bestimmt werden. Dies macht durchaus Sinn, da die Nachfrage unabhängig von diesen Hotels ist. Als andere Möglichkeit, könnten beispielsweise die Konkurrenzpreise aus dem Konkurrenzdatenset verwendet werden.

Feature-basierters XGBoost Modells mit Einbezug der Konkurrenzdaten (Konkurrenzdatensatz)
Da aufgrund der Konkurrenzsituation aus der Reservationsdaten kein signifikanter Einfluss gemessen wurde, wurde stattdessen die Konkurrenten aus dem Competitor Datenset (mit der grössten Feature Ähnlichkeit) als Konkurrenz gewählt.

Cross-Validation Modell
Datum: 12.05.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_4.ipynb, https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/commit/3b183b93dcb2a2ea14b53c6598451cf7899a4d7c

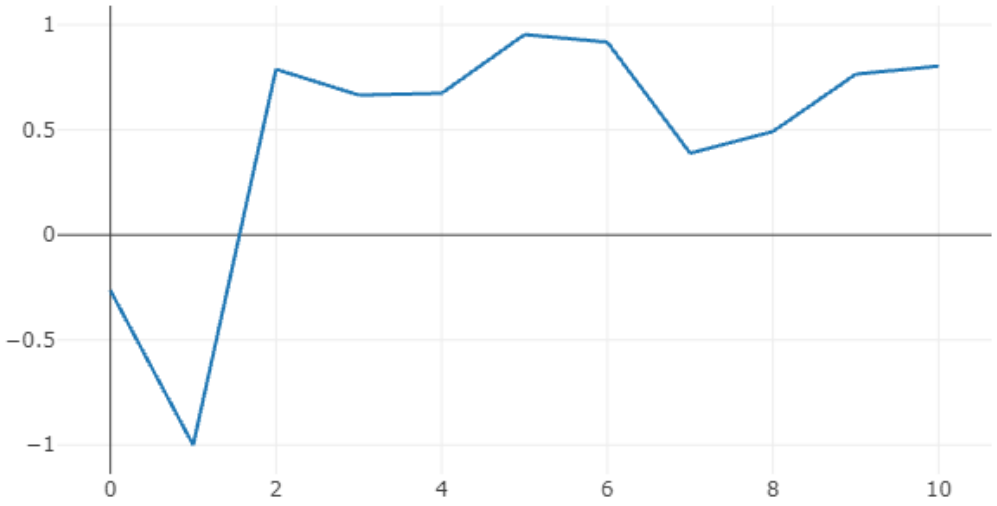
Aufbau
Folgende Features wurden verwendet:

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour', 'latitude', 'longitude', 'median_price', 'min_price', 'max_price', 'luxurious_competitor_1', 'elegant_competitor_1', 'relaxing_competitor_1', 'spacious_competitor_1', 'modern_competitor_1', 'restaurant_competitor_1', 'spa_competitor_1', 'pool_competitor_1', 'bar_competitor_1', 'fitness_competitor_1', 'leisure_competitor_1', 'gym_competitor_1', 'power_competitor_1', 'breakfast_x', 'compact_competitor_1', 'airport_competitor_1', 'ski_competitor_1', 'skiing_competitor_1', 'beach_competitor_1', 'view_competitor_1', 'desk_competitor_1', 'kitchenette_competitor_1', 'kettle_competitor_1', 'friendly_competitor_1', 'warm_competitor_1', 'business_competitor_1', 'hiking_competitor_1', 'cycling_competitor_1', 'bicycle_competitor_1', 'golf_competitor_1', 'fresh_competitor_1', 'playground_competitor_1', 'children_competitor_1', 'hairdryer_competitor_1', 'air conditioned_competitor_1', 'high speed_competitor_1', 'ski lift_competitor_1', 'coffee machine_competitor_1', '24 hour_competitor_1', 'price_avg', 'luxurious_competitor_2', 'elegant_competitor_2', 'relaxing_competitor_2', 'spacious_competitor_2', 'modern_competitor_2', 'restaurant_competitor_2', 'spa_competitor_2', 'pool_competitor_2', 'bar_competitor_2', 'fitness_competitor_2', 'leisure_competitor_2', 'gym_competitor_2', 'power_competitor_2', 'breakfast_x_competitor_2', 'compact_competitor_2', 'airport_competitor_2', 'ski_competitor_2', 'skiing_competitor_2', 'beach_competitor_2', 'view_competitor_2', 'desk_competitor_2', 'kitchenette_competitor_2', 'kettle_competitor_2', 'friendly_competitor_2', 'warm_competitor_2', 'business_competitor_2', 'hiking_competitor_2', 'cycling_competitor_2', 'bicycle_competitor_2', 'golf_competitor_2', 'fresh_competitor_2', 'playground_competitor_2', 'children_competitor_2', 'hairdryer_competitor_2', 'air conditioned_competitor_2', 'high speed_competitor_2', 'ski lift_competitor_2', 'coffee machine_competitor_2', '24 hour_competitor_2', 'price_avg_competitor_2', 'luxurious_competitor_3', 'elegant_competitor_3', 'relaxing_competitor_3', 'spacious_competitor_3', 'modern_competitor_3', 'restaurant_competitor_3', 'spa_competitor_3', 'pool_competitor_3', 'bar_competitor_3', 'fitness_competitor_3', 'leisure_competitor_3', 'gym_competitor_3', 'power_competitor_3', 'breakfast_x_competitor_3', 'compact_competitor_3', 'airport_competitor_3', 'ski_competitor_3', 'skiing_competitor_3', 'beach_competitor_3', 'view_competitor_3', 'desk_competitor_3', 'kitchenette_competitor_3', 'kettle_competitor_3', 'friendly_competitor_3', 'warm_competitor_3', 'business_competitor_3', 'hiking_competitor_3', 'cycling_competitor_3', 'bicycle_competitor_3', 'golf_competitor_3', 'fresh_competitor_3', 'playground_competitor_3', 'children_competitor_3', 'hairdryer_competitor_3', 'air

conditioned_competitor_3', 'high speed_competitor_3', 'ski lift_competitor_3', 'coffee machine_competitor_3', '24 hour_competitor_3', 'price_avg_competitor_3', 'quarter', 'month', 'year', 'day_of_week']

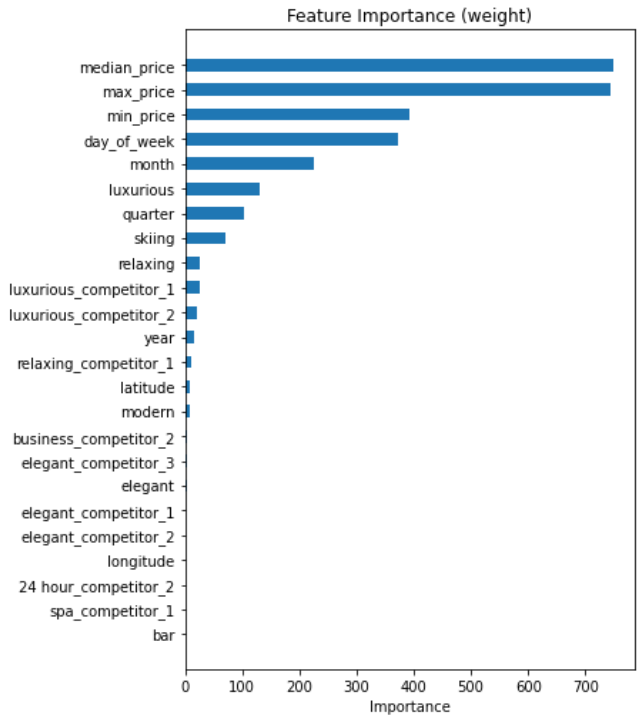
Resultate

Dabei wurde folgender R2 auf dem Cross Validation Set erreicht:



Die Nachfrage kann einigermassen gut vorhergesehen werden. Es gibt jedoch grosse Schwankungen.

Die Feature Wichtigkeit zeigt sehr stark, dass die Nachfrage abhängig vom medianen Preis, maximalen und minimalen Preis abhängt.



Die Konkurrenz hingegen hat wie zuvor keinen Einfluss auf die Nachfrage. Das heisst, mit diesem Modell kann nicht die Nachfrage abhängig von der Konkurrenz und dem eigenen Hotel bestimmt werden. Dies macht Sinn, da die Nachfrage unabhängig von diesen Hotels ist.

Feature-basiertes XGBoost Modells mit Einbezug der Konkurrenzdaten (Konkurrenzdatensatz aus dem selben Ort)

Da aufgrund der Konkurrenzsituation aus der Konkurrenzdatenset kein signifikanter Einfluss gemessen wurde, wurde stattdessen die Konkurrenten aus dem Competitor Datenset (mit demselben Ort -> Also alle Bristol) als Konkurrenz gewählt.

Cross-Validation Modell

Datum: 13.05.2022
Repository: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset
File: demand_model/demand_model_5.ipynb,

Aufbau

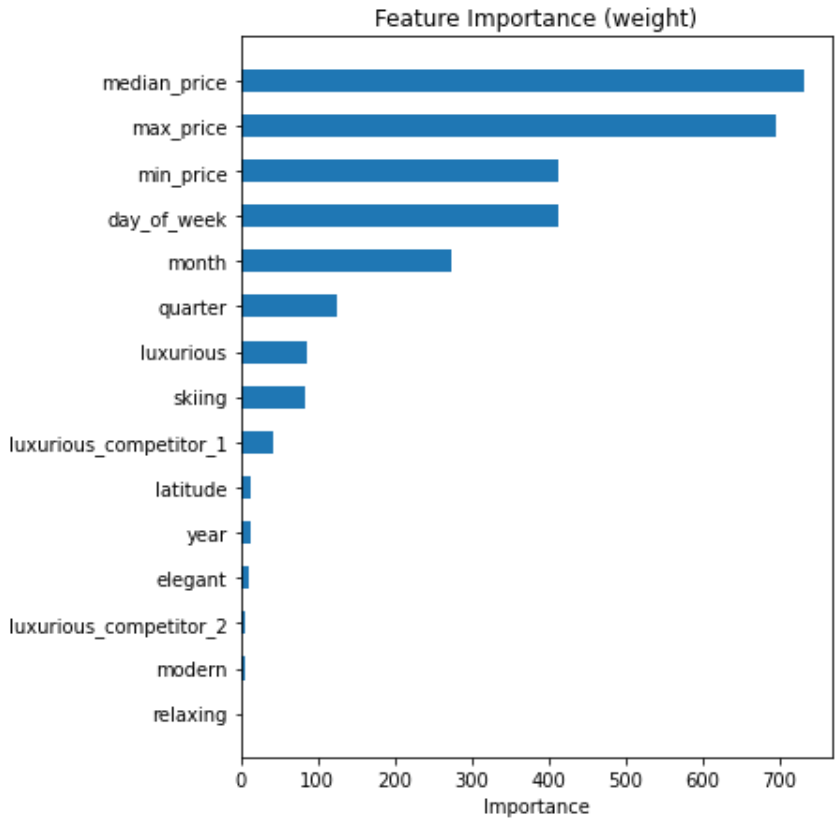
Folgende Features wurden verwendet:

['luxurious', 'elegant', 'relaxing', 'spacious', 'modern', 'restaurant', 'spa', 'pool', 'bar', 'fitness', 'leisure', 'gym', 'power', 'breakfast', 'compact', 'airport', 'ski', 'skiing', 'beach', 'view', 'desk', 'kitchenette', 'kettle', 'friendly', 'warm', 'business', 'hiking', 'cycling', 'bicycle', 'golf', 'fresh', 'playground', 'children', 'hairdryer', 'air conditioned', 'high speed', 'ski lift', 'coffee machine', '24 hour', 'latitude', 'longitude', 'median_price', 'min_price', 'max_price', 'luxurious_competitor_1', 'elegant_competitor_1', 'relaxing_competitor_1', 'spacious_competitor_1', 'modern_competitor_1', 'restaurant_competitor_1', 'spa_competitor_1', 'pool_competitor_1', 'bar_competitor_1', 'fitness_competitor_1', 'leisure_competitor_1', 'gym_competitor_1', 'power_competitor_1', 'breakfast_x', 'compact_competitor_1', 'airport_competitor_1', 'ski_competitor_1', 'skiing_competitor_1', 'beach_competitor_1', 'view_competitor_1', 'desk_competitor_1', 'kitchenette_competitor_1', 'kettle_competitor_1', 'friendly_competitor_1', 'warm_competitor_1', 'business_competitor_1', 'hiking_competitor_1', 'cycling_competitor_1', 'bicycle_competitor_1', 'golf_competitor_1', 'fresh_competitor_1', 'playground_competitor_1', 'children_competitor_1', 'hairdryer_competitor_1', 'air conditioned_competitor_1', 'high speed_competitor_1', 'ski lift_competitor_1', 'coffee machine_competitor_1', '24 hour_competitor_1', 'price_avg', 'luxurious_competitor_2', 'elegant_competitor_2', 'relaxing_competitor_2', 'spacious_competitor_2', 'modern_competitor_2', 'restaurant_competitor_2', 'spa_competitor_2', 'pool_competitor_2', 'bar_competitor_2', 'fitness_competitor_2', 'leisure_competitor_2', 'gym_competitor_2', 'power_competitor_2', 'breakfast_x_competitor_2', 'compact_competitor_2', 'airport_competitor_2', 'ski_competitor_2', 'skiing_competitor_2', 'beach_competitor_2', 'view_competitor_2', 'desk_competitor_2', 'kitchenette_competitor_2', 'kettle_competitor_2', 'friendly_competitor_2', 'warm_competitor_2', 'business_competitor_2', 'hiking_competitor_2', 'cycling_competitor_2', 'bicycle_competitor_2', 'golf_competitor_2', 'fresh_competitor_2', 'playground_competitor_2', 'children_competitor_2', 'hairdryer_competitor_2', 'air conditioned_competitor_2', 'high speed_competitor_2', 'ski lift_competitor_2', 'coffee machine_competitor_2', '24 hour_competitor_2', 'price_avg_competitor_2', 'luxurious_competitor_3', 'elegant_competitor_3', 'relaxing_competitor_3', 'spacious_competitor_3', 'modern_competitor_3', 'restaurant_competitor_3', 'spa_competitor_3', 'pool_competitor_3', 'bar_competitor_3', 'fitness_competitor_3', 'leisure_competitor_3', 'gym_competitor_3', 'power_competitor_3', 'breakfast_x_competitor_3', 'compact_competitor_3', 'airport_competitor_3', 'ski_competitor_3', 'skiing_competitor_3', 'beach_competitor_3', 'view_competitor_3', 'desk_competitor_3', 'kitchenette_competitor_3', 'kettle_competitor_3', 'friendly_competitor_3', 'warm_competitor_3', 'business_competitor_3', 'hiking_competitor_3', 'cycling_competitor_3', 'bicycle_competitor_3', 'golf_competitor_3', 'fresh_competitor_3', 'playground_competitor_3', 'children_competitor_3', 'hairdryer_competitor_3', 'air conditioned_competitor_3', 'high speed_competitor_3', 'ski lift_competitor_3', 'coffee

machine_competitor_3', '24 hour_competitor_3', 'price_avg_competitor_3', 'quarter', 'month', 'year', 'day_of_week']

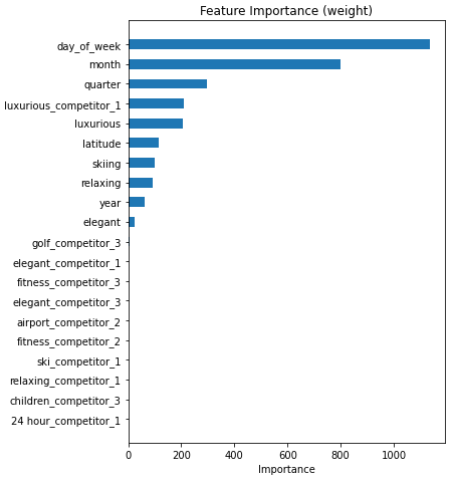
Feature-Wichtigkeiten

Auch hier wurden keine signifikanten Wichtigkeiten für die Konkurrenzeigenschaften entdeckt:



Als weiterer Schritt wurde versucht die Preisinformationen des Reservationsdatensets selbst zu entfernen, um zu schauen, ob dann die Konkurrenzinformationen relevanter sind.

Folglich wurden dieselben Features verwendet ausser min_price, max_price und median_price. Dabei gab es folgende Feature Wichtigkeiten:



Man sieht sehr klar, dass die Konkurrenzfeatures ebenfalls keinen grossen Einfluss haben. Folglich kann kein kombineirets Modell mit Einbezug der Konkurrenz entwickelt warden.

RL-Experimente

Übersicht Environments

Datum: 20.04.2022 -02.06.2022

Ziel: Dokumentation der Environments

Beschreibung

Es werden die verschiedenen Environments, die ausprobiert werden, dokumentiert.

Resultate

ID	# Konkurrenten	Konkurrenztyp	Rückkopplung	Nachfrage	Reservationspreis	State
1	3	Ortsname, erster	Nein	3 Kunden	Erster pro Tag	Preis Vortag, One Hot Beschreibung
2	3	**Orstname, erste drei Hotels mit Median Preis +- 10%, Setzen auf NaN anstatt 0**	Nein	3 Kunden	Erster pro Tag	Preis Vortag, One Hot Beschreibung
3	3	Orstname, erste drei Hotels mit Median Preis +- 10%, Setzen auf NaN anstatt 0	Nein	3-Kunden	Erster pro Tag	**Normalisierter Preis von -1 bis 1 für Preis**, One Hot Beschreibung
4	3	**3 ähnlichste gemäss Cos Similarität**	Nein	3 Kunden	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis, **Kontinuierliche Beschreibungen**
5	3	3 ähnlichste gemäss Cos Similarität	Nein	**Sommer Tag Nachfrage**, 150 Zimmer	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis, Kontinuierliche Beschreibungen
6	3	3 ähnlichste gemäss Cos Similarität	Nein	**Tourist und Business, Sommer Tag** Nachfrage, 150 Zimmer	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis, Kontinuierliche Beschreibungen
7	3, **Sampling mit Poisson**	3 ähnlichste gemäss Cos Similarität	Nein	Tourist und Business, Sommer Tag Nachfrage, 150 Zimmer	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis, Kontinuierliche Beschreibungen
8	3	3 ähnlichste gemäss Cos Similarität	Nein	Tourist und Business, Sommer Tag Nachfrage,	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis,

				Zimmer gemäss Daten, Konkurrenz gleich gross, Sampling der Nachfrage mit Poisson **Tiefster Preis erhält das Hotel, dann auffüllen!**		Kontinuierliche Beschreibungen
9	3	3 ähnlichste gemäss Cos Similarität	Nein	Tourist und Business, Sommer Tag Nachfrage, Zimmer gemäss Daten, Konkurrenz gleich gross, Sampling der Nachfrage mit Poisson Gewichtung **Gewichtung 0.5 mit Eigenschaften** ABIZ und tiefster Preis	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis, Kontinuierliche Beschreibungen
10	3	3 ähnlichste gemäss Cos Similarität	Nein	Tourist und Business, Sommer Tag Nachfrage, Zimmer gemäss Daten, Konkurrenz gleich gross, Sampling der Nachfrage mit Poisson Gewichtung **Gewichtung Random zwischen 0.1 und 0.9** mit Eigenschaften	Erster pro Tag	Normalisierter Preis von -1 bis 1 für Preis, Kontinuierliche Beschreibungen

				ABIZ und tiefster Preis		

Fazit

Die Resultate aus dem Experiment mit dem Environment 4 wurde mit statischem Nachfragemodell in die Dokumentation aufgenommen.

Aufsetzen RLLIB

Datum: 04.04.2022
Ziel: Erstes Dummy Projekt mit RLLib starten
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models

Beschreibung

Um erste Schritte mit RLLib zu erreichen, wird RLLib dockerisiert aufgesetzt.

Verwendete Quellen:

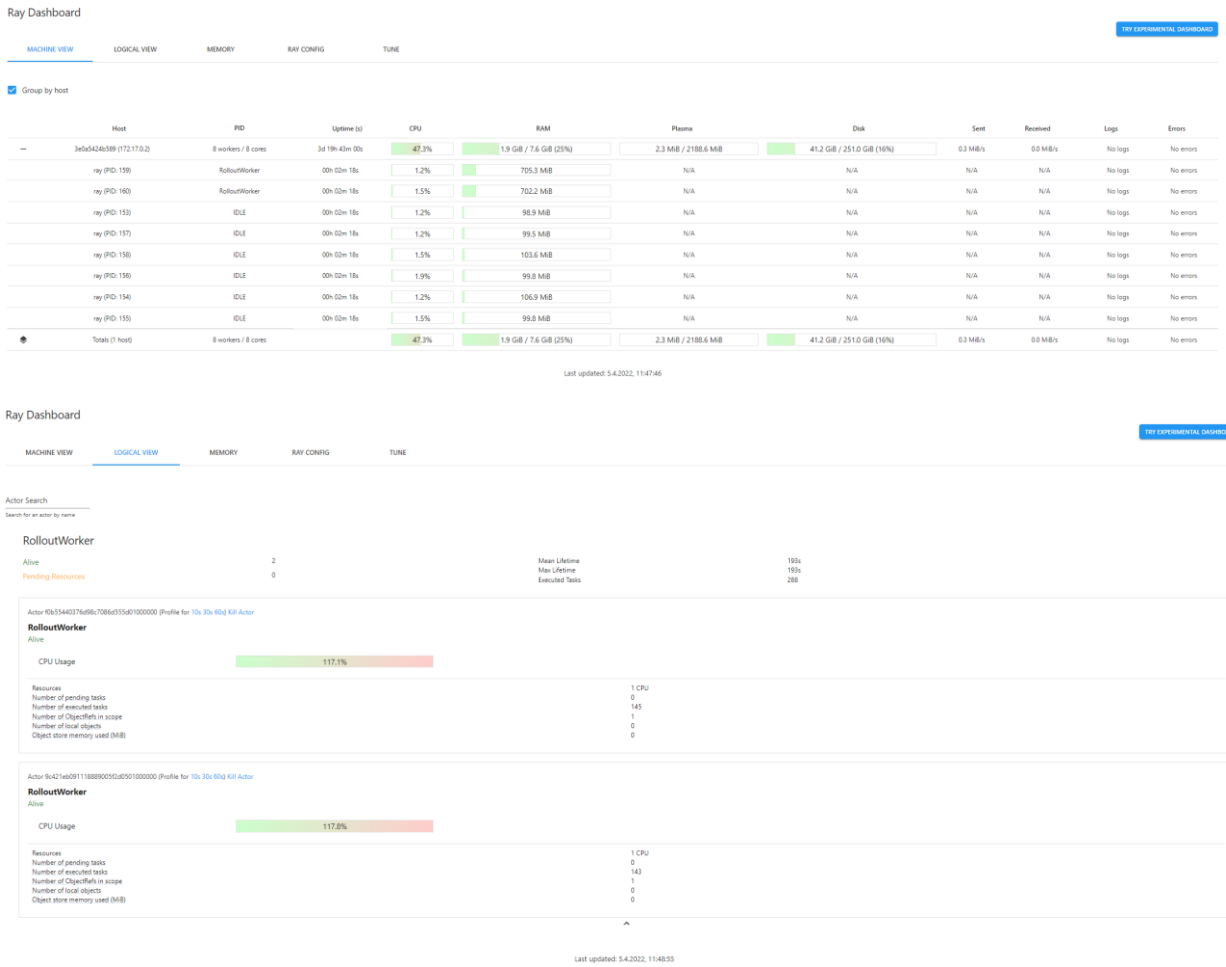
- https://docs.ray.io/en/latest/rllib/index.html
- https://github.com/ray-project/ray/blob/master/docker/ray/Dockerfile
- https://hub.docker.com/r/rayproject/ray
- https://docs.docker.com/language/python/build-images/

Vorgehen

1. Erstellung eines Dockerfiles: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/Dockerfile
2. Bauen des Files
3. Ausführen des Beispiels

Resultate

Es konnte das Taxi Problem erfolgreich mit einem Beispiel gelöst werden. Das Ray Dashboard konnte auf dem Host angezeigt werden:



Fazit

Das Aufsetzen ist intuitiv und damit können nun erste Konzepte gebaut werden.

RL Setup

Datum: 05.04.2022
Ziel: Definition der Schnittstellen für die Nachfrage.
File:

Beschreibung

In diesem Schritt wurde das RL Setup mit Hilfe von Interface genauer Beschrieben, um dieses in einem nächsten Schritt evaluieren zu können.

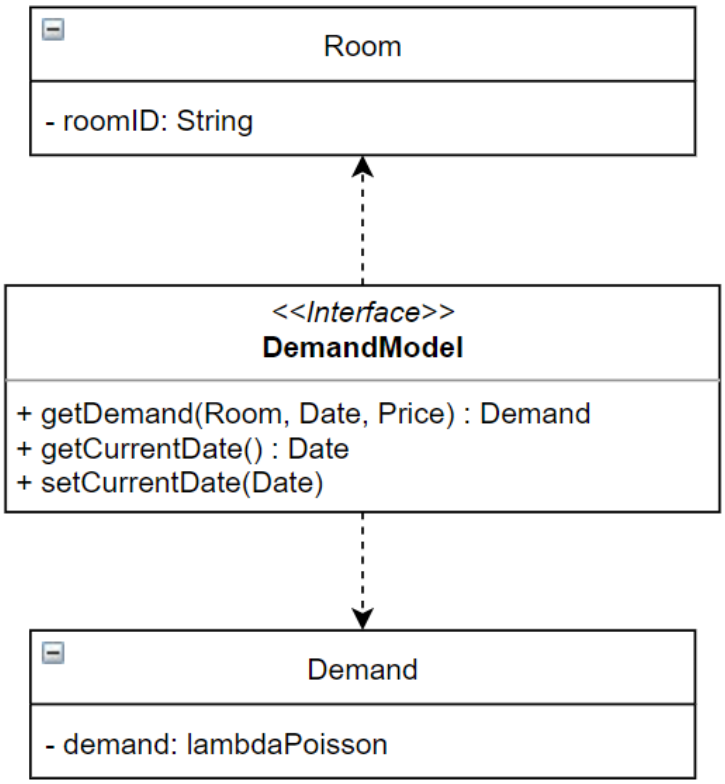
Vorgehen

1. Aufzeichnen der Nachfrage Schnittstellen
2. Aufzeichnen der Konkurrenz Schnittstellen

Resultate

Nachfragemodell

Das Nachfragemodell besteht aus den folgenden Entitäten:



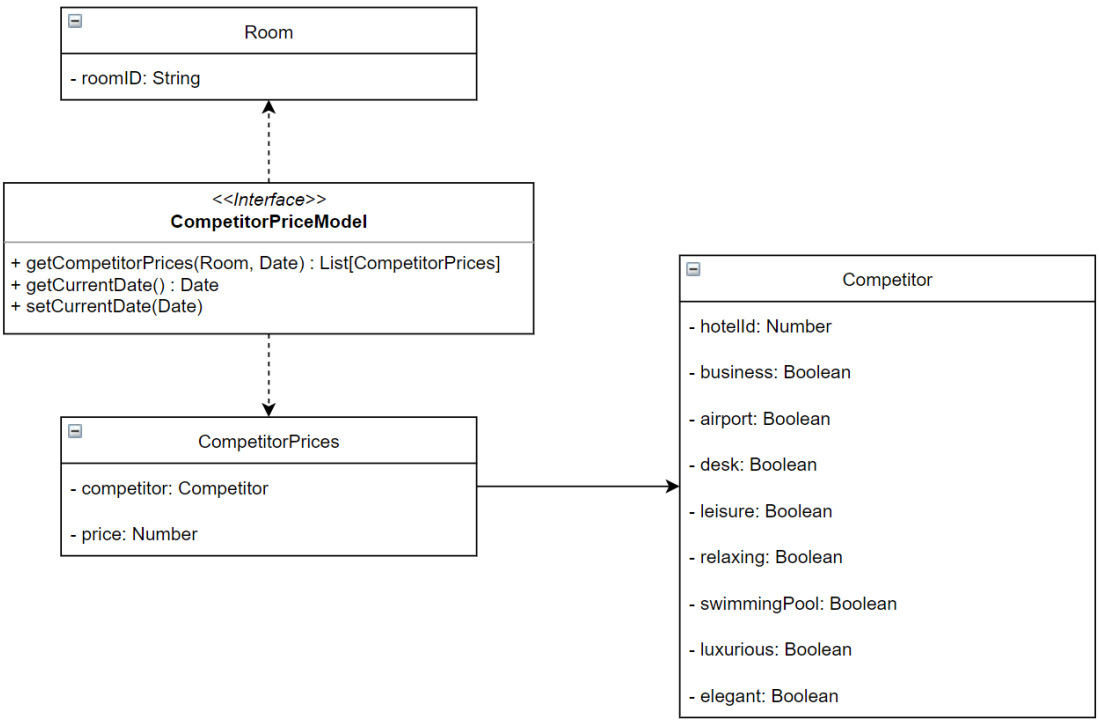
Beschreibung der Entitäten

Entität	Beschreibung	Kommentar
DemandModel	Modell, welches die Nachfrage berechnet. Hat intern einen aktuellen Tag, an dem diese Berechnung gemacht wird.	Gibt Nachfrage für einen Tag in der Zukunft zum aktuellen Tag heraus für einen Raum.

Room	Repräsentation eines Raumes inklusive eindeutiger ID	
Demand	Repräsentation der Nachfrage für einen Raum für einen gewissen Tag als Poisson Verteilung.	

Konkurrenzmodell

Das Nachfragemodell besteht aus den folgenden Entitäten:



Beschreibung der Entitäten

Entität	Beschreibung	Kommentar
CompetitorPriceModel	Modell, welches der Durchschnittspreis der Konkurrenz an einem bestimmten Tag in der Vergangenheit berechnet. Hat intern einen aktuellen Tag, an dem diese Berechnung gemacht wird.	Gibt Konkurrenzpreis für einen Tag in der Vergangenheit heraus für die Konkurrenz eines Raumes eines Hotels. Wie dieser Konkurrenzpreis berechnet wird, kann das Modell selbst bestimmen.
Room	Repräsentation eines Raumes inklusive eindeutiger ID	
CompetitorPrices	Preis welcher für diesen Tag angegeben wird	Gibt Preis und den Competitor zurück
Competitor	Beschreibung des Competitor anhand von Keywords	Beschreibung wird für State verwendet!

Feedback Daniel Pfäffli

- Demand abhängig vom Preis
- Poisson Verteilung für Demand (Man könnte samplen)
- Gemeinsamer Nenner: Entweder ID, Features Modell braucht

Erste Versuche mit CQL Conservative Q-Learning

Datum: 05.04.2022
Ziel: Erste Versuche mit Conservative Q-Learning zu machen

Beschreibung

Damit soll in einem ersten Versuch das CQL getestet werden.

Vorgehen

1. Generieren von Daten mit SAC Pendelum

```
rllib train -f examples/generate_data/pendulum-sac.yaml --no-ray-ui
```

Dabei muss aber sichergestellt werden, dass die Resultat JSONs der Läufe geloggt werden. Dazu muss ein mount erzeugt werden:

```
docker run -p 8265:8265 --mount
type=bind,src="$(pwd)"/src/examples/generate_data/results,target=/home/ray/ray
_results/pendulum-sac rl-models-test
```

2. Diese JSON Files können dann für CQL verwendet werden

Dazu wird das File: generate_data/results/SAC_Pendulum-v1_99747_00000_0_2022-04-06_00-17-21/output-2022-04-06_00-17-27_worker-0_0.json

Im

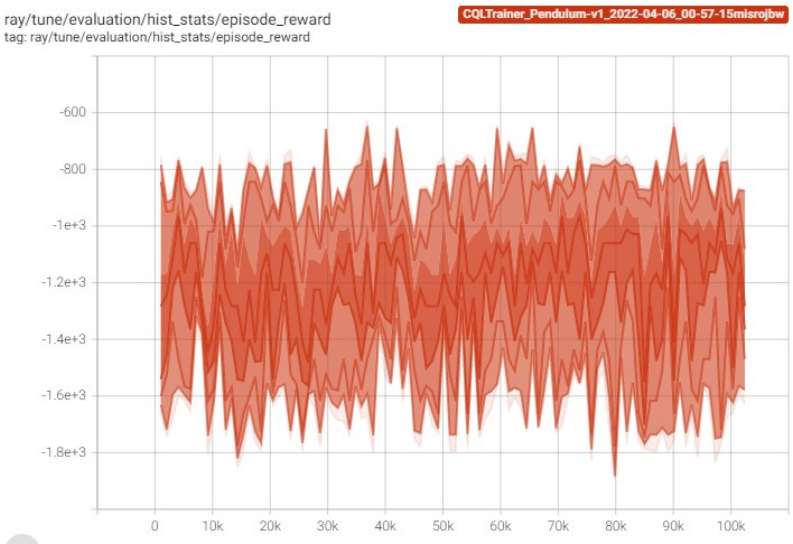
3. Die Resultate des Trainings können dabei mit dem TensorBoard angesehen werden.

```
sudo python3 -m tensorboard.main --logdir=src/examples/results/
```

Dies kann dann unter http://localhost:6006/ betrachtet werden.

Resultate

Es konnte zwar ausgeführt werden, jedoch waren die Resultate nicht besonders gut. Die Rewards wurden über die Zeit nicht besser!



Fazit

Es konvergiert zwar nicht, aber es können bereits erste Läufe gemacht werden!

Vorbereiten der Daten als Features

Datum: 07.04.2022 – 08.04.2022
Ziel: Vorbereiten der Features der Hotels und Agenten
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/data_preparation/competitor_and_demand_model.ipynb

Beschreibung

Vorbereiten der Features der Hotels und Agenten, damit sie für die Nachfrage- und Kundenmodelle verwendet werden können.

Vorgehen

1. Vorbereiten der Konkurrenzdaten
2. Vorbereiten der Nachfragedaten

Resultate

Die Daten wurden in einem vereinfachten File gespeichert.

Fazit

Dadurch können die Features einfach in das Modell reingeladen werden.

Erstes Generieren von Trainingsdaten für Conservative Q-Learning

Datum: 06.04.2022 – 15.04.2022
Ziel: Trainingsdaten generieren
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/tree/main/src/simulator

Beschreibung

In diesem Fall wird ein Environment aufgesetzt, wie das Trainingsdaten für den Conservative Q-Learning Algorithmus generiert werden.

Vorgehen

Dabei muss zuerst eine Simulationsinfrastruktur aufgebaut werden und der State, die Action und der Reward definiert werden.

Reward

Der Reward wird folgendermassen definiert:

$$Reward = Umsatz - Kosten$$

Wobei der Umsatz jeweils pro Tag berechnet wird und folgendermassen definiert, ist:

$$Umsatz = Anzahl\ verkaufte\ Zimmer * Preis\ pro\ Zimmer$$

Die Kosten des Hotels sind linear zur Anzahl der Zimmer. Kosten für die Benutzung der Zimmer werden ignoriert. Durch eine Einführung von Kosten wird das Anbieten von Zimmern attraktiv gemacht.

$$Kosten = Anzahl\ Zimmer * Kosten\ pro\ Zimmer$$

Dabei wird jeweils der Demand anhand des Demand Modells berechnet.

Bestimmung der erhaltenen Gäste

Jedes Hotel hat 4 Zimmer. In einem ersten Schritt werden jeden Tag 3 Kunden simuliert. Jeder Kunde hat ein eigenes Entscheidungsmotiv:

Student

- Nimmt ausschliesslich immer das Zimmer mit dem tiefsten Preis

Tourist

- Gibt eine 50% Gewichtung für seine Entscheidung daran, wie hoch die Summe von Leisure, Relaxing, Swimming Pool und Airport ist (Maximal 4)
- Ist bereit einen Preis von 50 bis 100 Franken pro Nacht zu zahlen. Wenn es darüber oder darunter ist, kommt er sicher nicht. Je näher der Preis bei 50 Franken ist, desto eher kommt er. Dieser Wert berechnet sich aus: (100 Gesetzter Preis) / 12.5 (Maximal 4)

Das heisst der Tourist wählt das Zimmer, welches den folgenden Wert maximiert

$$Value = \begin{cases} 0.5 * (Leisure + Relaxing + Swimming\ Pool + Airport) + 0.5 * ((200 - Price)/25), & 200 >= \text{Preis} >= 100 \\ 0 & ,sonst \end{cases}$$

Business

- Nimmt eher ein teureres Hotel (50% Gewichtung) Teuerstes Hotel = 4 Punkte, Zweitteuerstes Hotel = 3 Punkte, … (Maximal 4)
- Gibt eine 50% Gewichtung für seine Entscheidung daran, wie hoch die Summe von Business, Airport, Desk, Luxurious, Elegant ist (Maximal 5)

Die Entscheidung welches Hotel, welche Kunden erhält, wird jeweils gemacht, sobald alle den Preis gesetzt haben.

State

Als State werden folgende Merkmale definiert.

State	**Beschreibung**	**Wertebereich**	**Gegeben durch**
Datum	Das aktuelle Datum	0-365 Für 1.1.2020 -31.12.2020	Inkrementierend / Reservationen Datenset
Hotel Id	Die HotelId	0-7 für alle Hotels	
Preis des Vortages	Preis des Agenten des Vortags	Diskrete Zahlen von 0-500	Agent
Showpreis 1	Preis der Konkurrenz 1 des Vortags	Diskrete Zahlen von 0-500	Konkurrenzmodell
Showpreis 2	Preis der Konkurrenz 2 des Vortags	Diskrete Zahlen von 0-500	Konkurrenzmodell
Showpreis 3	Preis der Konkurrenz 3 des Vortags	Diskrete Zahlen von 0-500	Konkurrenzmodell
Agent und Konkurrenz 1 - 3			
Business	Flag, ob es Business ist oder nicht	0 oder 1	Beschreibung
Airport	Flag, ob es in der Nähe des Airports ist	0 oder 1	Beschreibung
Desk	Flag, ob es ein Tisch hat oder nicht	0 oder 1	Beschreibung
Leisure	Flag, ob es ein Leisure Hotel ist oder nicht	0 oder 1	Beschreibung
Relaxing	Flag, ob es Relaxing ist oder nicht	0 oder 1	Beschreibung
Swimming pool	Flag, ob es ein Swimmng Pool hat oder nicht	0 oder 1	Beschreibung
Luxurious	Flag, ob es luxurious ist oder nicht	0 oder 1	Beschreibung
Elegant	Flag, ob es elegant ist oder nicht	0 oder 1	Beschreibung

Action

Die Aktion des Agenten ist jeweils die Festsetzung des Preises auf den aktuellen Tag. Als kontinuierlicher Wert zwischen 0 und 500.

Agent

Als Agent, werden alle Hotels in Bristol verwendet. Folgende Hotel Ids:

1709, 1527, 1098, 1099, 1100, 1101, 1102,1103

Dabei muss sichergestellt werden, dass folgende Schnittstelle sichergestellt wird:

```
class MyEnv(gym.Env):
    def __init__(self, env_config):
        self.action_space = <gym.Space>
        self.observation_space = <gym.Space>
    def reset(self):
        return <obs>
    def step(self, action):
        return <obs>, <reward: float>, <done: bool>, <info: dict>
```

https://docs.ray.io/en/latest/rllib/rllib-env.html

Wenn das Environment geladen wurde im CQL Learner gab es Fehler beim Init. Deshalb wurde der Gym Env Checker verwendet:

https://stable-baselines.readthedocs.io/en/master/common/env_checker.html

File "c:\src\school\baa\rl-models\src\simulator\main.py", line 69, in <module>

run_cql()

File "c:\src\school\baa\rl-models\src\simulator\main.py", line 67, in run_cql

run_cql_runner()

File "c:\src\school\baa\rl-models\src\simulator\runners\cql_runner.py", line 80, in run_cql_runner

trainer = cql.CQLTrainer(config=config)

vs\env-rllib\lib\site-packages\ray\rllib\policy\dynamic_tf_policy.py", line 403, in __init__

auto_remove_unneeded_view_reqs=True)

File "C:\Users\User\anaconda3\envs\env-rllib\lib\site-packages\ray\rllib\policy\dynamic_tf_policy.py", line 696, in _initialize_loss_from_dummy_batch

losses = self._do_loss_init(train_batch)

File "C:\Users\User\anaconda3\envs\env-rllib\lib\site-packages\ray\rllib\policy\dynamic_tf_policy.py", line 779, in _do_loss_init

losses = self._loss_fn(self, self.model, self.dist_class, train_batch)

File "C:\Users\User\anaconda3\envs\env-rllib\lib\site-packages\ray\rllib\agents\cql\cql_tf_policy.py", line 104, in cql_loss

policy_t, log_pis_t = action_dist_t.sample_logp()

Der Grund für diesen Fehler war, dass CQL keine Diskreten Action Space erlaubt. Deshalb musste dieser auf kontinuierlich angepasst werden.

Zusätzlich musste eine mindestlänge an Daten generiert werden, ansonsten gab es folgenden Fehler: «Value has to be dict»

Zusätzlich muss die Action als Liste zurückgegeben werden.Die Fehler wurden eingegrenzt indem einfache Fake Environments erzeugt wurden!

Zeitaufwand: Ca. 8 Stunden

Resultate

Es konnte ein erstes JSON generiert werden.

https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/generated_data/output-2022-04-15_14-13-32_worker-0_0.json

Fazit

In einem ersten Schritt konnten Daten generiert werden. Diese sind nur sehr klein und sehr statisch. Dennoch sollten diese in einem nächsten Schritt für ein erstes Training mit CQL verwendet werden.

Feedback Reza

Demand Function: Initialize different Distributions

Student: More lowest price For each student different center! (Distribution over Date) -> Number of nights
Business : 5*
Tourist: Middle Price Neighbourhood Attractions

Number of nights, date, duration, …

1 Student

1 Business Man

1 Tourist

4 Rooms -> 4 Hotels 4 Rooms (1 is RL, 3 others) (2 Weeks Simulation)

3 Variables for Distance to Airport

State

Number of Stars Competitors

Distance to Airport, Business

Price of Competitors

https://docs.ray.io/en/latest/rllib/rllib-offline.html

Verwenden von eigenen Daten innerhalb des CQL

Datum: 12.04.2022 – 19.04.2022
Ziel: Die generierten Daten innerhalb des CQL Algorithmus verwenden.
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/runners/cql_runner.py

Beschreibung

In diesem Fall sollen die zuvor erzeugten Daten im CQL Algorithmus verwendet werden.

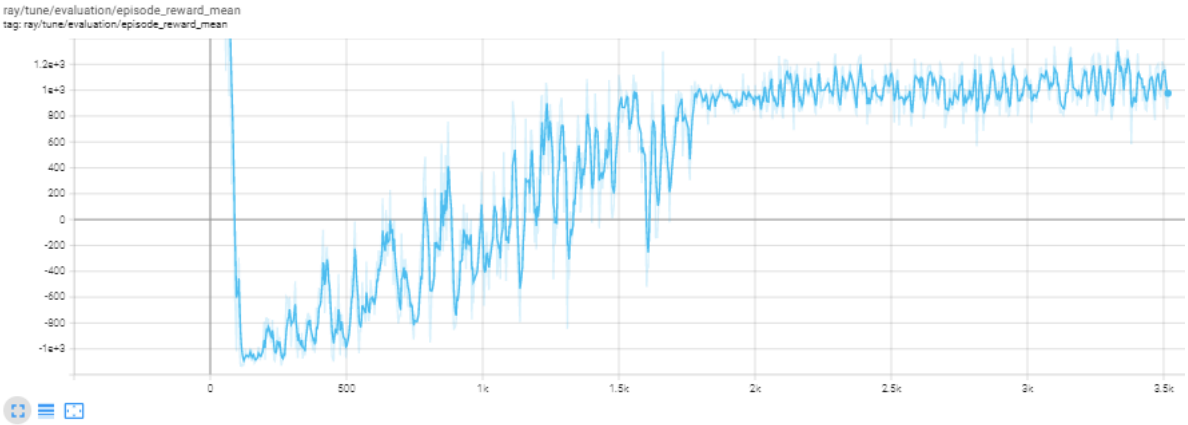
Vorgehen

1. Zuerst wurde versucht, das CQL Runner Beispiel von zuvor erneut einzubeziehen
2. Anschliessend wurde das eigene Environment einbezogen

Resultate

Es gab sehr viele Probleme beim Ausführen! Siehe oben!

Mit den ersten Versuchen war sehr klar sichtbar, dass der Reward in der Evaluierung zu Beginn über die Zeit immer mehr abgenommen hat. Jedoch verbessert sich der Agent immer mehr.



Bei einer Analyse der gesetzten Aktionen fällt auf, dass je länger der Agent trainiert wird, desto mehr tendieren die gesetzten Preise gegen 0 Franken pro Nacht gesetzt. Jedoch beginnt nach einer gewissen Zeit sich die Performance auch wieder zu verbessern und der mittlere Reward steigt wieder an. Dies wird erreicht, indem der Preis immer mehr erhöht wird. Jedoch bleibt der Algorithmus schlussendlich in einer Preisrange zwischen 65 und 150 Franken pro Nacht.

Dabei bleibt der Reward bei ca. 1000. Hier wird klar sichtbar, dass das Hotel zum Teil noch den Touristen abholt sich aber sehr stark auf das Business Hotel fokussiert.

Fazit

Es gibt bereits eine sehr gute erste Sicht, wie es funktioniert. Jedoch ist das Environment noch sehr einfach.

Verbessern der Daten

Datum: 19.04.2022
Ziel: Bessere Daten generieren für die Simulation.
File:

Beschreibung

Um mehr Daten und diversere Daten zu ermöglichen, sollen die Daten besser generiert werden.

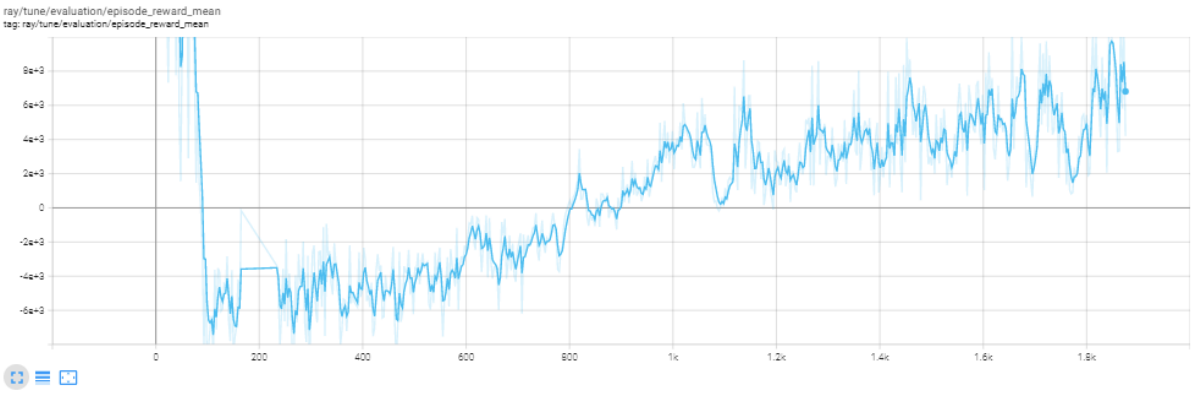
Vorgehen

1. Alle Hotels aus Bristol miteinbeziehen
2. Alle Daten aus 2020 verwenden
3. Überprüfen, dass Datum für dieses Hotel im Reservationsset vorhanden ist
4. Mehr Daten generieren
5. CQL Trainieren und Reward Kurve ansehen

Resultate

CQLTrainer_my_env_2022-04-19_10-21-06jjxklz55

Es scheint eine gewisse Maximierung des Rewards je nach Länge der Episode von 1700 bis 10000 zu entwickeln.



In diesem Fall schwanken aber die Preise sehr viel stärker zwischen 23 bis 500 Franken pro Nacht.

Fazit

Es scheint auch bei komplexeren Daten eine Tendenz zu geben, dass das Modell was lernt. Dieses Verhalten kann dann mit dem folgenden Test betrachtet werden.

Speichern/Bereitstellen und Validieren des Modells

Datum: 19.04.2022
Ziel: Aufruf des Modells durch API
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/validator/runner_validator.py

Beschreibung

Um ein trainiertes Modell auszutesten, wird das Modell gespeichert und bereitgestellt. Ebenfalls soll dabei für einen zufälligen Zeitraum das Modell mit den Reservationsdaten verglichen.

Vorgehen

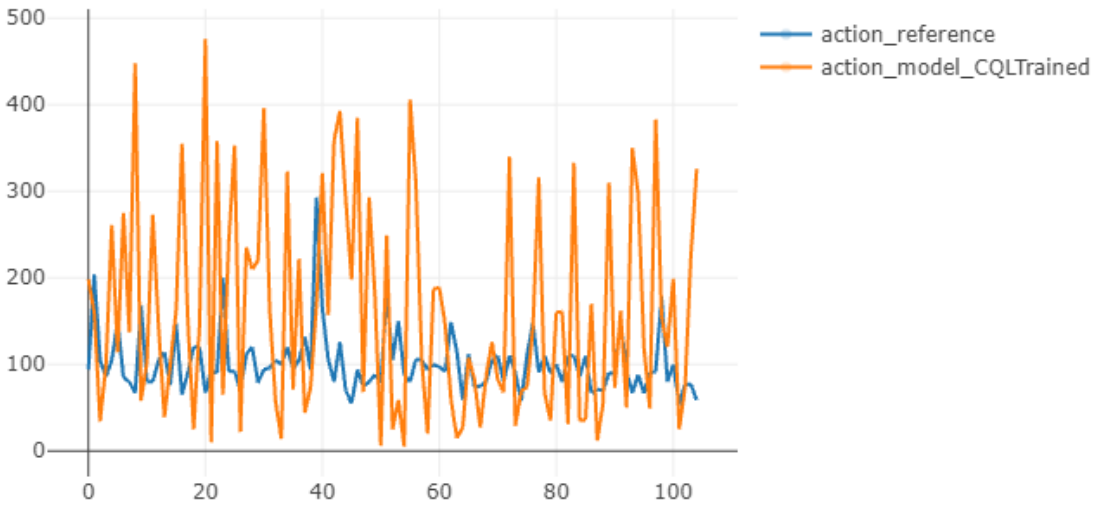
1. Speichern des Modells mittels trainer.save("/tmp/rllib_checkpoint")
2. Laden des Modells https://docs.ray.io/en/latest/serve/tutorials/rllib.html
3. Vergleichen der Modelle
4. Loggen der Resultate

Resultate

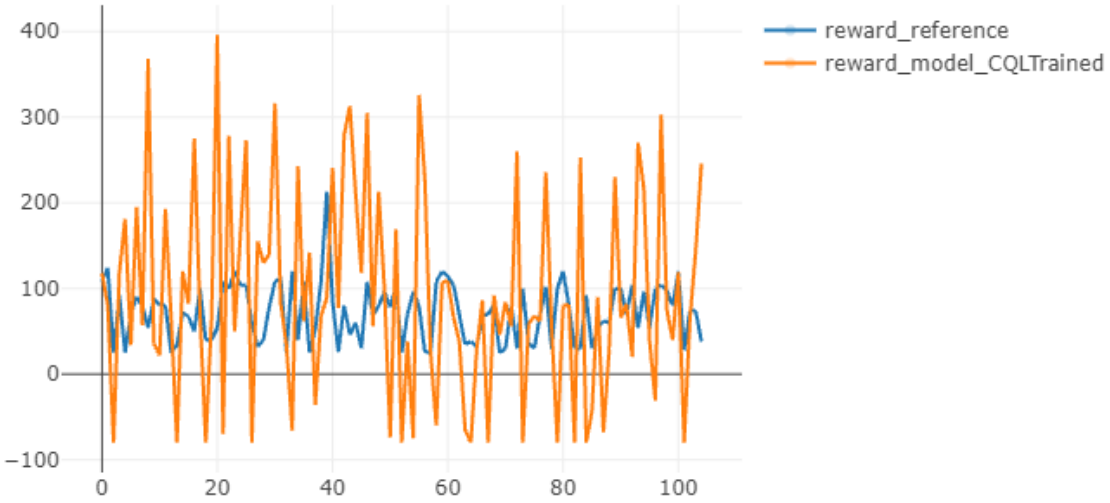
MLflow

2020-09-17T00:00:00 - 2020-12-31T00:00:00 für Hotel Id: 1098

Man kann jetzt die Aktionen der Modelle vergleichen.



Ebenfalls die Rewards, dabei erreicht das Hotel folgende Rewards: Referenz: 7559, Modell: 9751



Fazit

Es ermöglicht es sehr gut, dass man einen trainierten Agent mit dem Reservations Agent vergleichen kann.

Analyse des Verhaltens an einem bestimmten Zeitpunkt mit Environment 1

Datum: 20.04.2022

Ziel: Für die Weihnachtszeit wird das Verhalten des CQL Modells mit dem Reservation-Datensatz verglichen.

File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/validator/runner_validator.py , https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/f7f2cac7e9ce991318c466bc13d5cce91d5ed6d2

Beschreibung

Um die Performance des Modells zu überprüfen, wird für zwei Wochen Ende 2020 das Modell verglichen.

Environment Id: 1

Vorgehen

1. Erstellen eines fixen Preis Modells
2. Setzten des Envs auf 17.12.2020 für das Hotel 0 -> 1709
3. Simulation für sowohl das Fix-Preis Modell, das Reservations Modell und das Competitor Modell

Resultate

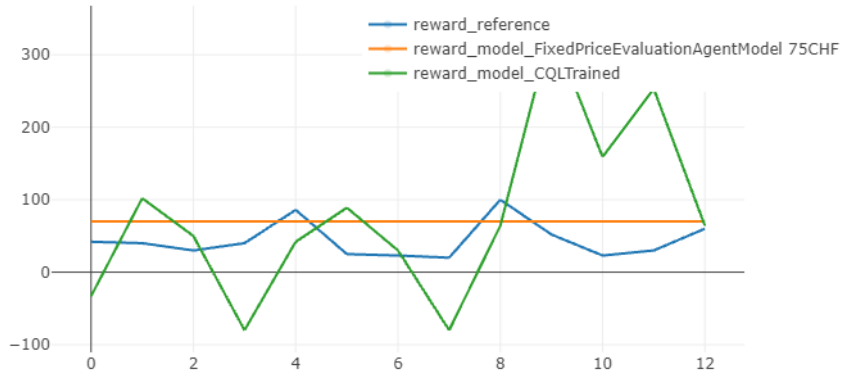
Environment: Version 1
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

Rewards

Folgende Totale Rewards wurden erzielt:

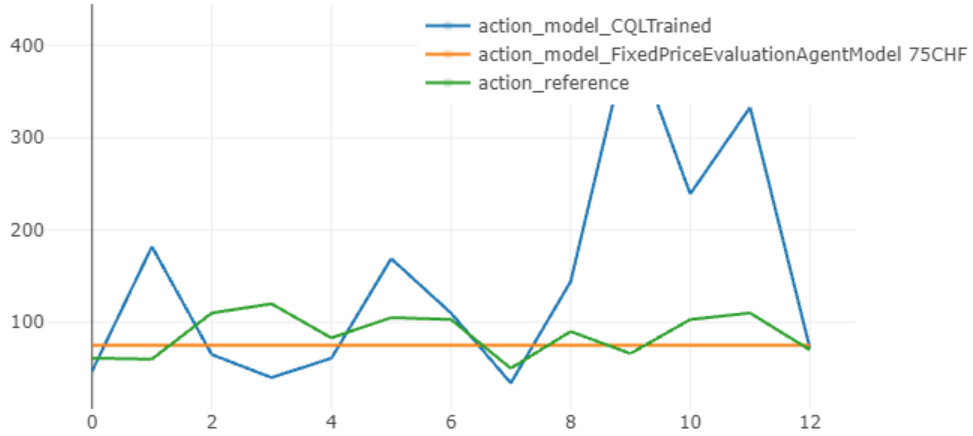
Referenz: 571
CQL: 997
Fixer Preis 75: 910

Über die Zeit gibt es jeweils folgende Rewards die erzielt wurden:



Dabei ist sehr spannend zu sehen, dass das CQL Modell sehr häufig Verluste erzielt und erst an den Tagen 9, 10 und 11 profitabel wird. Dies sind die Tage 27-29. Dezember 2020. Die Preise, die durch die Modelle jeweils gesetzt wurden, sehen folgendermassen aus:

Hierbei ist zu beachten, dass die Farben nicht dieselben sind!



Man sieht, dass das CQL Modell gegen Ende sehr stark versucht den Businesskunden zu erhalten. Dabei setzt er sehr hohe Preise.

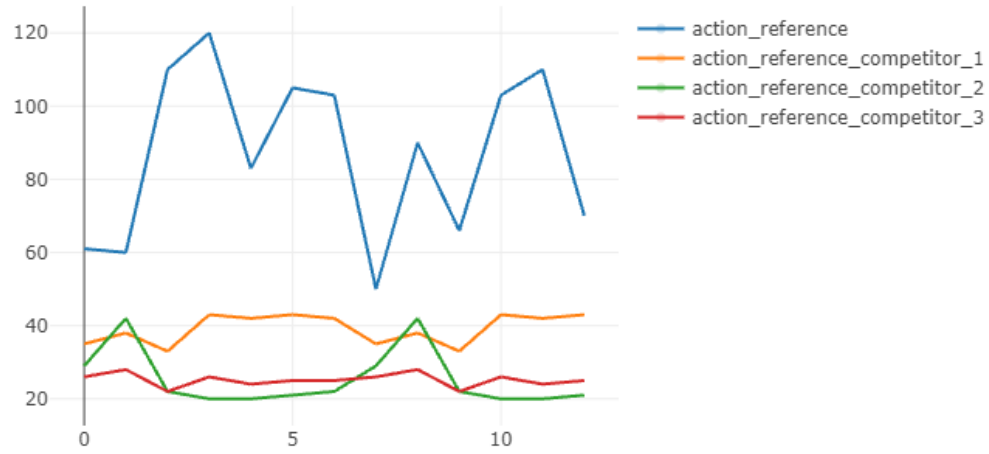
Dies macht er sehr wahrscheinlich, weil er bereits einen Vorteil hat durch seine Eigenschaften gegenüber der Konkurrenz.

Description Competitor 1	Business:0 Airport:1 Desk:0 Leisure:0 Relaxing:0 Swimming Pool:0 Luxurious:0 Elegant:0
Description Competitor 2	Business:0 Airport:1 Desk:0 Leisure:0 Relaxing:1 Swimming Pool:0 Luxurious:0 Elegant:0
Description Competitor 3	Business:0 Airport:1 Desk:1 Leisure:0 Relaxing:0 Swimming Pool:0 Luxurious:0 Elegant:0
Description Hotel	Business:1 Airport:0 Desk:0 Leisure:0 Relaxing:0 Swimming Pool:0 Luxurious:0 Elegant:1

So ist er viel mehr auf den Businesskunden ausgerichtet.

Fazit

Es gibt sehr interessante Resultate und der CQL Trainer erzeugt bereits spannende Resultate. Jedoch sind die Schwankungen noch sehr stark in den Resultaten und die Konkurrenz ist sehr unterschiedlich zum eigentlichen Hotel. Dies sieht man in den Preisen, die die Konkurrenz setzt.



Diese sind signifikant tiefer als das eigene Hotel. Deshalb muss die Konkurrenz besser definiert werden umso interessantere Resultate zu erhalten. Ebenfalls müssen damit die Modelle länger trainiert werden.

Bestimmung der Konkurrenz anhand des Median Preises und Setzen von NaN statt

Datum: 20.04.2022
Ziel: Bessere Konkurrenz Env 2 generieren
File:

Beschreibung

Die Auswahl der Konkurrenz soll anhand des Median Preises gewählt werden, um somit interessantere Vergleiche zu ermöglichen und wirkliche Konkurrenzsituationen generieren. Zusätzlich gibt es Rückkopplung innerhalb des Systems.

Vorgehen

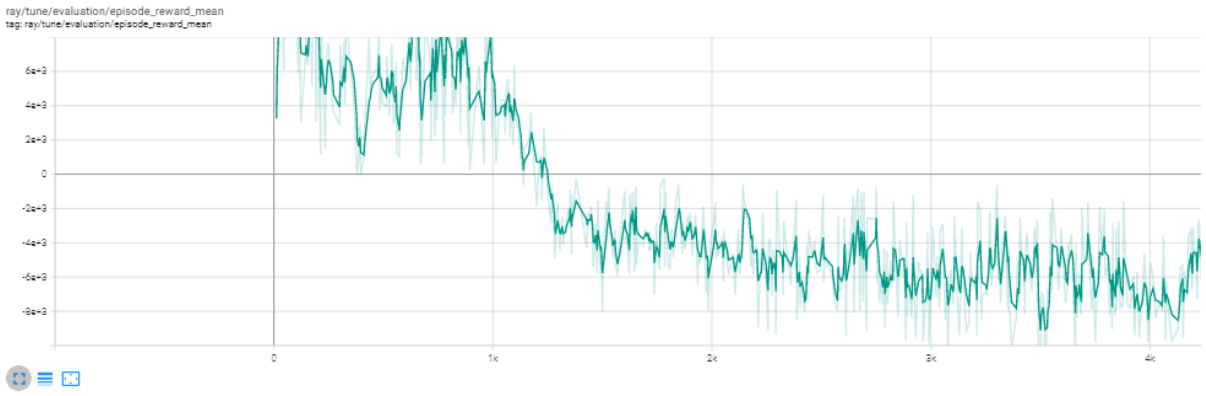
1. Auslesen des Median Preises des Agent Hotels
2. Suchen der ersten drei Hotels aus gleichem Ort mit Median Preis +- 10%
3. **Setzten der unbekannten Features auf np.NaN statt 0, sorgt für bessere Resultate**
    a. Behebt, dass die Preise der Konkurrenz nicht mehr so tief sind
4. Updaten der Version auf v2

Resultate

Environment: Version 1
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

CQL Training

Problem gab es vor allem dabei, dass es nicht mehr konvergiert. Das heisst der Mean Reward bleibt auf sehr tiefem Niveau stehen.



Wenn man dies nach 2300 Iterationen evaluiert, erhält man bei den zwei Wochen folgende Resultate:

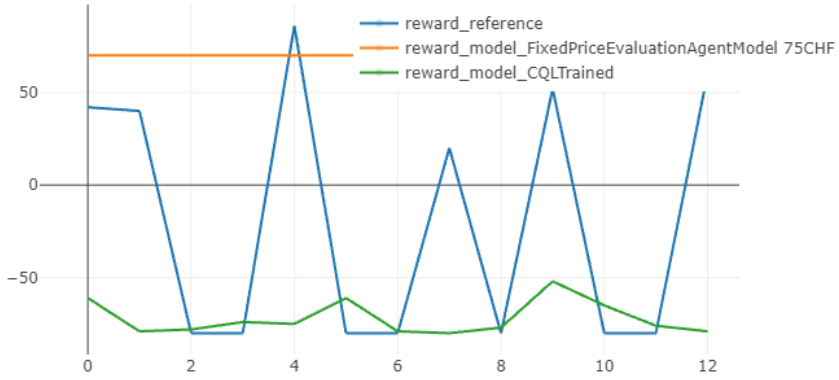
Rewards

Folgende Totale Rewards wurden erzielt:

Referenz: -260
CQL: -936
Fixer Preis 75: 910

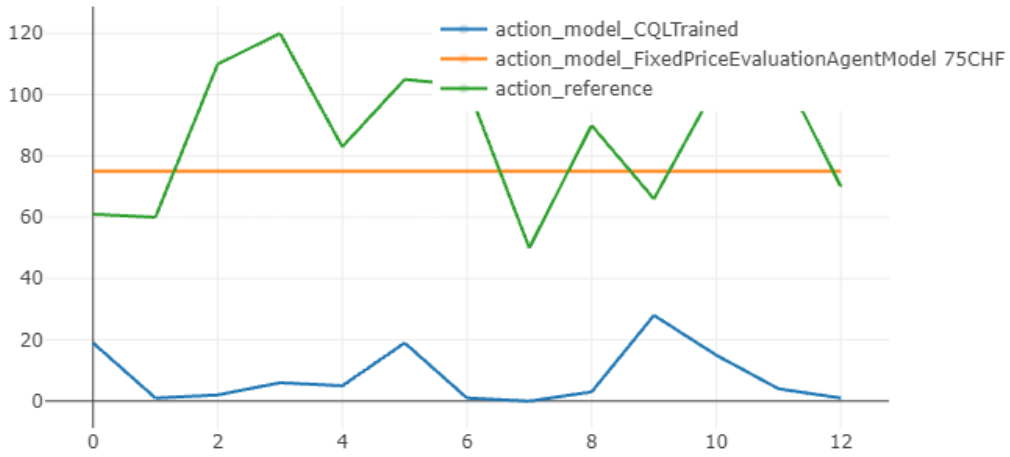
Resultate Checkpoint nach 2300 Iterationen

Über die Zeit gibt es jeweils folgende Rewards die erzielt wurden:



Dabei ist sehr spannend zu sehen, dass das CQL Modell ausschliesslich Verluste erzielt. Das Referenzmodell performt sehr ähnlich erzielt dabei aber einen höheren Reward.

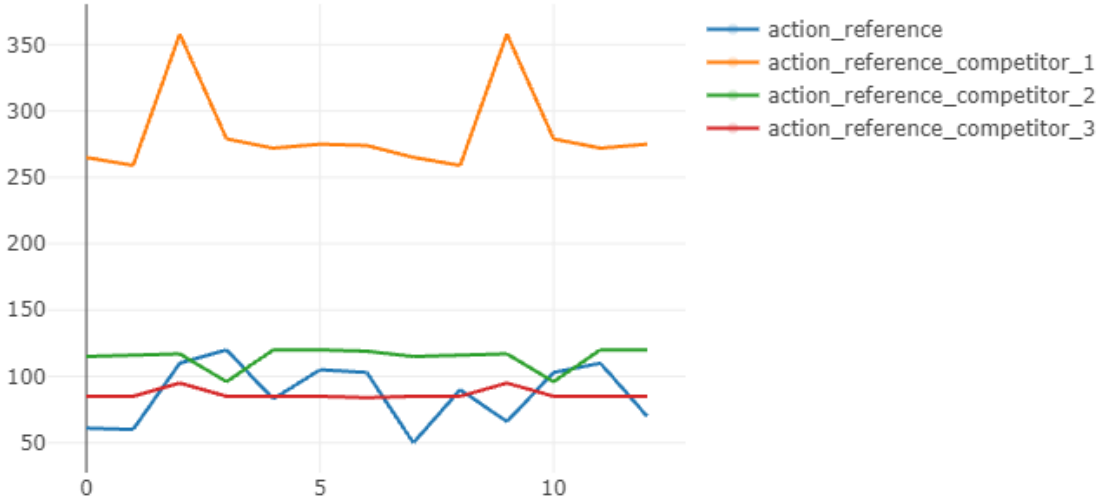
Hierbei ist zu beachten, dass die Farben nicht dieselben sind!



Man sieht, dass das CQL Modell sehr tiefe Preise setzt. Dies lässt sich nicht wirklich erklären. Jedoch ist es möglich, dass die Grösse des Replay Buffers zu klein ist und somit nur ein sehr kleiner Teil des Konkurrenzverhaltens effektiv abgebildet werden kann.

Description Competitor 1	Business:0 Airport:0 Desk:0 Leisure:1 Relaxing:0 Swimming Pool:0 Luxurious:1 Elegant:1
Description Competitor 2	Business:0 Airport:1 Desk:0 Leisure:0 Relaxing:0 Swimming Pool:0 Luxurious:0 Elegant:1
Description Competitor 3	Business:0 Airport:0 Desk:1 Leisure:0 Relaxing:0 Swimming Pool:0 Luxurious:0 Elegant:0
Description Hotel	Business:1 Airport:0 Desk:0 Leisure:0 Relaxing:0 Swimming Pool:0 Luxurious:0 Elegant:1

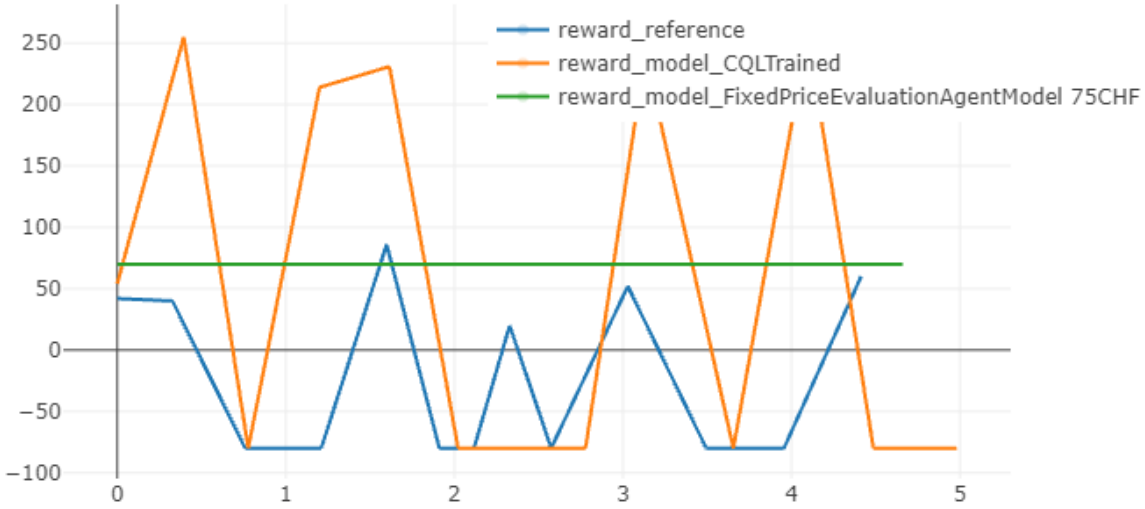
Man kann auch nicht direkt erkennen, warum es dies getan hat. So scheint es doch sehr ähnliche Eigenschaften wie der Konkurrent 1 zu haben und wäre eigentlich für den Businesskunden sehr gut. Jedoch lernt er es nicht, dass er hohe Preise setzen muss und folglich geht der Business Kunde an den Konkurrent 1 und es besteht höchstens die Möglichkeit, dass er den Studenten anspricht.



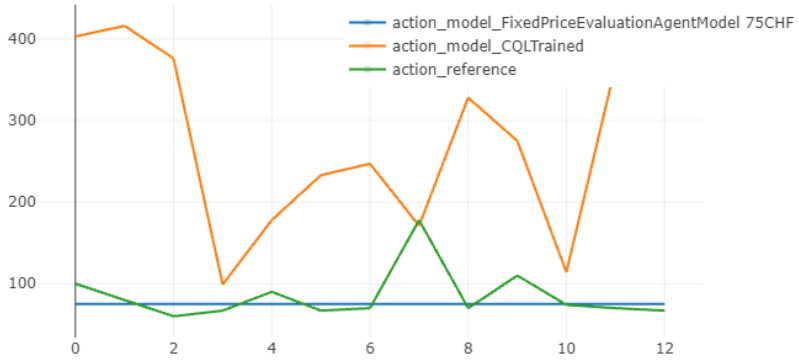
Wenn man das Verhalten vergleicht mit einem Checkpoint nach 200 Iterationen erhält man folgendes Resultat:

Resultate Checkpoint nach 200 Iterationen

Es scheint sehr ähnliches Verhalten zu haben, wie das Referenzdatenset jedoch macht es eher höhere Preise was zu einem höheren Reward führt.



Dies sieht man auch an den Resultaten der Aktionen, die ausgeführt werden, das Modell lernt hauptsächlich höhere Preise zu setzen.



Fazit

Es gibt nicht zuverlässige Resultate, dies könnte jedoch auch am angepassten Environment liegen, welches einen Competitor erzeugt welcher mehr Daten hat. Eine Alternative dafür wäre aber auch, dass man den Replay Buffer vergrössert und so mehr Experiences rein lädt. Denn es war ausschliesslich eine Buffer Size von 10000 festgelegt, das zu Grunde liegende Datenset hat aber über 30'000 Sample Batches. Zusätzlich ist es sehr spannend zu sehen, dass die Performance sich über die Zeit verschlechtert.

Jedoch könnte es auch sein, dass zu wenig lange trainiert wird und das Lernen der höheren Preisen nicht passiert.

Erhöhung des Replaybuffers auf 100000

Datum: 21.04.2022 – 22.04.200
Ziel: Erhöhung des Replaybuffers
File:

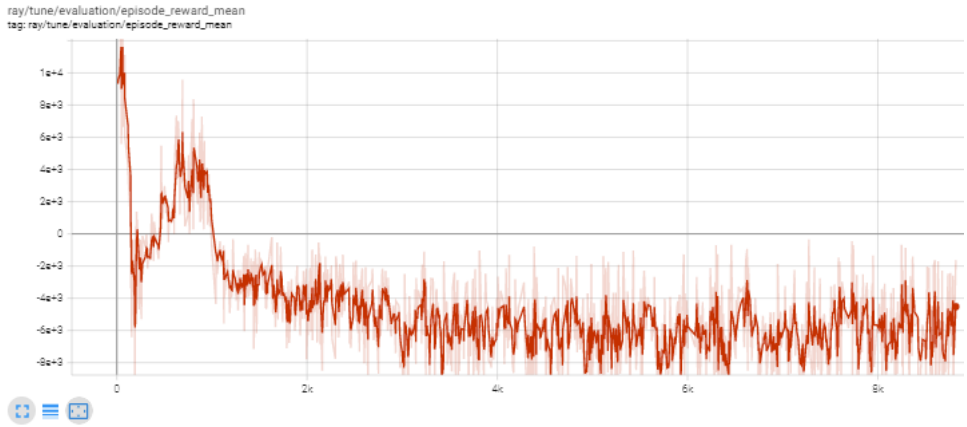
Beschreibung

Um das ganze Datenset reinzuladen, wird der Replaybuffer von 10000 auf 100'000 erhöht. Dadurch wird es möglich, dass aus mehr Experiences gelernt wird und evt. bessere Resultate erzielt werden.

Vorgehen

1. Erhöhung des Replaybuffers auf 100'000
2. Ausführen des CQL Algorithmus auf 2900 Iterationen
3. Evaluieren auf dem Datenset

Resultate



Es scheint erneut ein Konvergieren auf sehr tiefem Niveau zu geben.

Fazit

Als Anpassung werden folgende Schritte versucht:

1. Normalisieren des Preises

2. Eher 3 Layer mit 64 -> Nicht so viel über Vergangenheit!
- > Je tiefer desto weniger Nichtlinearität

3. Learning Rate: Evt tiefer

Normalisieren der Preise

Datum: 22.04.2022
Ziel: Normalisieren der Preise
File:

Beschreibung

Das Ziel ist, dass die Inputs für die Preise normalisiert sind, das heisst, dass die Preise zwischen -1 und 1 liegen.

Vorgehen

Dazu wird das folgende Mapping gemacht:

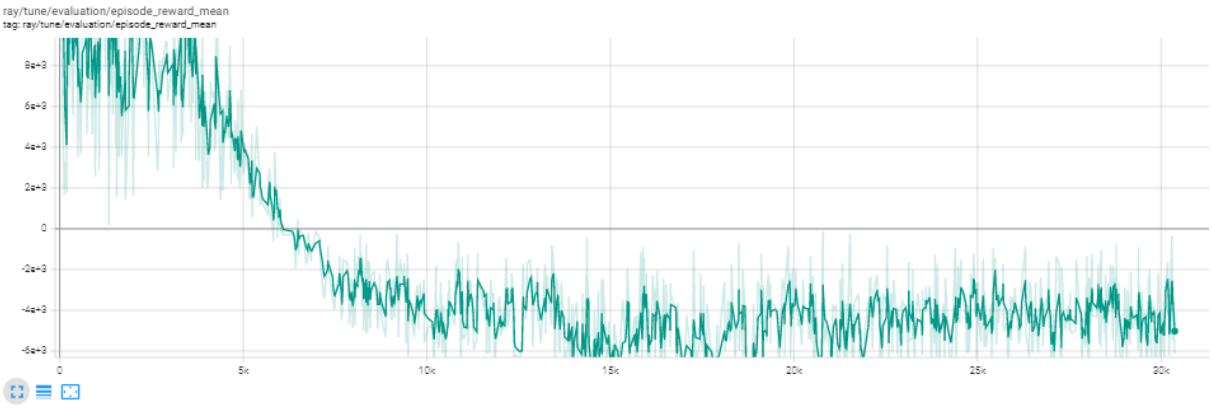
0 : -1
500 : 1

Ebenfalls musste der Observation Space angepasst werden, damit man den Preis kontinuierlich abspeichern kann. Dieser ist nun ein Box Space.

Dies sorgte für das Environment 3.

Resultate

Man sieht sehr stark, dass auch bei einer Normalisierung der Preise ein sehr ähnliches Verhalten entsteht und es ein Konvergieren auf tiefem Niveau gibt.



Fazit

Eine andere Möglichkeit wäre es tiefere Netze zu verwenden, um somit mehr nicht lineares Verhalten zu verwenden.

Tieferes Netzwerk

Datum: 26.04.2022
Ziel: Tieferes Netzwerk
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/main/src/simulator/runners/cql_runner.py

Beschreibung

Zusätzlich zur Normalisierung wurde das Netzwerk tiefer gemacht. Dadurch soll zusätzliches nicht lineares Verhalten gelernt werden.

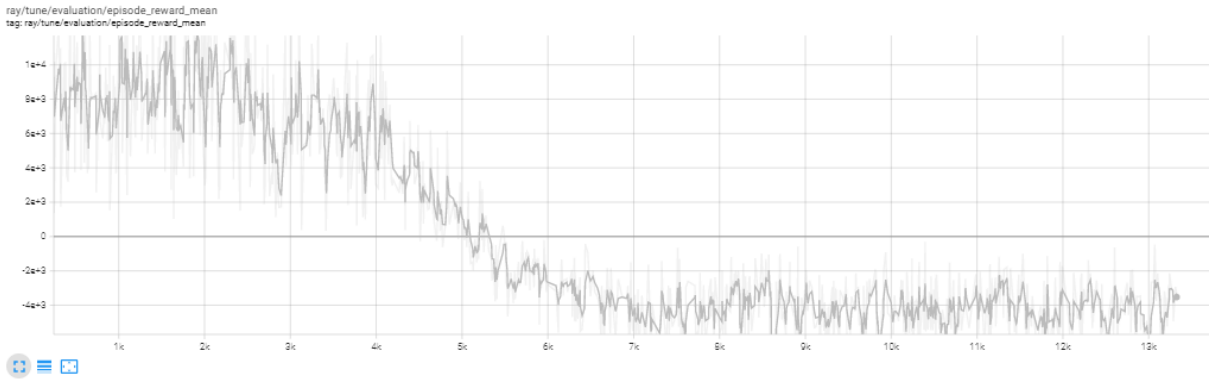
Vorgehen

Verändern der Konfiguration des Netzwerks auf 64 x 64 x64

```
config["Q_model"] = {
    "fcnet_hiddens": [64, 64, 64],
    "fcnet_activation": "relu",
}
config["policy_model"] = {
"fcnet_hiddens": [64, 64, 64],
"fcnet_activation": "relu",
}
```

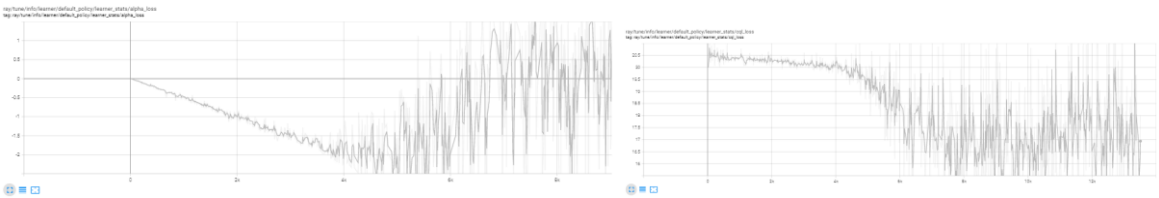
Resultate

Nach 4400 Episoden gab es folgendes Ergebnis:



Es scheint immer noch eine Konvergenz auf tiefem Niveau zu geben.

Sowohl der Alpha Loss, wie auch der CQL Loss beginnen ab dem Punkt wo sich die Performance verschlechtert stark zu schwanken:



Fazit

Es sollte eventuell länger trainiert werden und mehr Batches auf einmal trainiert werden. Dadurch soll die Abhängigkeit von einem Sample reduziert werden.

Längeres Training und höhere Batch Size

Datum: 25.04.2022
Ziel: Längeres Training und höhere Batch Size
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/90c1c89162b94d416bc639580df8cfab906e482d

Beschreibung

In einem weiteren Versuch wurde das Modell länger trainiert. Dies wurde erreicht, indem die Evaluation statt jedes Mal nur alle 50 Trainingsschritte gemacht wird. Ebenfalls wurde der Train Batch Size erhöht von 1 auf 256. Dadurch soll eine tiefere Varianz erzielt werden. Dadurch soll eine tiefere Schwankung erreicht werden.

Vorgehen

Es wurde weniger evaluiert, um ein schnelleres Training und damit längeres Training zu ermöglichen:

Längeres Training:

- config["evaluation_interval"] = 1
- config["evaluation_interval"] = 50

Zusätzlich wurde die Batch Size erhöht, um mehr Experiences pro Trainingsschritt miteinzubeziehen. Ebenfalls soll dies die Varianz pro Trainingsschritt reduzieren.

Erhöhung der Batch Size:

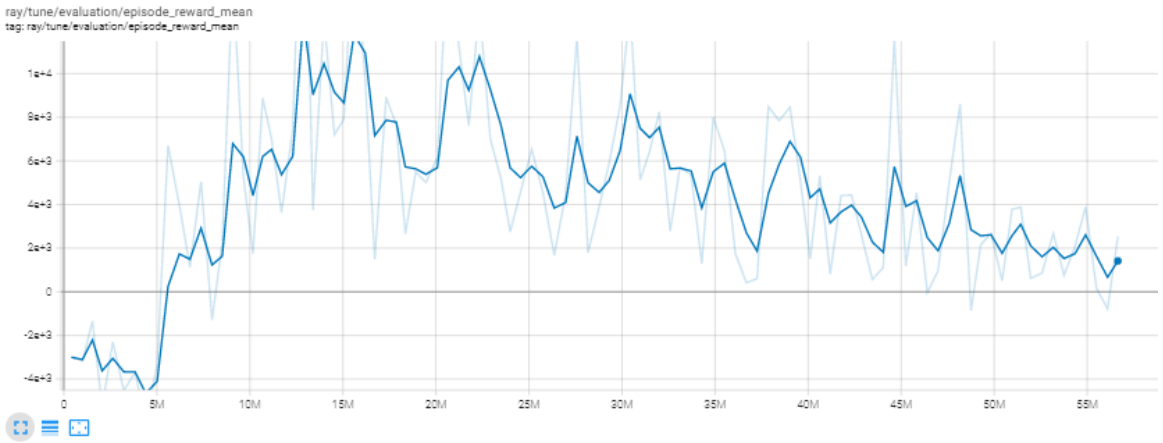
- Vorher: config["train_batch_size"] = 1
- Nachher: config["train_batch_size"] = 256

Resultate

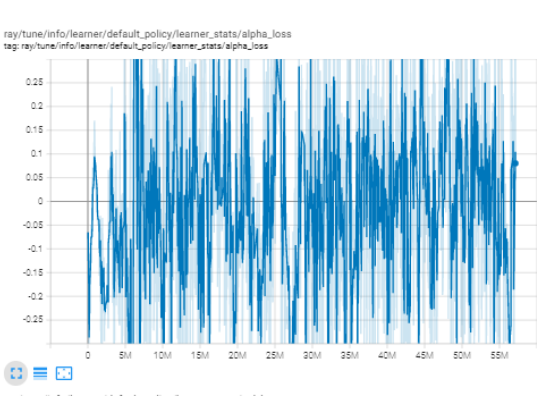
CQL Training

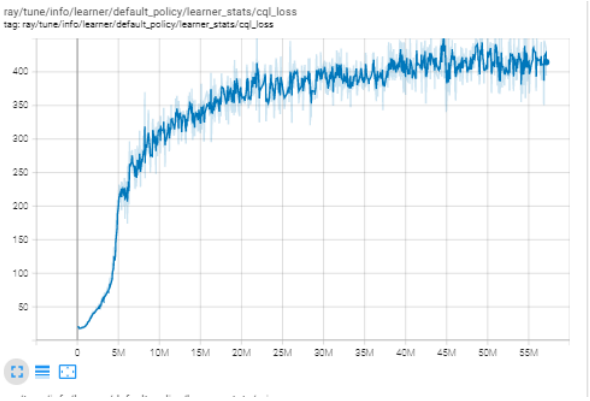
Im Training sieht man sehr stark, dass nun sehr viel mehr Experiences geladen werden und mehr Steps gemacht werden. Die Entwicklung des Rewards über 5000 Iterationen mit jeweils 256 Batches sieht es folgendermassen aus:

Die Mean Rewards sind im Training positiv, was zeigt, dass jetzt etwas gelernt wird.

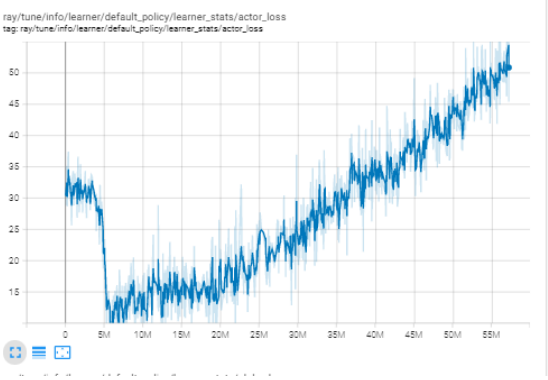


Der Alpha Loss und der CQL Loss stabilisieren sich. Dies zeigt eine gewisse Konvergenz.





Jedoch scheint es nach einer gewissen Zeit wieder zu einer Verschlechterung der Performance zu geben. Dies zeigt sich auch in der Entwicklung des Actor Losses, welcher sehr stark anzusteigen scheint.



Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

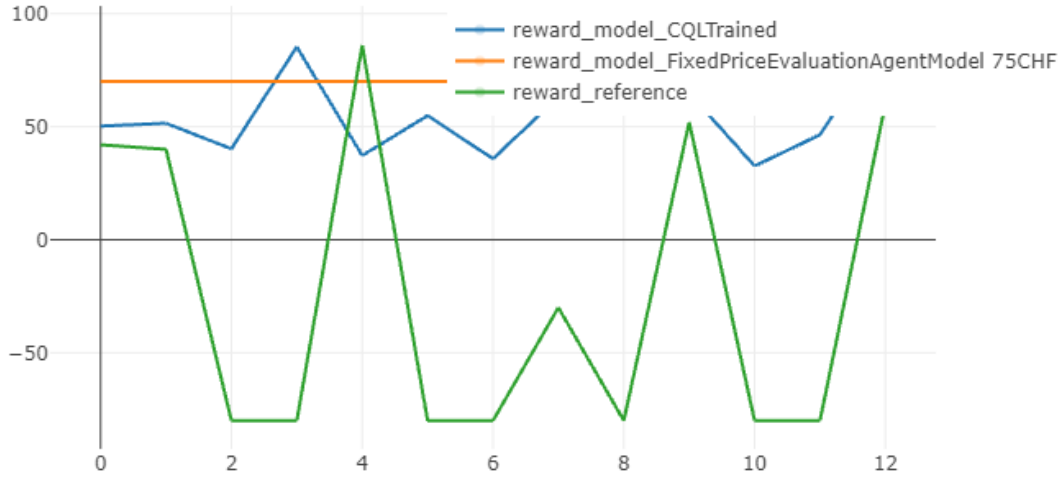
Rewards

Environment: Version 3
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

Folgende Totale Rewards wurden erzielt:

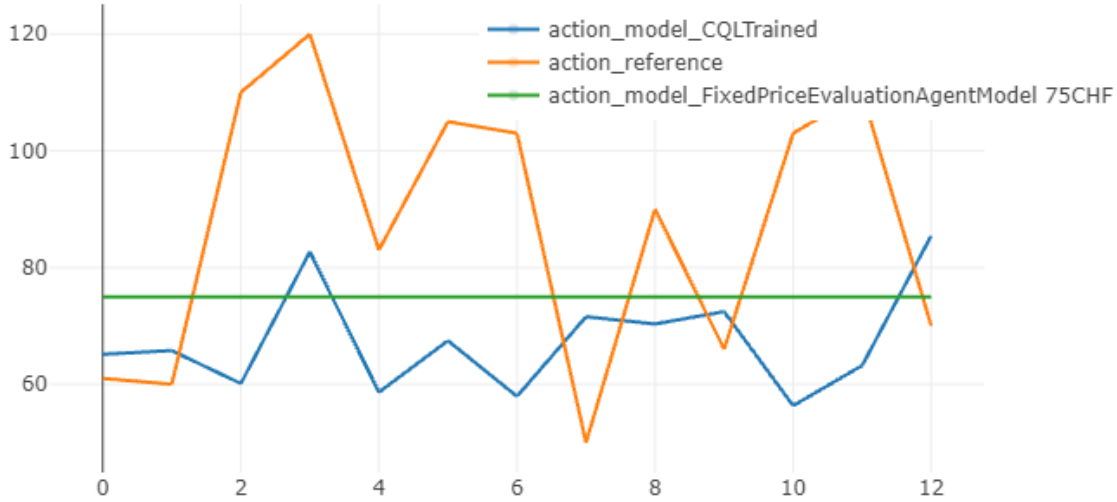
Referenz: -310
CQL: 714.1
Fixer Preis 75: 910

Über die Zeit wurden folgende Rewards erzielt:

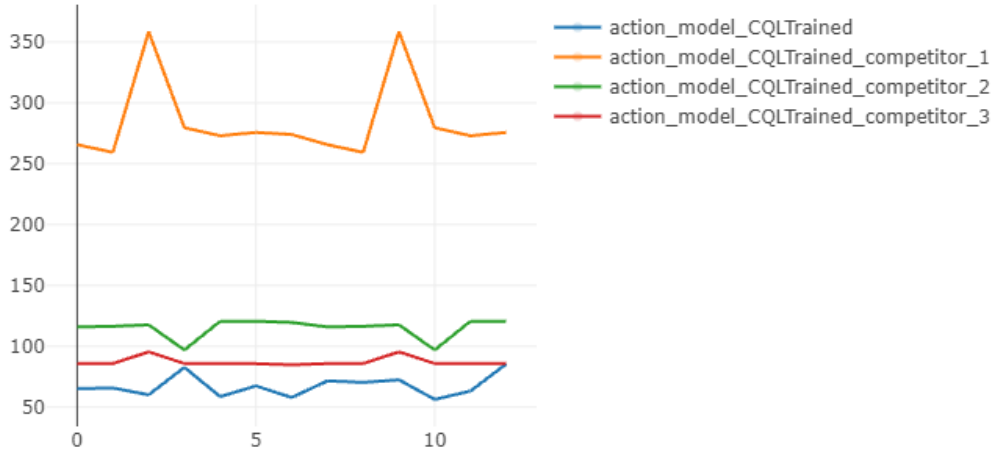


Das Modell performt sehr viel besser als das die Referenz und erzielt relativ konstant gute Resultate. Jedoch ist die Performance immer noch schlechter als die des Fixpreis Modells.

Wenn man die Aktionen vergleicht, wird sichtbar, dass das Modell einen Preis zwischen 56 und 85 Franken setzt. Ebenfalls ist spannend, dass der gesetzte Preis nie tiefer als 50 Franken wird und immer tiefer als 100 Franken ist, das heisst man versucht den Touristen anzupeilen.



Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:



Es wird klar ersichtlich, dass das Modell gelernt hat die Konkurrenz zu unterbieten.

Fazit

Die Resultate sind immer noch sehr gut.

Das Modell hat implizit die untere Schwelle des Touristen gelernt von 50 Franken. Ebenfalls hat er in diesem Fall gelernt die Konkurrenz zu unterbieten.

Jedoch gibt sehr starke Schwankungen in den Aktionen und ein unstabiler Trainingsprozess. Um ein Divergieren des Actor Losses zu minimieren, soll die Learning Rate des Actors reduziert werden.

Actor Learning Rate reduzieren

Datum: 25.04.2022 – 26.04.2022
Ziel: Actor Loss kontrollieren

Beschreibung

Um das Divergieren des Actor Losses zu vermeiden, wird die Learning Rate des Actors reduziert. Damit soll ein weiteres absenken erreicht werden.

Vorgehen

Reduktion der Actor Learning Rate:

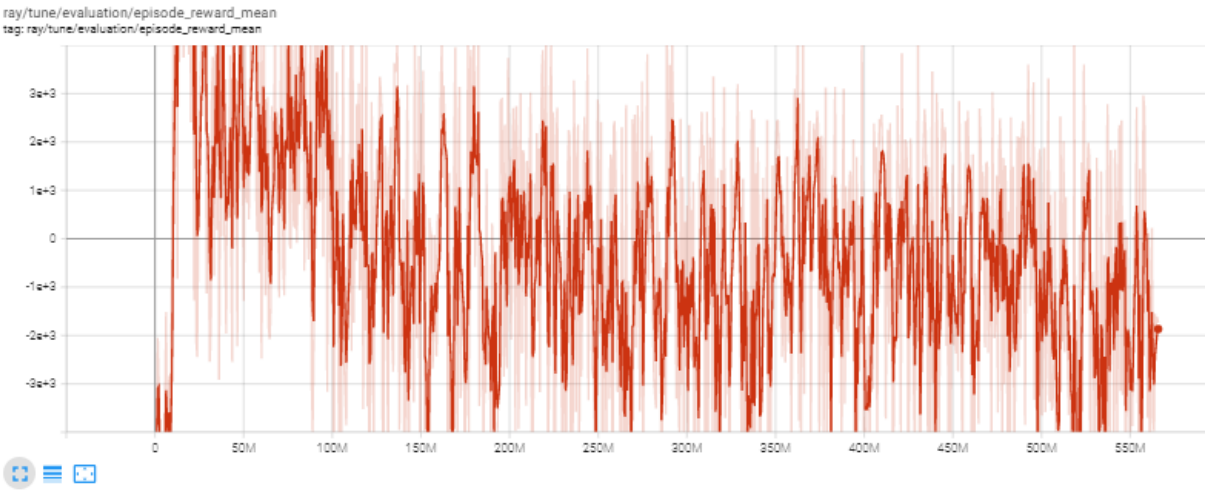
- Vorher: "actor_learning_rate": 3e-4
- Nachher: "actor_learning_rate": 3e-5

Resultate

CQL Training

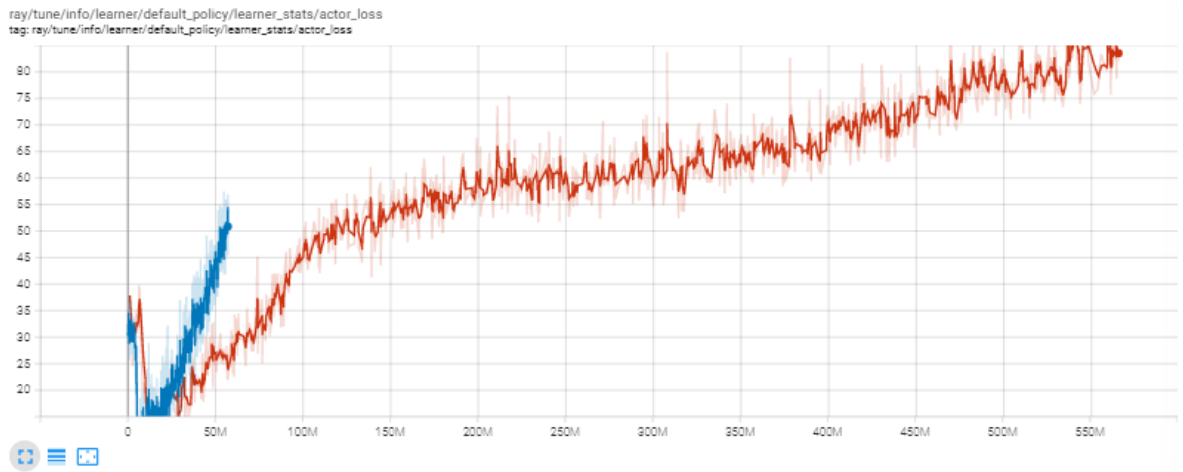
Training über 55000 Iterationen.

Auch hier scheint es den Trend zu geben, dass sich der Reward über die Zeit wieder verschlechtert.

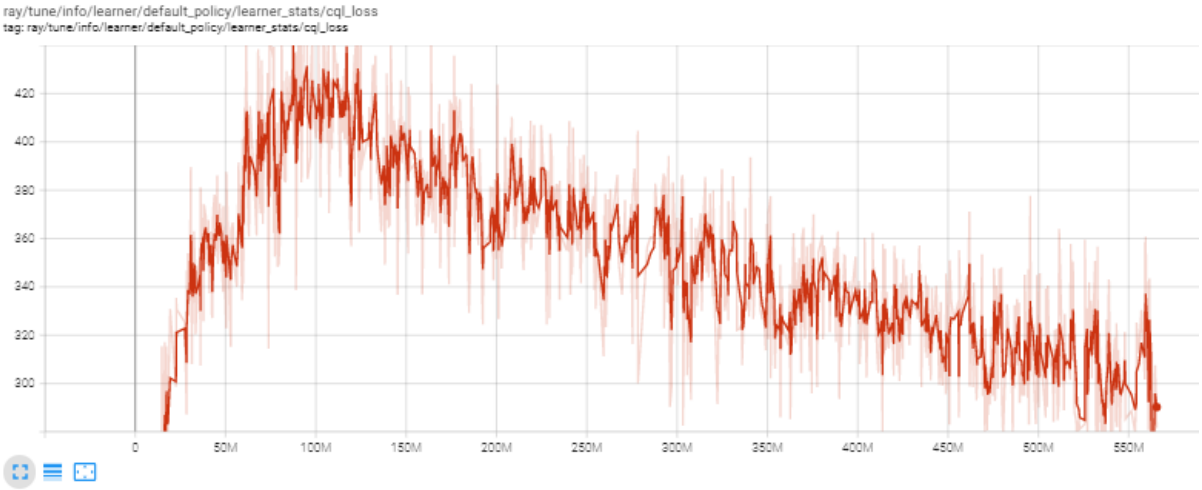


Es sind jedoch auch sehr starke Schwankungen sichtbar in der Performance.

Der Actor Loss steigt zwar langsamer als zuvor (blau), dennoch divergiert dieser nach einer gewissen Zeit.



Auch im CQL Loss wird eine Kehrtwende sichtbar, wo sich die Performance sehr stark verschlechtert.



Rewards

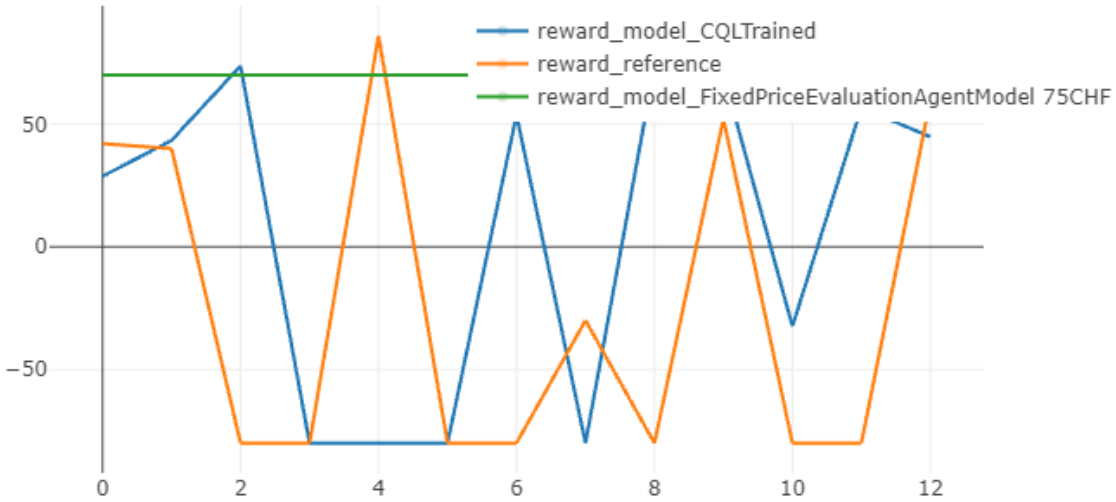
Environment: Version 3
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

Folgende Totale Rewards wurden erzielt:

Referenz:	-310
CQL:	92.34
Fixer Preis 75:	910

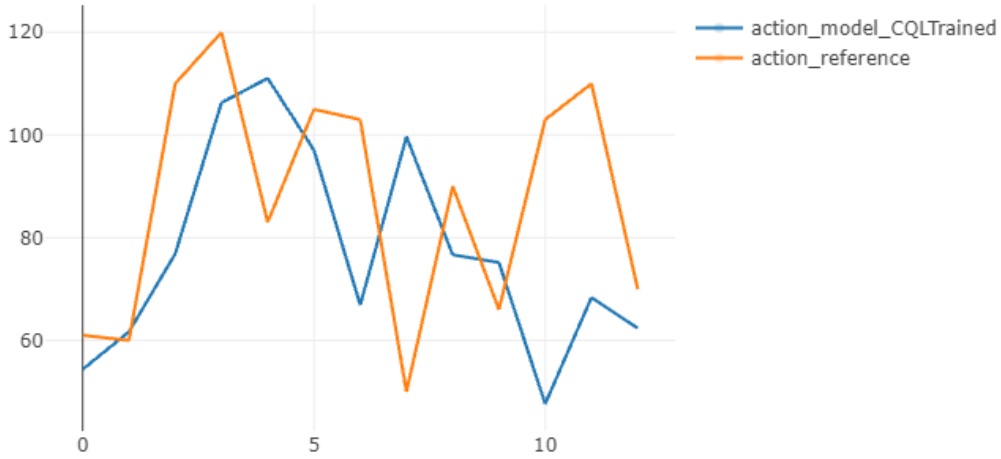
Über die Zeit wurden folgende Rewards erzielt:

Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:



Man sieht, dass die Performance sehr schlecht ist und es sehr viele Zeitpunkte mit negativem Reward gibt.

Zusätzlich ist sichtbar, dass sich die Aktionen sehr stark dem Referenzmodell ähneln.



Jedoch ist die Performance nicht zufrieden stellend.

Fazit

Folglich muss ein Weg gefunden werden, wie sichergestellt werden kann, dass die Performance nach einer gewissen Zeit nicht wieder sinkt.

https://www.reddit.com/r/reinforcementlearning/comments/9zwr0r/why_do_rewards_start_to_drop_after_a_certain/

https://openreview.net/forum?id=B1Yy1BxCZ

Target Network Update Frequenz erhöhen

Datum: 26.04.2022
Ziel: Reduktion der Schwankungen
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/blob/992513c3fa18ad8b7fe3fc3d899137460109f188/src/simulator/runners/cql_runner.py

Beschreibung

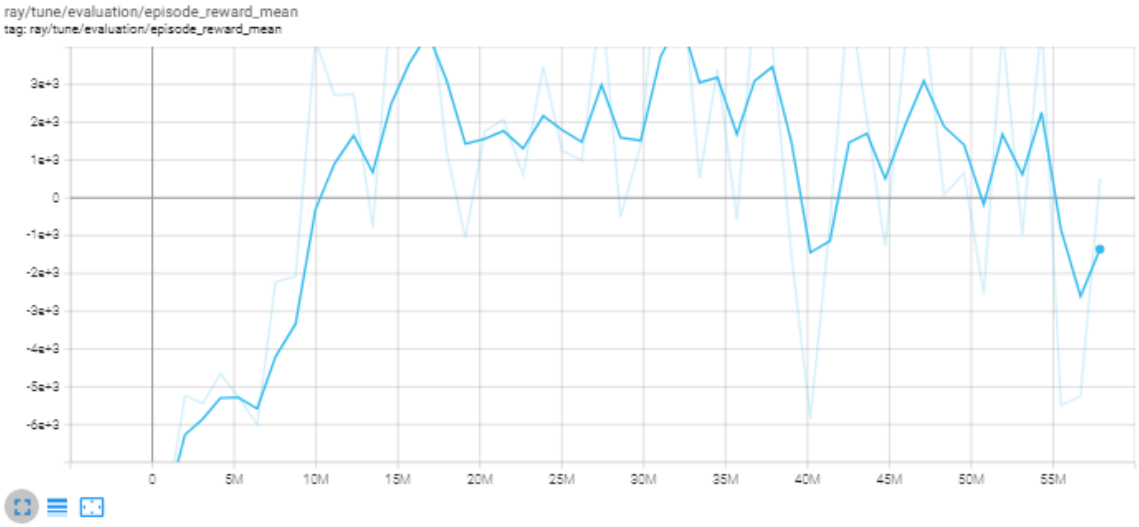
Um das Target Modell nicht immer upzudaten, wird nur alle 10 Iterationen das Target Modell geupdatet.

Vorgehen

- Vorher: config["target_network_update_freq"] = 1
- Nachher: config["target_network_update_freq"] = 10

Resultate

Auch hier scheint es sehr ähnlich zu einer Reduktion des Rewards über die Zeit zu geben.



Fazit

Diese Änderung scheint keine grosse Verbesserung zu geben. Deshalb wird sie rückgängig gemacht!

Batch Size erhöhen

Datum: 26.04.2022
Ziel: Erhöhung der Batch Size

Beschreibung

Weil bereits zuvor eine Erhöhung der Trainings Batch Size signifikant geholfen hat, wird sie erneut erhöht.

Vorgehen

- Vorher: config["train_batch_size"] = 256
- Nachher: config["train_batch_size "] = 512

Resultate

Auch hier sehen wir wieder ein ähnliches Resultat und die Performance beginnt nach einer gewissen Zeit sich zu reduzieren.

Fazit

Es scheint, als ob auch mit dieser Anpassung keine Verbesserung der Performance erreicht werden kann.

Verbesserung der Konkurrenz

Datum: 27.04.2022
Ziel: Realistischere Konkurrenz
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/c76502c99e4820761d8208026b38273ae69466b2

Beschreibung

Für die Wahl der Konkurrenz wird die grösste Similarität anhand der Deskription berechnet. Ebenfalls wird die Beschreibung im State nicht mehr als One-Hot Vektoren gespeichert, sondern es wird als kontinuierlicher Wert genommen.

Vorgehen

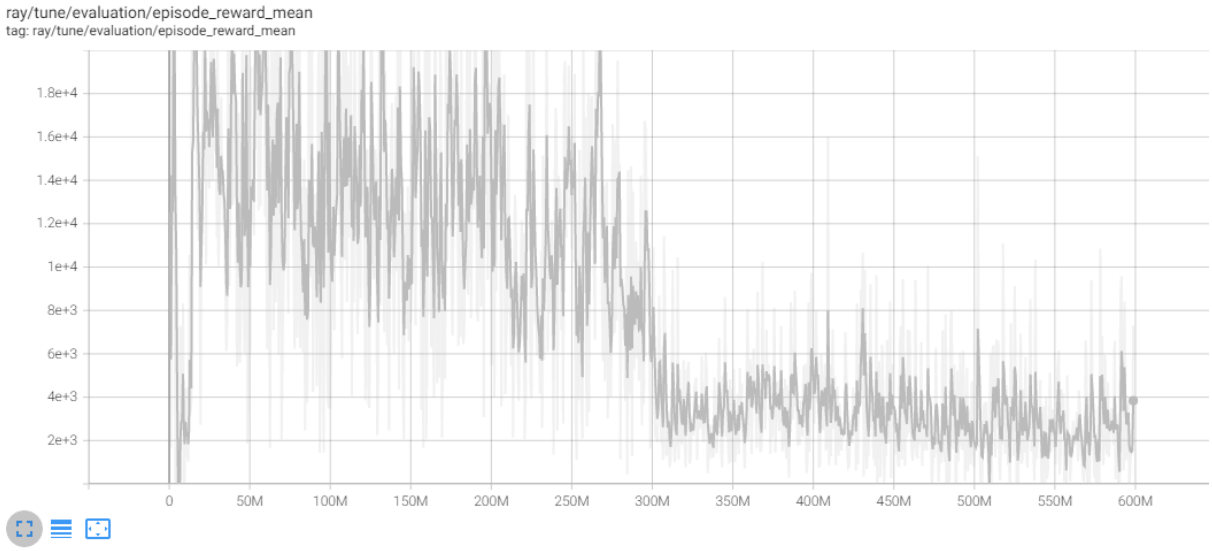
Environment v4

- Competitors haben die grösste Cosinus Similarität aufgrund der Features
- Vorberechnen der ähnlichsten Hotels für jedes Hotel des Reservation Datensets
- Wählen der nächsten 3 Konkurrenten
- Anzeigen der Description als kontinuierliche Zahl
- Training

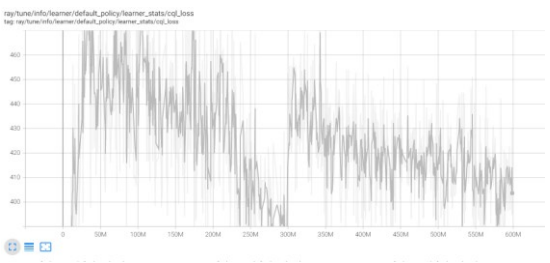
Resultate

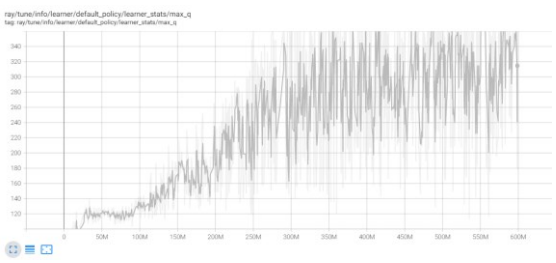
CQL Training

Die Entwicklung des Rewards über 50000 Iterationen mit jeweils 512 Batches sieht es folgendermassen aus:



Die Mean Rewards sind im Training positiv, was zeigt, dass jetzt etwas gelernt wird. Zu Beginn scheint es eine sehr gute Strategie gelernt zu haben, welch dann plötzlich abrupt schlechter wird. Sowohl im CQL Loss wie auch in den maximalen Q Values scheint es eine plötzliche Änderung zu geben:





Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

Rewards

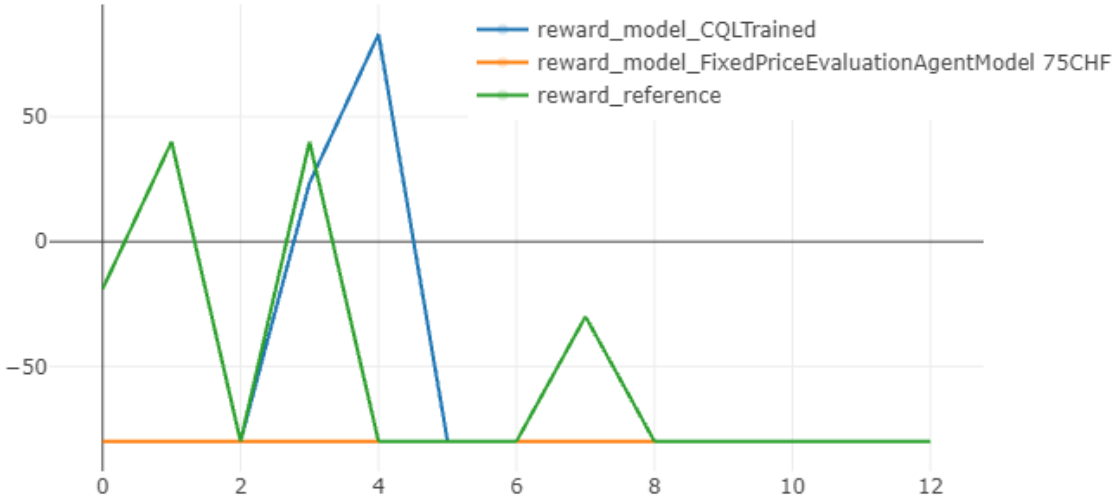
Environment: Version 4
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

Folgende Totale Rewards wurden erzielt:

Referenz: -689
CQL: -773.4
Fixer Preis 75: -1040

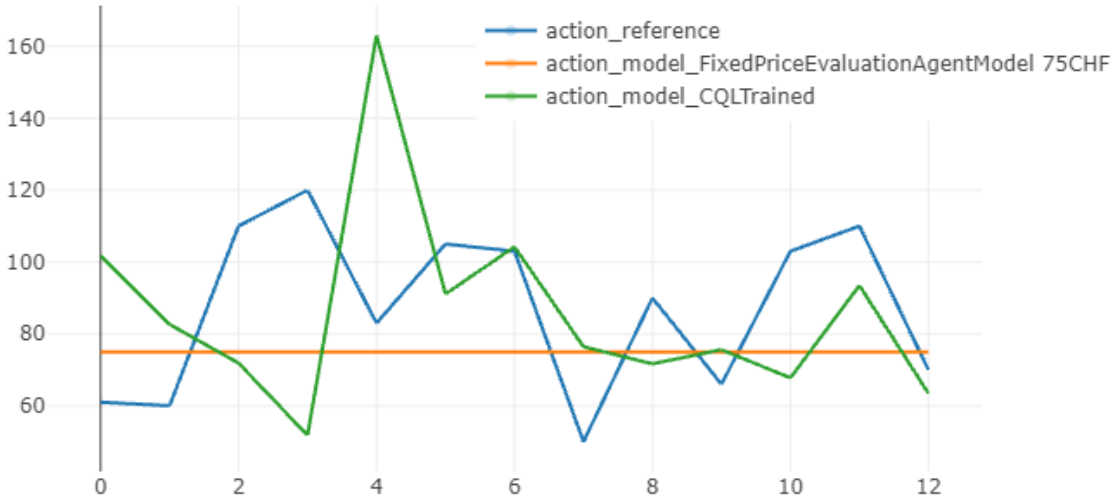
Es ist sehr spannend zu beobachten, dass das Referenzmodell die beste Performance erreicht. Diese ist immer noch unzureichend aber im Vergleich am besten.

Über die Zeit wurden folgende Rewards erzielt:

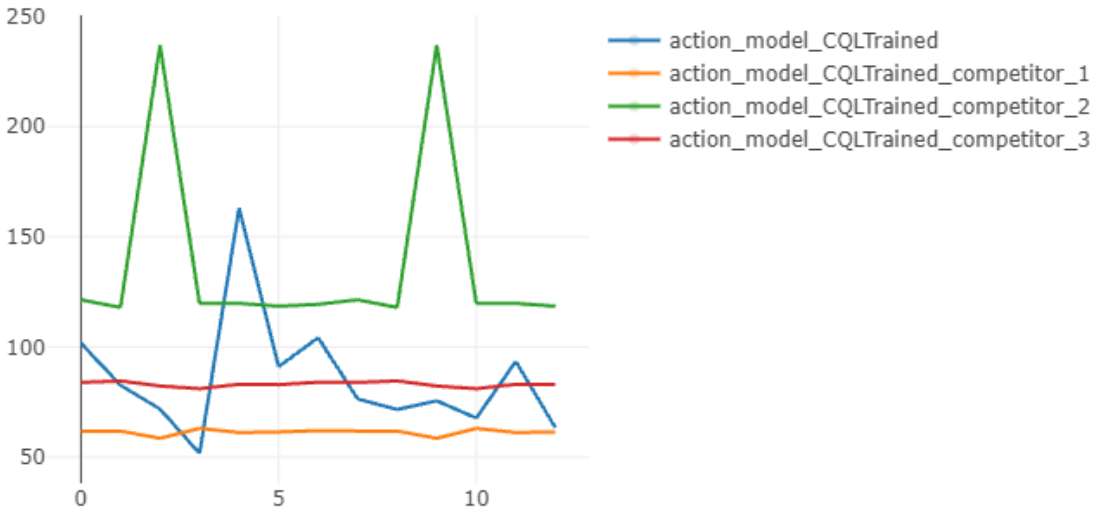


Das Fixpreis Modell scheint bei diesen komplexeren Konkurrenten keinen Reward mehr zu erzielen.

Wenn man die Aktionen vergleicht, wird sichtbar, dass keines der Modelle eine konsequente Strategie hat.



Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:



Es wird klar ersichtlich, dass das Modell es nicht schafft, sich über den Preis gegenüber der Konkurrenz zu positionieren.

Fazit

Folgende zwei Dinge müssen in einem nächsten Schritt verbessert werden:

- Mehr angleichen an den Referenzdatensatz
- Verbessertes Training

Behaviour Cloning Pretraining

Datum: 28.04.2022
Ziel: Um einen schnelleren Start zu ermöglichen, wird ein Pretraining gemacht
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/15903e1fad1adb24c3d63d8461d558a076ca083c

Beschreibung

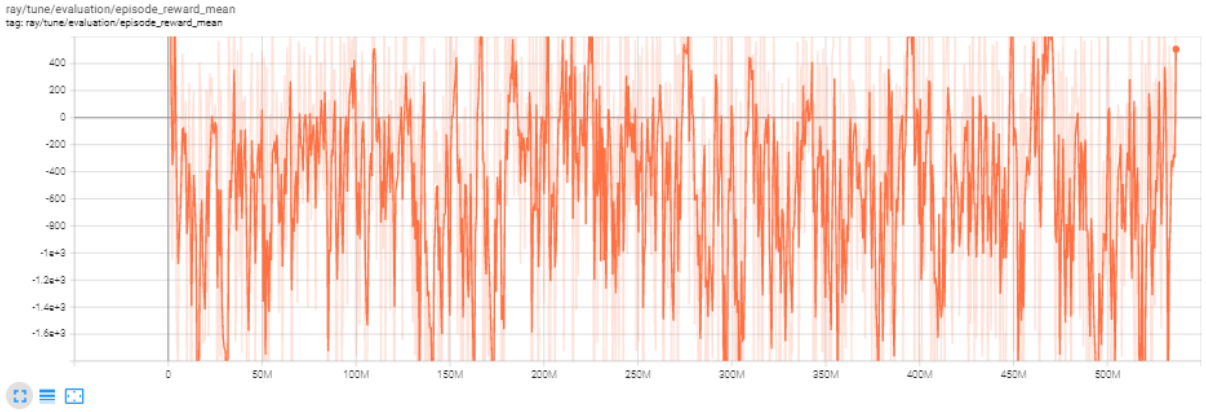
Um ein gewisses Pretraining zu ermöglichen, wird zuerst Behavorial Cloning gemacht.

Vorgehen

- Vorher: config["bc_iters"] = 0
- Nachher: config["bc_iters "] = 20000

Resultate

Es scheint hier nie ein gutes Resultat bei den Rewards zu geben, das heisst das Behaviour Cloning ist eher negativ:



Fazit

Diese Änderung scheint eine Verschlechterung zu geben. Dies wird folglich zurückgesetzt.

Längeres Training inkl. höhere Batch Size und tieferer Learning Rate

Datum: 29.04.2022

Ziel: Kein Divergieren der Performance mehr

File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/87ae65cb83c903c61e79392d0312354414655632

Beschreibung

Um einen Ausbruch zu minimieren wird die Learning Rate gesenkt und die Batch Size erhöht.

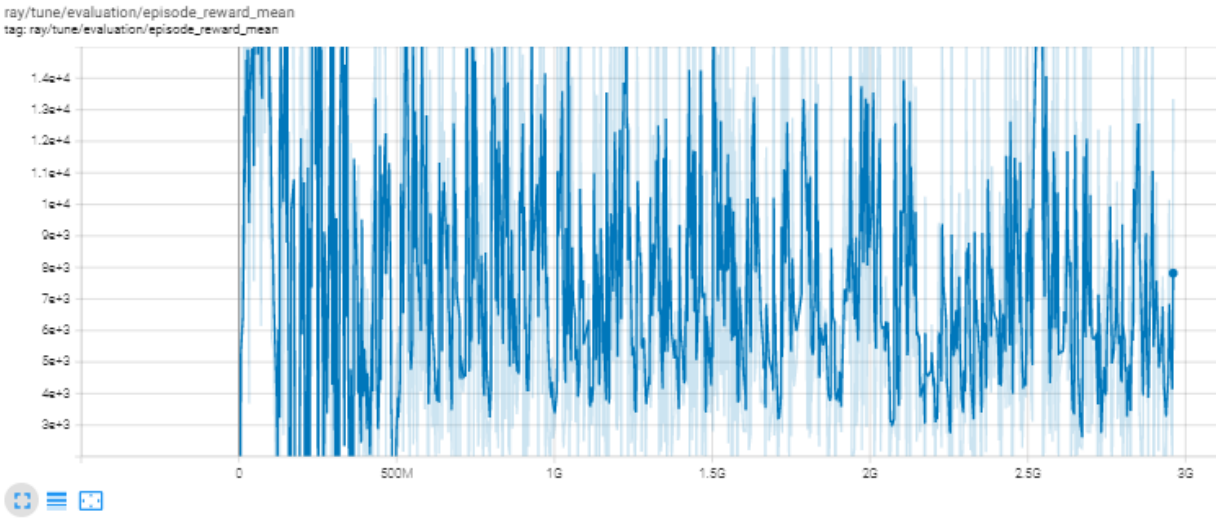
Vorgehen

- Vorher: “actor_learning_rate”: 3e-5
- Nachher: “actor_learning_rate”: 3e-4
- Vorher: "critic_learning_rate": 3e-4
- Nachher: "critic_learning_rate": 1e-5
- Vorher: "entropy_learning_rate": 3e-4
- Nachher: "entropy_learning_rate": 1e-5
- Vorher: config["train_batch_size"] = 512
- Nachher: config["train_batch_size"] = 1024

Resultate

CQL Training

Die Entwicklung des Rewards über 235000 Iterationen mit jeweils 1024 Batches sieht es folgendermassen aus:



Jetzt scheint es zu funktionieren, dass die Performance gut bleibt.

Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

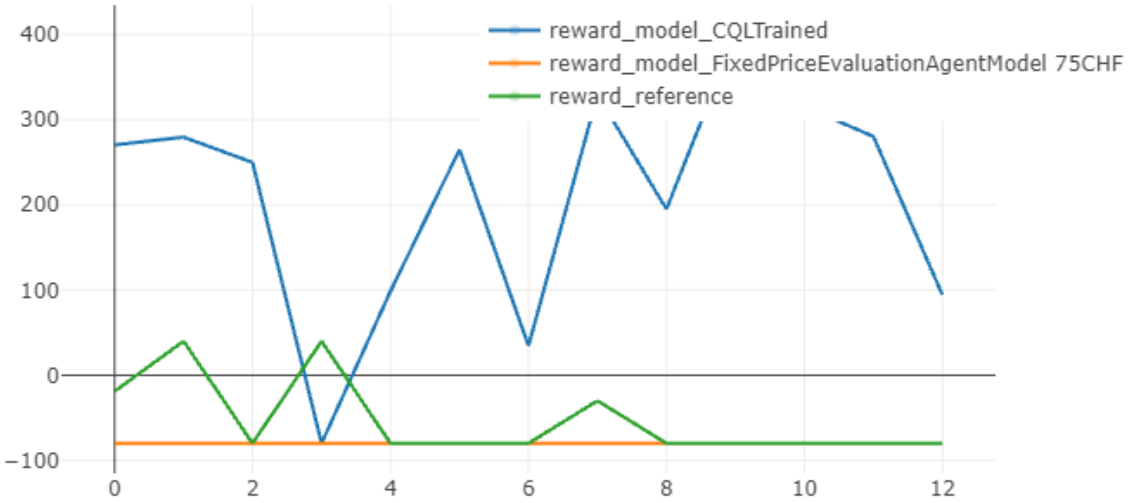
Rewards

Environment: Version 4
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

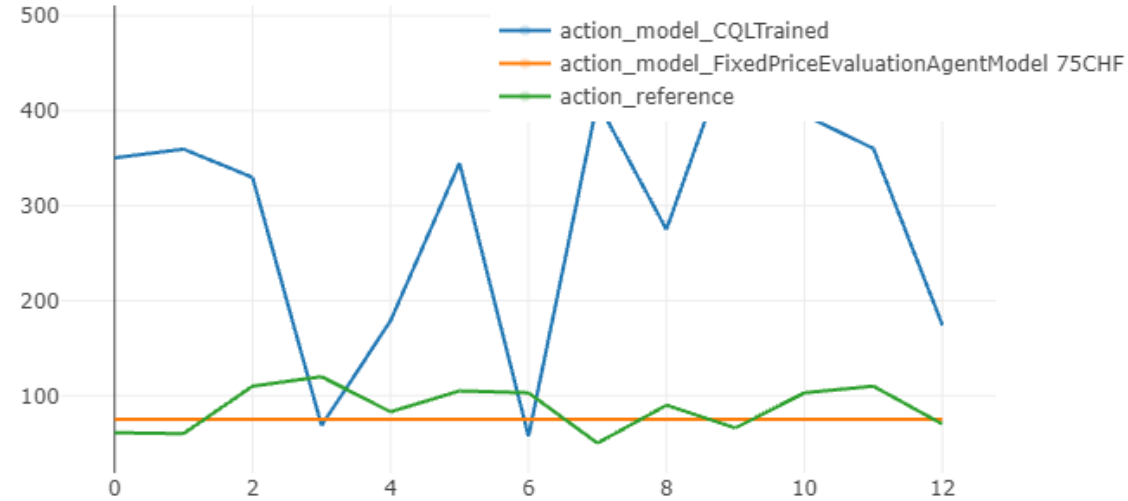
Folgende Totale Rewards wurden erzielt:

Referenz: -689
CQL: 2731.1
Fixer Preis 75: -1040

Über die Zeit wurden folgende Rewards erzielt:

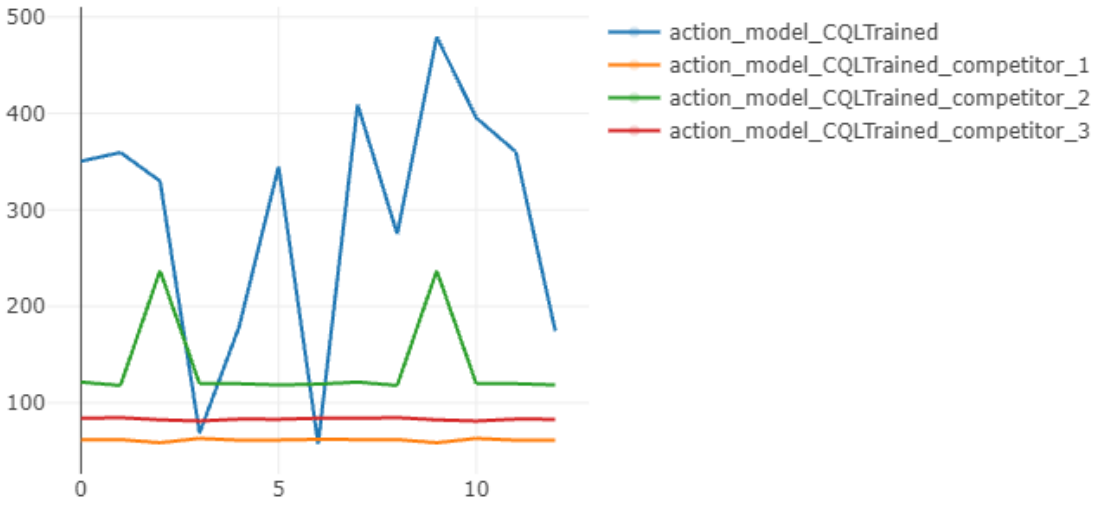


Das CQL Model hat ein sehr guter Reward erzielt.



Wenn man die Aktionen vergleicht, wird sichtbar, dass das CQL Modell gelernt hat, einen sehr hohen Preis zu setzen.

Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:



Es wird klar ersichtlich, dass das Modell es schafft, sich über den Preis gegenüber der Konkurrenz zu positionieren. Zum grössten Teil setzt es den Preis höher und zum Teil auch tiefer.

Fazit

Dies zeigt sehr schön, dass das Modell gelernt hat den Business Kunden anzusprechen. Jedoch muss dies mit einem neuen Nachfragemodell validiert werden.

Evaluation Environment v5

Datum: 03.05.2022 – 04.05.2022
Ziel: Evaluation des Env v5
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/e7c338f5d15c73a062f3ef8ea8ca16951b037a5b

Beschreibung

Im folgenden Versuch soll das Environment v5 mit der verbesserten Nachfrage evaluiert und getestet werden.

Vorgehen

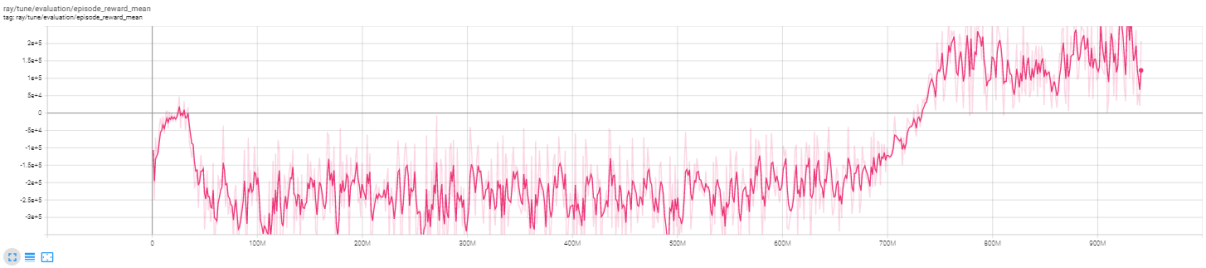
Generieren der Daten für 500 Iterationen bei einer Hotelkapazität von 150 Hotels und 76900 Simulationsschritten.

Resultate

Wenn dieses Modell auf 500 Iterationen simuliert wird bei einer maximalen Kapazität von 150 Hotels, dann gibt das folgende Resultate nach 76900 Simulationsschritten:

CQL Training

Die Entwicklung des Rewards über 76900 Iterationen mit jeweils 1024 Batches sieht es folgendermassen aus:



Jetzt scheint es zu funktionieren, dass die Performance gut bleibt.

Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

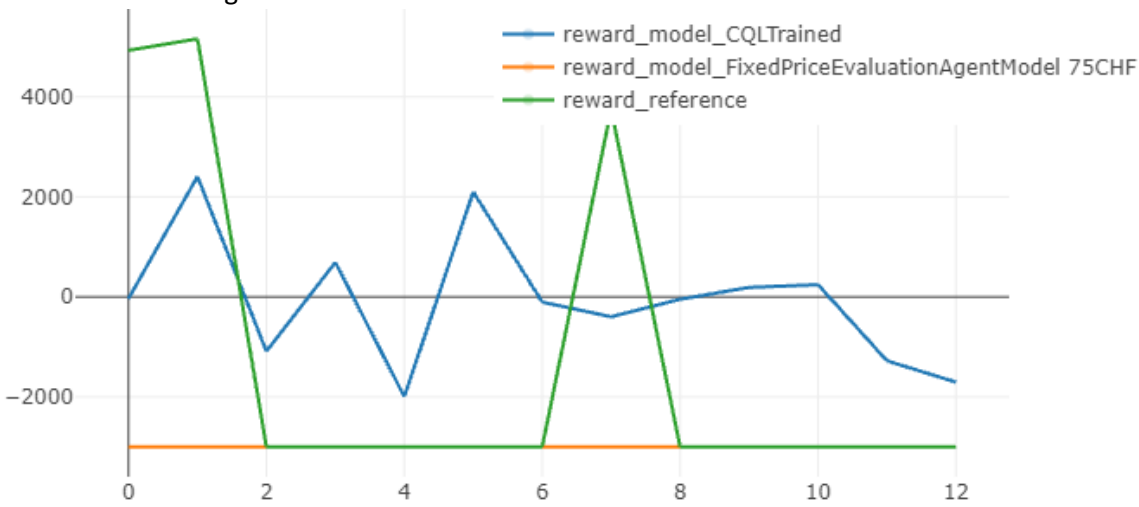
Rewards

Environment: Version 5
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

Folgende Totale Rewards wurden erzielt:

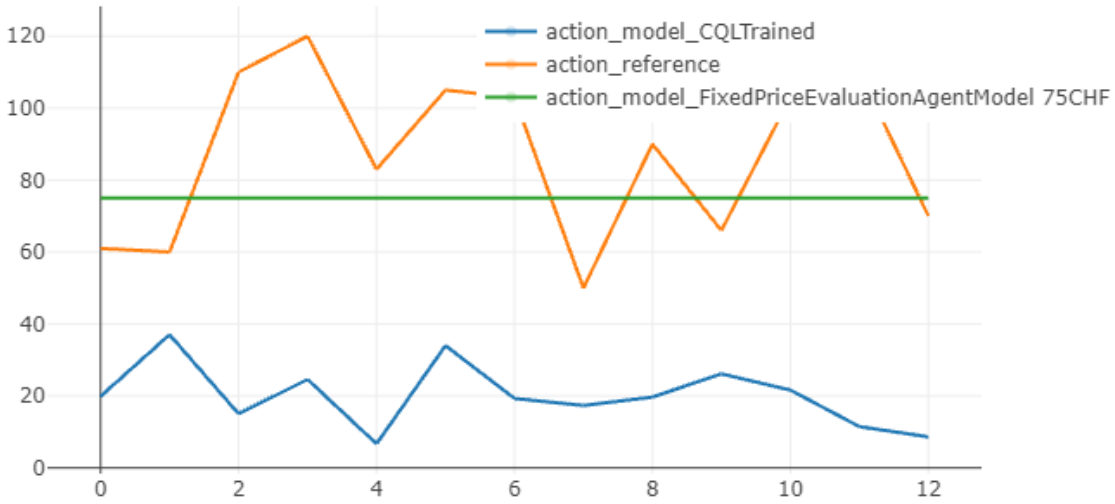
Referenz: -16160
CQL: -1024.7
Fixer Preis 75: -39000

Über die Zeit wurden folgende Rewards erzielt:



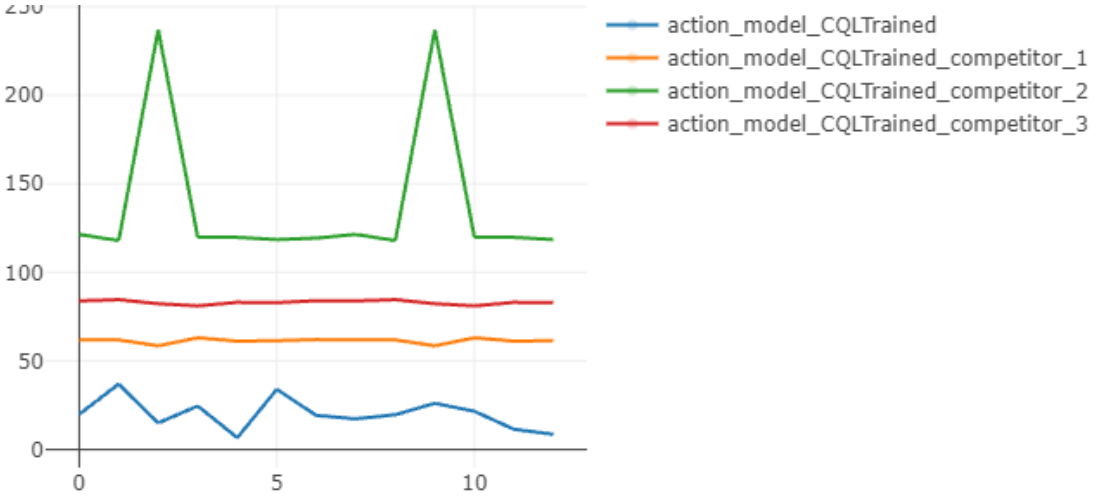
Das CQL Model hat im Verhältnis den besten Reward erzielt. Jedoch auch nur sehr knapp immer positiv.

Wenn man die Aktionen vergleicht, wird sichtbar, dass das CQL Modell gelernt hat, einen sehr tiefen Preis zu setzen.



Dies ist dem Resultat geschuldet, dass jetzt immer der tiefste Preis die Kunden erhält.

Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:



Es wird klar ersichtlich, dass das Modell es schafft, sich über den Preis gegenüber der Konkurrenz zu positionieren. Zum grössten Teil setzt es den Preis sehr tief.

Fazit

Dies zeigt sehr schön, dass das Modell gelernt hat einen tiefen Preis zu setzen. Jedoch ist der Preis, welcher gesetzt wird, eher tief. Dennoch sollte es überprüft werden, ob bei evt längerem Training bessere Resultate erzielt werden. Ebenfalls sollte die andere Konkurrenzsituation modelliert werden.

Verbessertes Kundenmodell mit Kundengruppen

Datum: 03.05.2022 – 04.05.2022
Ziel: Segmentierung der Kunden
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/bt-rl-dynamic-pricing-dataset/-/commit/0e46be5e2ebb84251dc6c8cadc01f7570866572b

Beschreibung

Um eine spannendere Nachfragekurve zu erzeugen, wurden zwei unterschiedliche Kundegruppen gesucht. Die Kunden, die nur unter der Woche dort sind (eher Business) und solche welche über das Wochenende bleiben (Touristen).

Vorgehen

Aufteilung von allen Buchungen anhand des folgenden Kriteriums:

- Gruppe Business: Keine der Buchungstage am Wochenende
- Gruppe Tourist: Mind. Einer der Buchungstage am Wochenende

Anschliessend soll erneut pro Tag und Sommer/Winter ein Modell erzeugt werden und verglichen werden, ob es signifikant andere Resultate bezüglich der Preissensitivität gibt.

Resultate

Dabei wurden folgende Resultate erzielt:

Gruppe	**Modell**	**Mean MSE (Std)**	**Mean R2 (Std)**
Business	Linear	880.27 (426.86)	0.156 (0.079)
Business	Quadratisch	858.339 (415.759)	0.175 (0.086)
Tourist	Linear	5240.73 (2417.65)	0.110 (0.020)
Tourist	Quadratisch	5156.21 (2424.15)	0.127 (0.030)

Erneut wird ersichtlich, dass das lineare Modell minimal schlechtere Resultate erzielt. Jedoch ist dieser Unterschied minimal. Deshalb wird erneut das lineare Modell gewählt.

Wenn man die Preiselastizitäten Sommer, *Nicht Sommer* der Nachfrage betrachtet, erhält man folgende Werte:

	Montag	**Dienstag**	**Mittwoch**	**Donnerstag**	**Freitag**	**Samstag**	**Sonntag**
Business							
50 -> 60 CHF	-0.285 *-0.325*	-0.274 *-0.314*	-0.284 *-0.338*	-0.226 *-0.353*	-0.086 *-0.062*	-	-
100 -> 120 CHF	-0.795 *-0.962*	-0.753 *-0.915*	-0.792 *-1.019*	-0.584 *-1.090*	-0.188 *-0.132*	-	-
150 -> 180 CHF	-1.981 *-2.782*	-1.812 *-2.528*	-1.965 *-3.116*	-1.238 *-3.596*	-0.310 *-0.212*	-	-
Tourist							
50 -> 60 CHF	-0.288 *-0.228*	-0.299 *-0.243*	-0.312 *-0.261*	-0.282 *-0.233*	-0.200 *-0.168*	-0.192 *-0.160*	-0.247 *-0.192*
100 -> 120 CHF	-0.810 *-0.591*	-0.854 *-0.641*	-0.908 *-0.707*	-0.786 *-0.609*	-0.499 *-0.405*	-0.476 *-0.380*	-0.657 *-0.475*
150 -> 180 CHF	-2.041 *-1.257*	-2.235 *-1.414*	-2.495 *-1.639*	-1.943 *-1.313*	-0.998 *-0.761*	-0.938 *-0.703*	-1.469 *-0.935*

Es ist sehr interessant zu sehen, dass der Businesskunde ein umgekehrtes Verhalten zum Touristen zeigt. So hat der Business Kunde ausserhalb des Sommers eine höhere Preiselastizität als im Sommer selbst. Am Freitag ist die Elastizität sehr gering, da die wenigen Businesskunden, die am Freitag dort sind, sowieso gehen.

Hingegen die Touristen haben im Sommer eine höhere Preiselastizität.

Fazit

Es sind signifikant andere Verhalten kann ein spannendes Resultat ermöglichen. In einem weiteren Schritt soll dieses Modell evaluiert werden.

Evaluation Environment v6

Datum: 04.05.2022 – 05.05.2022
Ziel: Evaluation des Env v6
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/ba0347728c2570ae336317873cb555b9623ba921

Beschreibung

Im folgenden Versuch soll das Environment v6 mit der verbesserten Nachfrage (Business, Tourist) evaluiert und getestet werden.

Vorgehen

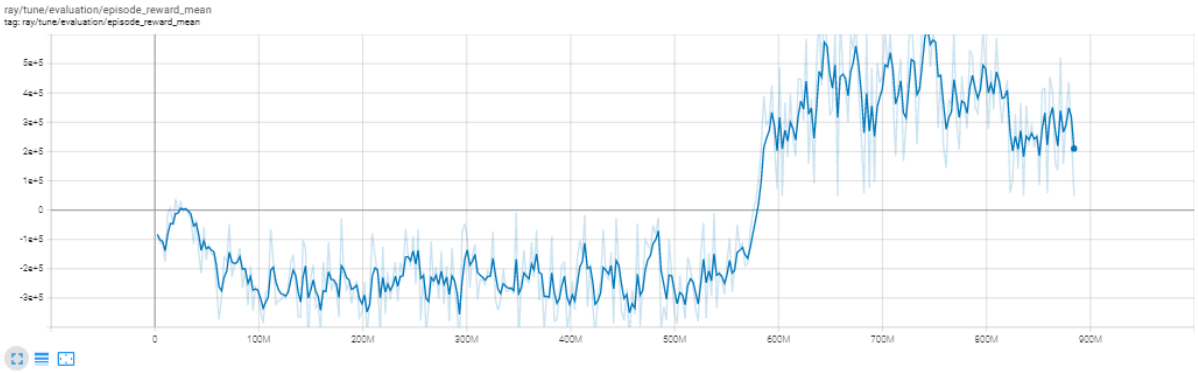
Generieren der Daten für 500 Iterationen bei einer Hotelkapazität von 150 Hotels und 70600 Simulationsschritten.

Resultate

Wenn dieses Modell auf 500 Iterationen simuliert wird bei einer maximalen Kapazität von 150 Hotels, dann gibt das folgende Resultate nach 70600 Simulationsschritten:

CQL Training

Die Entwicklung des Rewards über 70600 Iterationen mit jeweils 1024 Batches sieht es folgendermassen aus:



Das Modell scheint nach einer gewissen Zeit signifikant was gelernt zu haben.

Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

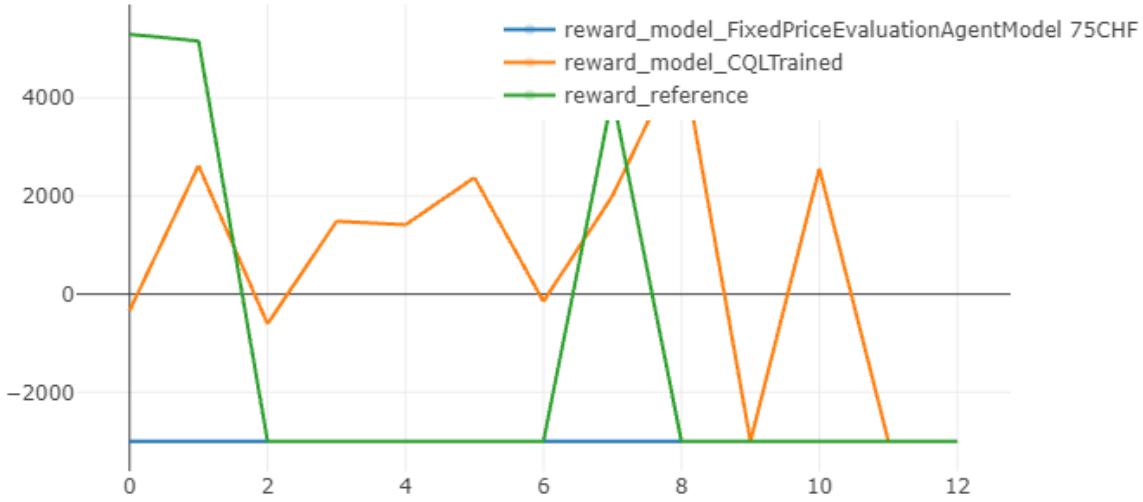
Rewards

Environment: Version 6
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

Folgende Totale Rewards wurden erzielt:

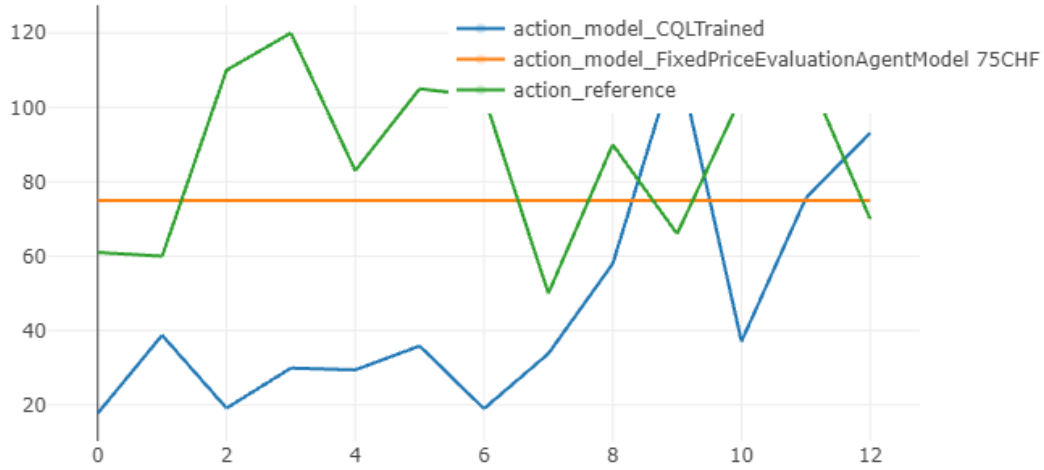
Referenz: -15494
CQL: 7307.6
Fixer Preis 75: -39000

Über die Zeit wurden folgende Rewards erzielt:



Das CQL Model hat im Verhältnis den besten Reward erzielt. Zum Teil sehr stark positiv.

Wenn man die Aktionen vergleicht, wird sichtbar, dass das CQL Modell gelernt hat, einen sehr tiefen Preis zu setzen.



Dies ist dem Resultat geschuldet, dass jetzt immer der tiefste Preis die Kunden erhält.

Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:

Es wird klar ersichtlich, dass das Modell es schafft, sich über den Preis gegenüber der Konkurrenz zu positionieren. Zum grössten Teil setzt es den Preis sehr tief.

Fazit

Dies zeigt sehr schön, dass das Modell gelernt hat einen tiefen Preis zu setzen. Jedoch ist der Preis, welcher gesetzt wird, eher tief. Ebenfalls ist das Verhalten der Konkurrenz sehr klar sichtbar, deshalb wird in einem Folgeschritt auf das Sampling gewechselt.

Samplen der Konkurrenzpreise

Datum: 04.05.2022
Ziel: Konkurrenzpreise nicht fix zurückgeben

Beschreibung

Damit die Preise der Konkurrenz für einen bestimmten Tag nicht fix sind, werden die erhaltenen Preise der Konkurrenz durch eine Poisson Verteilung mit Mittelwert = Modelpreis gesampelt.

Diese gesampelten Preise werden für die Datengenerierung und das Training verwendet. Jedoch werden die Preise nicht gesampelt bei der Evaluation, damit eine Vergleichbarkeit der Modelle gewährleistet ist.

Vorgehen

- Parameter definieren, damit es gesampelt werden kann.
- Sampling in der Datengenerierung
- Sampling beim Training
- Kein Sampling in der Validierung

Resultate

- Dies macht nur sehr bedingt Sinn, da es kein stochastischer Prozess ist.

Fazit

Dies kann wieder ausgeschaltet werden.

Anpassung gemäss Sitzung Daniel Pfäffli env v8

Datum: 06.05.2022

Ziel: Samplen aus Demand und nicht Preise

File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/0512db2e6ca1e3ee252d30eedaf07318c1f462af

Beschreibung

Um die Feedbacks einzuarbeiten, wurden folgende Anpassungen gemacht. Dadurch entsteht das Environment v8.

Vorgehen

- Kein Sampeln der Konkurrenzpreise
- Bestimmung der Nachfrage durch den Mean der Preise (Konkurrenz + Agent) -> So hat Agent auch Einfluss
- Samplen mit Poisson bei der Nachfrage
- Bestimmen der Hotelzimmer des Agenten durch Max aus den Daten

```
train_demand.groupby("hotel_id")["guest_count"].max()
✓ 0.9s
hotel_id
1098    351.0
1099    349.0
1100    349.0
1101    347.0
1102    328.0
1103    341.0
1527    328.0
1706     52.0
1708    107.0
1709    345.0
1795     46.0
Name: guest_count, dtype: float64
```

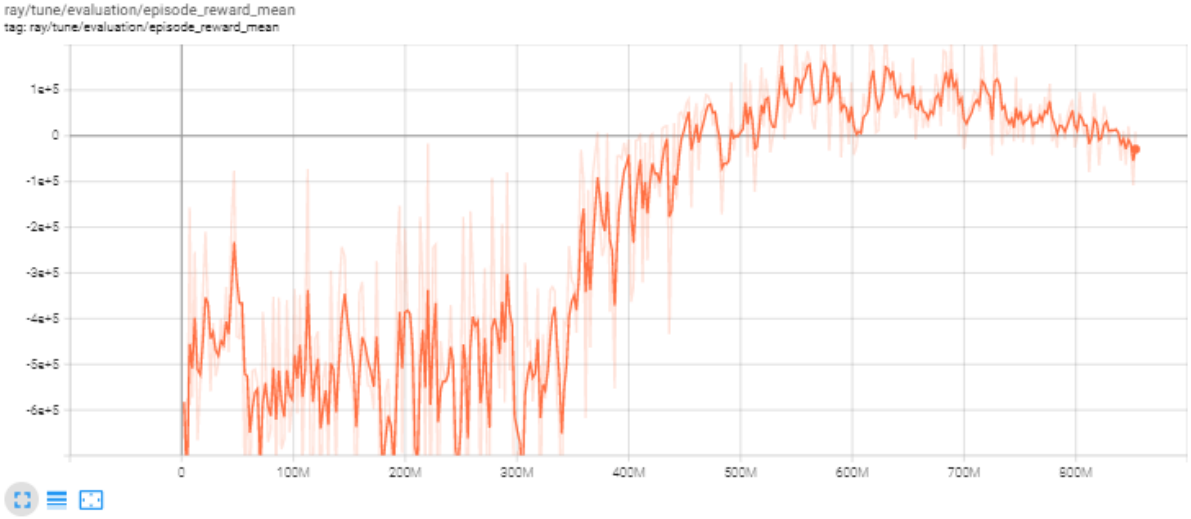
- Konkurrenzgrösse ??? -> Gleich wie Agent?
  - 200
- Welches Hotel erhalten wir:
  - Tiefster Preis, dann füllen, dann zweittiefster Preis, …

Resultate

Wenn dieses Modell auf 750 Iterationen simuliert wird bei einer maximalen Kapazität von 150 Hotels, dann gibt das folgende Resultate nach 76700 Simulationsschritten:

CQL Training

Die Entwicklung des Rewards über 76700 Iterationen mit jeweils 1024 Batches sieht es folgendermassen aus:



Das Modell scheint nach einer gewissen Zeit signifikant was gelernt zu haben. Auch wenn die Werte eher wieder absinken.

Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

Rewards

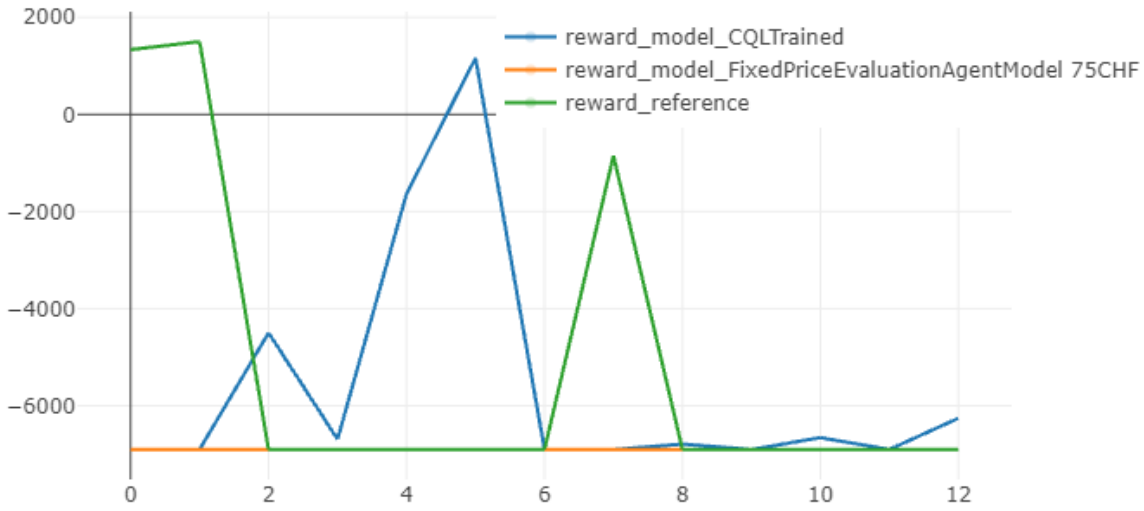
Environment: Version 8
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

In diesem Fall sind die Resultate nicht mehr 100 % reproduzierbar und vergleichbar, da die Nachfrage gesampelt wird.

Folgende Totale Rewards wurden erzielt:

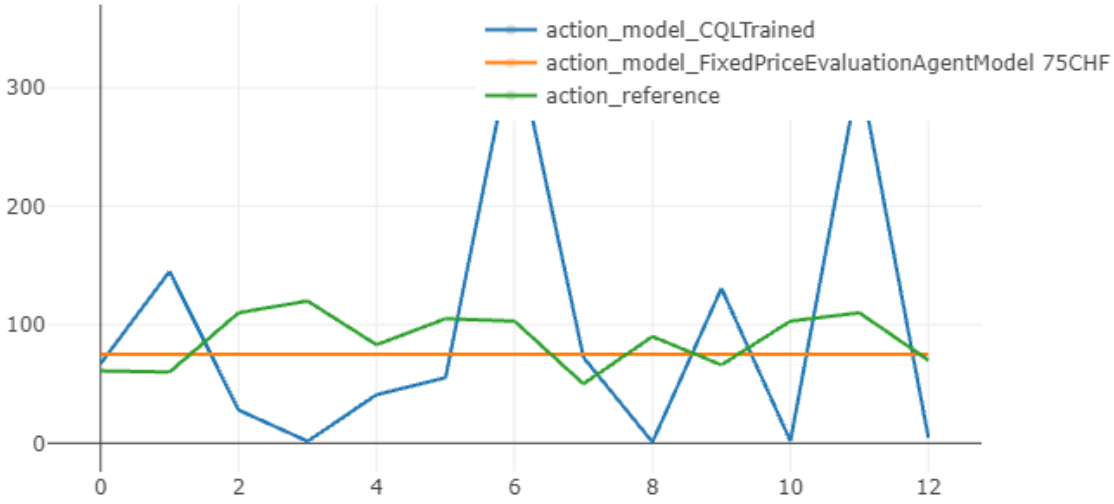
Referenz: -67015
CQL: -72757.9
Fixer Preis 75: -89700

Über die Zeit wurden folgende Rewards erzielt:

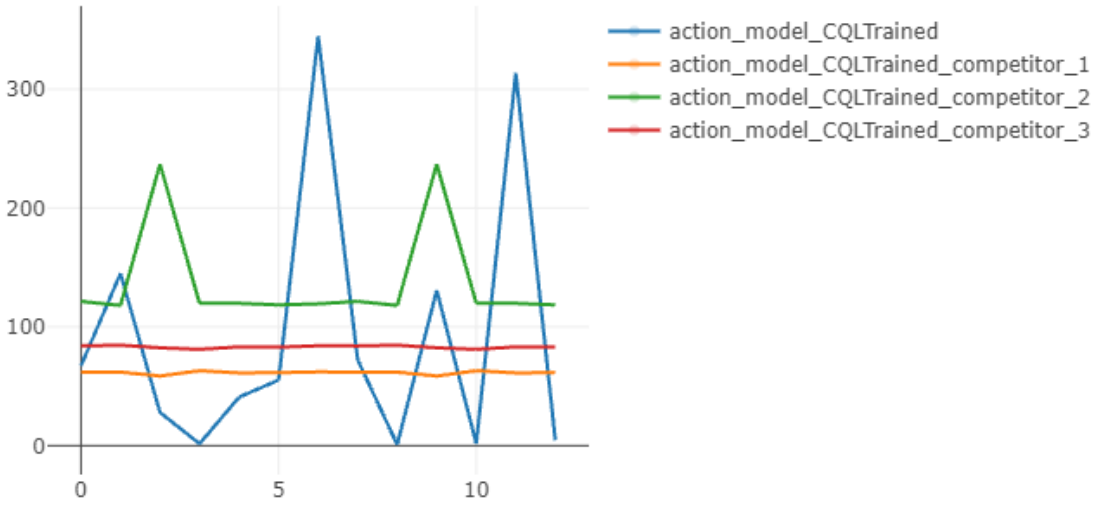


Das CQL Model hat keine wirkliche Strategie gelernt.

Wenn man die Aktionen vergleicht, wird sichtbar, dass das CQL Modell gelernt hat, einen sehr schwankenden Preis zu setzen.



Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:



Es wird klar ersichtlich, dass das Modell nicht schafft eine sinnvolle Preisstrategie zu setzen, entweder wählt es einen sehr tiefen Preis oder einen sehr hohen!

Fazit

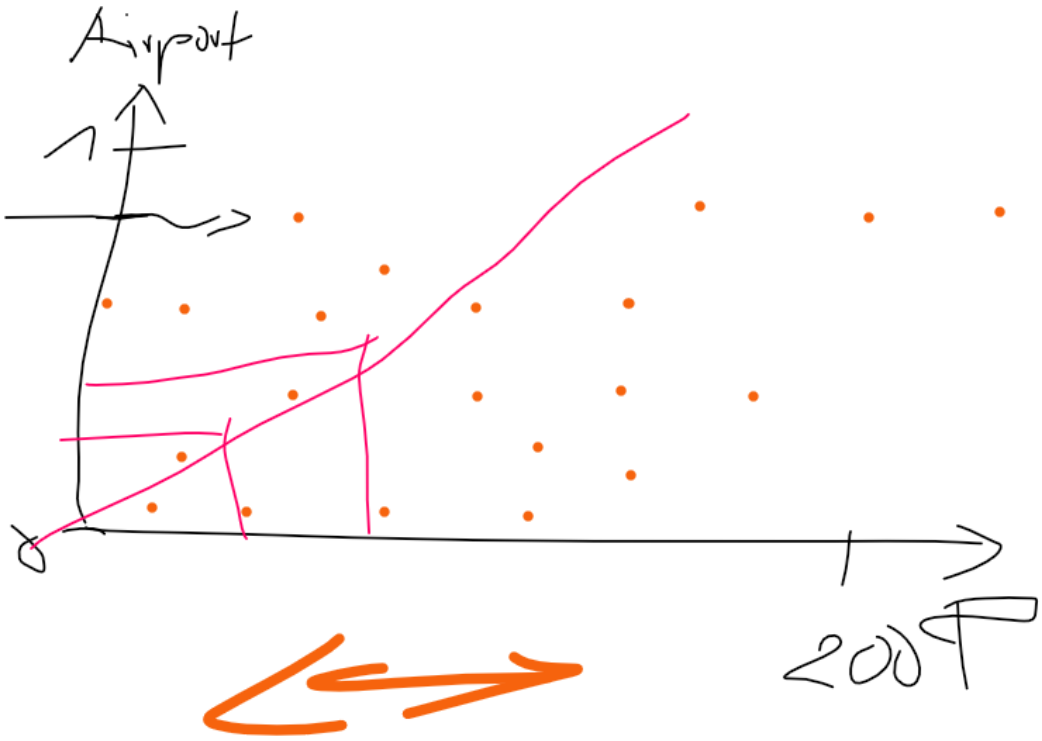
Das Modell hat nicht wirklich sinnvolles Verhalten gelernt. Es scheint nicht mit der schwankenden Nachfrage umgehen zu können.

Diskussion mit Tobias Baltensperger
Datum: 06.05.2022
Ziel: Nachfrage besser darstellen

Beschreibung
Damit die Nachfrage und die Kundengruppen besser modelliert werden können, wurde eine ausführliche Diskussion geführt. Einzelne Notizen sind unten ersichtlich:

Vorgehen
Diskussionspunkte:

Modellierung der Hotelfeatures mit linearer Regression



Durchschnitt in Pricerange nehmen

Gesamtnachfrage: 4 * In die Kurve

4 gibt es Gruppen

Wie bringt man die Abhängigkeit!

Durchschnittlicher Umsatz pro Wochentag vorgeben

Umsatz ist Konstant?

Mail von Tobias Baltensperger 06.05.2022:

ich habe nichts konkretes zu multi product inverse demand curves gefunden, aber ich denke auch, dass wir uns die Sache zu kompliziert machen. Was denkst du zu folgendem Vorschlag:

Kategorien wie gehabt: Wochentag, Business / Tourist, Sommer / nicht-Sommer

Dann berechnest du für jede Kategorie den Umsatz (inkl Konkurrenz).

Anschliessend samplest du für jede Kategorie Kunden = Kundenfeature-Kombinationen, wobei die Kundenfeature-Kombinationen auf effektiven Buchungen basieren. Ein Kunde nach dem anderen wählt ein Hotel basierend auf seiner Kostenfunktion, die du aus dem bestehenden Kundenmodell übernimmst. Du samplest so lange bis der durchschnittliche Umsatz für diese Kategorie erreicht wurde.

Die Preis- sowie die Featureelastizität hast du bereits in deiner bestehenden Kostenfunktion drin (du gibst sie implizit vor durch die Struktur und Parametrisierung deiner Kostenfunktion); du musst sie also nicht nochmals explizit berechnen. Später könntest du das dann allerdings machen und die Elastizitäten explizit in der Kundenkostenfunktion einbauen. Da kommt auch die Crosselasticity ins Spiel: auch diese hast du bereits implizit in deinem Kundenmodell drin via deiner bestehenden Kostenfunktion.

Was denkst du dazu?

Falls du denkst das funktioniert: implementier doch dieses Ansatz Mal und schau was raus kommt. Später kannst du dann das samplen immer noch komplexer gestalten, z.B. in dem du nicht mehr nur bestehende Kundenfeature-Kombinationen samplest, sondern die Kunden Features als multivariate Verteilungen modellierst und daraus samplest. Das wird nicht ganz einfach: Die Korrelation dieser multivariaten Verteilungen enthält implizit die Crosselasticity, dh du müsstest Verteilungen finden mit einer Korrelation, die mit der Crosselasticity in den Daten übereinstimmt... Aber schauen wir wenn wir so weit sind, evt reicht ein einfaches Sampling aus.

Fazit

Wiedereinführung Nutzerpräferenzen env v9

Datum: 09.05.2022
Ziel: Wiedereinführung der Nutzer
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/db6d4d42d85d56d0ad74cc6a819a59ada92f9d75

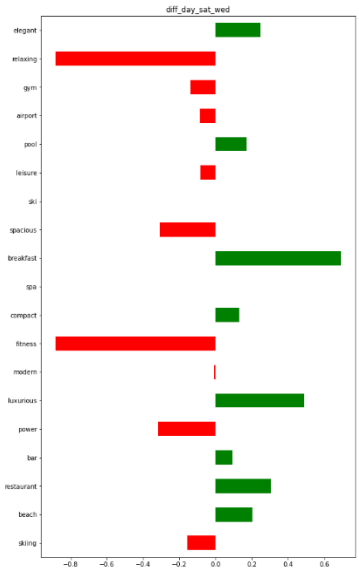
Beschreibung

Um eine intelligentere Entscheidung zu erhalten, werden die Nutzerpräferenzen wieder eingeführt.
Env v9

Vorgehen

- Zwei Kundengruppen
  - Freizeit (Tourist)
  - Business
- 0.5 * Preisrang + 0.5 * Präferenz des Kunden (Präferenz mal Eigenschaft Hotel)
- + = Präferenz Business, - = Präferenz Leisure

```
- skiing       -0.157260
- beach         0.202552
- restaurant    0.307710
- bar           0.093022
- power        -0.320148
- luxurious     0.490483
- modern       -0.006262
- fitness      -0.883408
- compact       0.130124
- spa           0.000000
- breakfast     0.691794
- spacious     -0.309221
- ski           0.000000
- leisure      -0.084453
- pool          0.170209
- airport      -0.087877
- gym          -0.139031
- relaxing     -0.884178
- elegant       0.246283
```

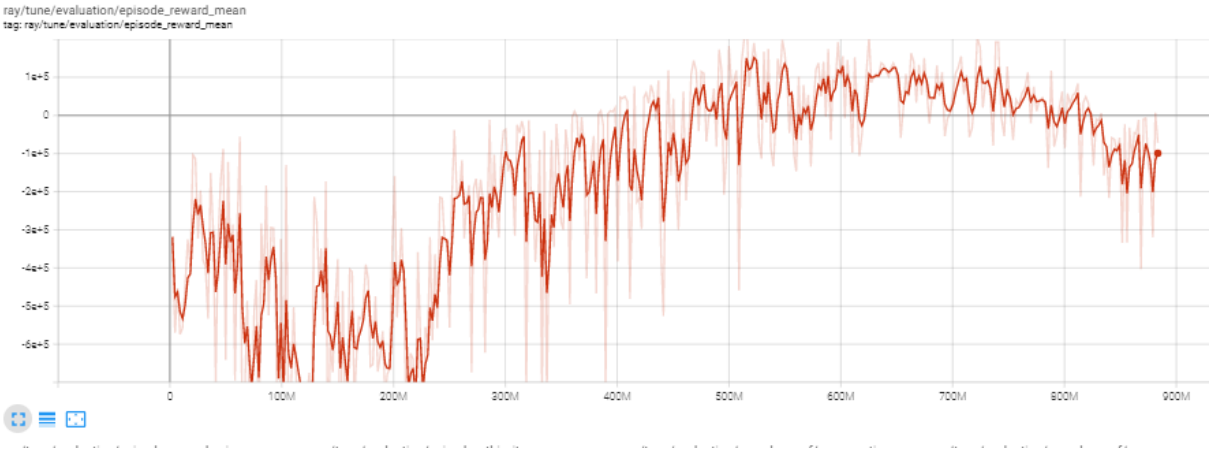


Resultate

Wenn dieses Modell auf 750 Iterationen simuliert wird bei einer maximalen Kapazität von 150 Hotels, dann gibt das folgende Resultate nach 80000 Simulationsschritten:

CQL Training

Die Entwicklung des Rewards über 80000 Iterationen mit jeweils 1024 Batches sieht es folgendermassen aus:



Das Modell scheint nach einer gewissen Zeit signifikant was gelernt zu haben. Auch wenn die Werte eher wieder absinken.

Eine Analyse des Simulationszeitraums für das Hotel 1709 gibt folgendes Resultat:

Rewards

Environment: Version 9
Simulationszeitraum: 18.12.2020 -30.12.2020
Hotel: 1709
Modelle: Referenz / Reservations, CQL, Fixer Preis 75CHF

In diesem Fall sind die Resultate nicht mehr 100 % reproduzierbar und vergleichbar, da die Nachfrage gesampelt wird.

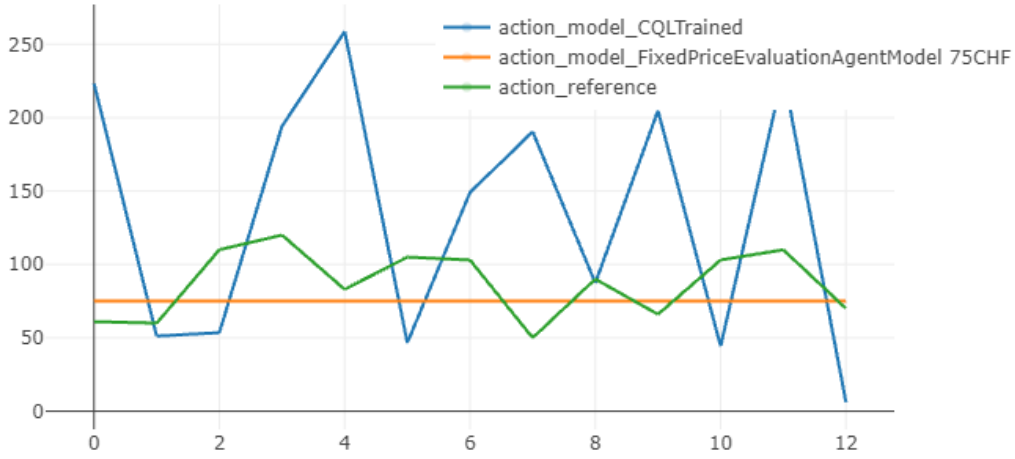
Folgende Totale Rewards wurden erzielt:

Referenz: -65748
CQL: -66093.9
Fixer Preis 75: -89700

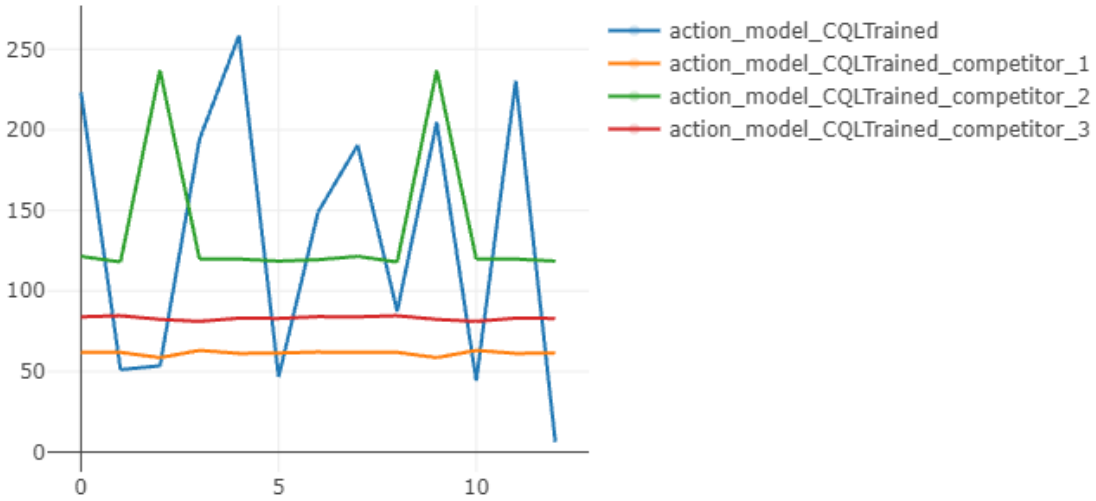
Über die Zeit wurden folgende Rewards erzielt:

Das CQL Model hat nicht wirklich eine stark sichtbare beste Strategie erreicht.

Wenn man die Aktionen vergleicht, wird sichtbar, dass das CQL Modell gelernt hat, einen stark schwankenden Preis zu setzen.



Wenn man es im Vergleich zu den Konkurrenten ansieht, erhält man folgendes Ergebnis:



Das Modell schafft es nicht sich von der Konkurrenz zu differenzieren.

Fazit

Das Modell scheint noch nicht mit dem grösseren State Space umgehen zu können und lernt nicht wirklich eine gute Strategie.

Variable Gewichtung der Nutzerpräferenz env v 10

Datum: 10.05.2022

Ziel: Nicht immer fixe Gewichtung

Beschreibung

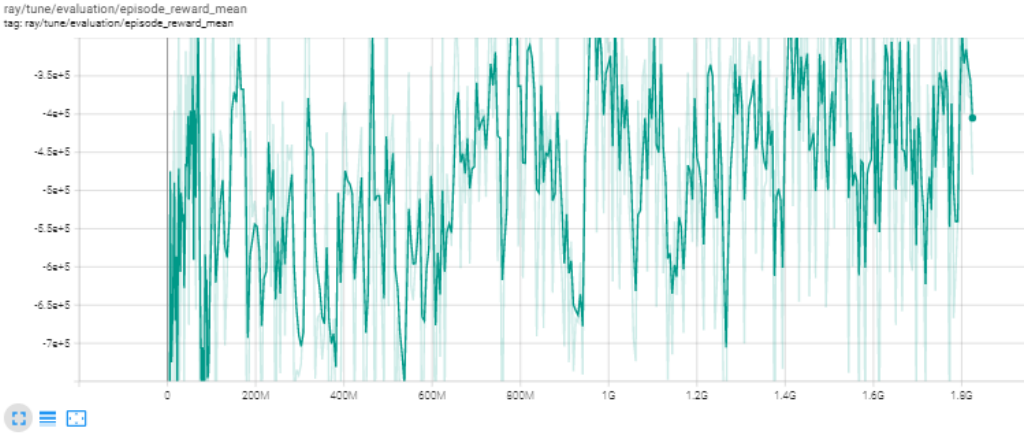
Anstatt das immer 0.5 und 0.5 gewählt wird, wird zufällig für jeden Kunden die Gewichtung zwischen 0.1 und 0.9 Preisgewichtung randomisiert. Environment v10.

Vorgehen

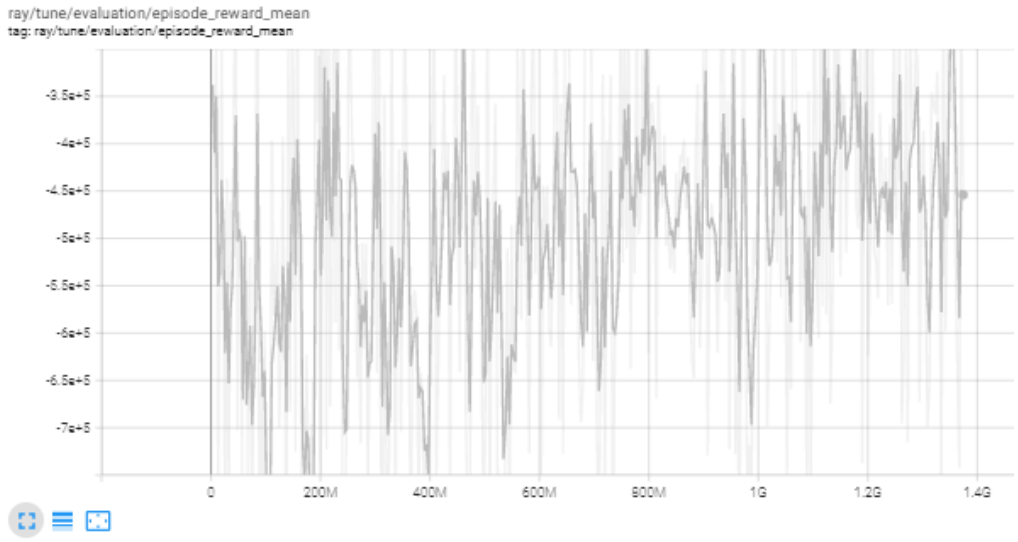
- Zwei Kundengruppen
  - Freizeit (Tourist)
  - Business
- Preisgewichtung * Preisrang + (1-Preisgewichtung) * Präferenz des Kunden (Präferenz mal Eigenschaft Hotel)
- + = Präferenz Business, - = Präferenz Leisure

Resultate

Es scheint nicht wirklich einen grossen Lerneffekt zu geben:



Bei einer Erhöhung der Modellgrösse von 64x64x64 auf 128x128x128 gab es nicht wirklich eine grosse Verbesserung.



Es gibt zwar eine geringe Aufwärtsbewegung, jedoch scheint diese nicht wirklich stark zu sein.

Eine grössere Trainingsdatenmenge von 2000 statt 500 Iterationen und eine längere Trainingsdauer von 500'000 statt 200'000 Schritte gibt folgendes Resultat:

Das Modell konvergiert auf der Umgebung nicht.

Fazit

Durch die sehr komplexe Modellierung scheint das Modell nicht zu konvergieren, evtl. müsste ein tieferes Netzwerk trainiert werden.

Da jedoch dieses Nachfragemodell nicht richtig umgesetzt werden kann, wird es nicht weiterverfolgt.

Evaluation des ENV V4 für Dokumentation

Datum: 17.05.2022
Ziel: Evaluation für Dokumentation
File: **https://gitlab.enterpriselab.ch/baa-dynamic-pricing/mlflow-results/-/commit/34d03f824ba7582efff7f00b22d80580fbe68e84**

Beschreibung

Es wurden folgende Modelle evaluiert:

- Fixpreismodell
- CQL Modell
- Referenzmodell
- Min Price 0.95

Vorgehen und Resultate

Siehe Dokumentation

Behavior Cloning

Datum: 04.05.2022
Ziel: Behavior Cloning Algorithmus
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/d0d454481125ca20533c282099e6fadcc63a791c

Beschreibung

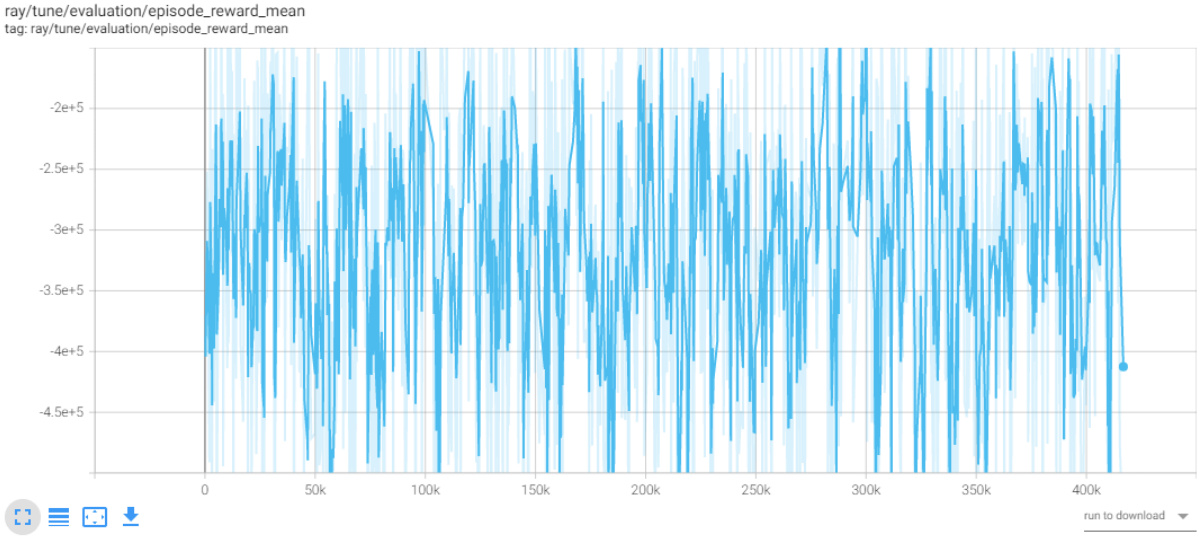
Ein anderer Algorithmus, welcher als Vergleich verwendet werden kann, ist der Behavior Cloning Algorithmus. Dieser versucht das Verhalten der offline Daten zu imitieren.

Vorgehen

- Neuer Runner hinzufügen
- Trainieren

Resultate

Es wird erkannt, dass der Mean Reward tief bleibt.



Fazit

Da das Nachfragemodell nicht besonders viel Sinn macht, wird dieser Algorithmus nicht integriert.

MARWIL Algorithmus

Datum: 05.05.2022 -23.05.2022
Ziel: Marwil Algorithmus
File: https://gitlab.enterpriselab.ch/baa-dynamic-pricing/rl-models/-/commit/2b48f640465bb84ffef3d836dec6b444eac4ac81

Beschreibung

Ein anderer Algorithmus, welcher als Vergleich verwendet werden kann, ist der MARWIL Algorithmus.

Vorgehen

- Neuer Runner hinzufügen
- Trainieren

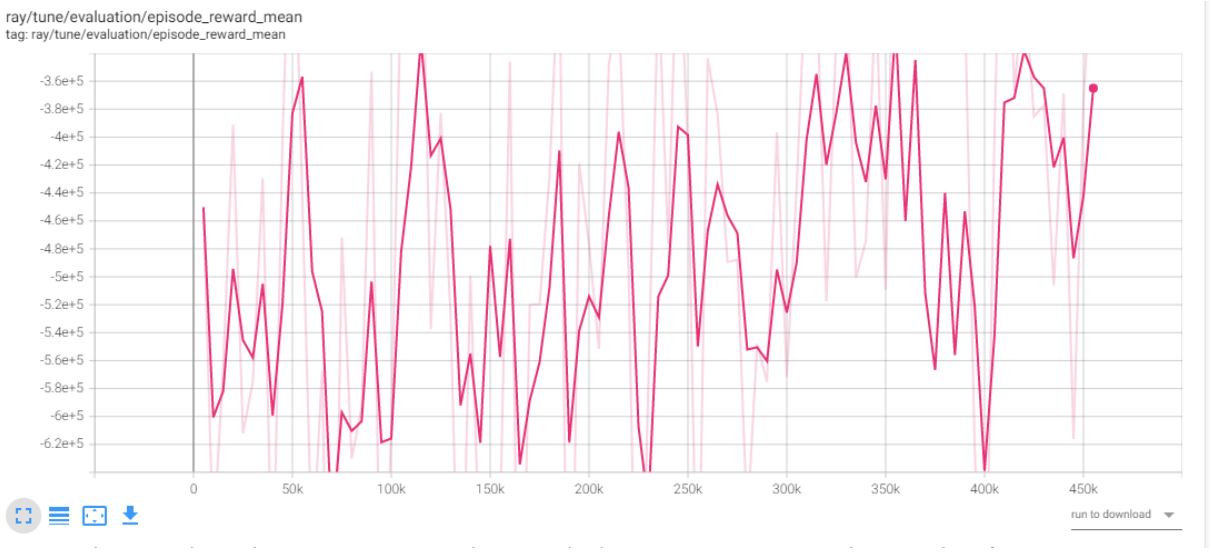
Das Problem, welches beim MARWIL Agenten auftritt, ist dass es als Aktion immer NaN ausgibt in der Evaluation!

Das Problem konnte mit folgender Konfiguration gelöst werden:

```
config["grad_clip"] = 40
```

Resultate

Dennoch steigt der Reward nicht an.



Fazit

In einer Weiterführung der Arbeit und bei einer weiteren Integration müssten die Hyperparameter des MARWIL Agenten verbessert werden.