ALADIN - Projektbericht

Projektseminar WS 2023/24

Inhaltsverzeichnis

1. Einleitung

Die Lehre an einer Universität oder Fachhochschule in Deutschland repräsentiert die höchste Ebene des Bildungssystems und ist darauf ausgerichtet, Studierenden umfassende theoretische Kenntnisse, praktische Fähigkeiten und wissenschaftliche Methodenkompetenz zu vermitteln. Um diese Fähigkeiten und Kompetenzen zu erlangen, benötigen die Studierenden vielseitiges Übungsmaterial, um das Wissen zu verinnerlichen.

Daher werden in der Lehre diverse Aufgabenformate benötigt, darunter beispielsweise Übungsaufgaben und Probeklausuren. Jedoch erfordert die Erstellung solcher Aufgaben einen erheblichen Zeitaufwand von Lehrkräften, die in der Regel dazu neigen, die Aufgaben manuell zu erstellen und zu korrigieren. Dies führt zu einer eingeschränkten Variation der Aufgabenstellungen, wodurch Studierende nur wenige oder gar identische Aufgabenformulierungen erhalten. Als Konsequenz entsteht ein Mangel an vielfältigen Übungsaufgaben für die effektive Vorbereitung auf Klausuren.

1.1. Vorstellung ALADIN

Die Bezeichung ALADIN steht als Akronym für "Generator für Aufgaben und Lösungshilfen aus der Informatik sowie angrenzenden Disziplinen". Dieser Generator wurde entwickelt, um die zufallsbasierte Generierung von einer Vielzahl von Aufgaben zu ermöglichen. Studierende erhalten somit die Möglichkeit, diese Aufgaben online zu lösen.

Die aktuelle Ausrichtung von ALADIN liegt auf Aufgabenstellungen, die primär aus den Bereichen Produktionswirtschaft, Datenbanken und Projektmanagement stammen. Im Rahmen des Projektseminars soll gezeigt werden, dass ALADIN Aufgabenbereiche aus interdisziplinären Fachgebieten eingesetzt werden kann, um die Vielseitigkeir und Anwendungsbreite von ALADIN hervorzuheben. Dies soll zur Digitalisierung der Lehre beitragen.

Die Software unterteilt sich in zwei Bereiche. Jeder Bereich hat dabei ein eigenes Framework. Das Framework für das Backend, mit welchem die Aufgaben sowie die Lösungen generiert werden heißt ALADIN. Für den zweiten Bereich (das Frontend) Es unterteilt sich in die Backendsoftware "ALADIN" und das dazugehörige Frontend "CARPET"

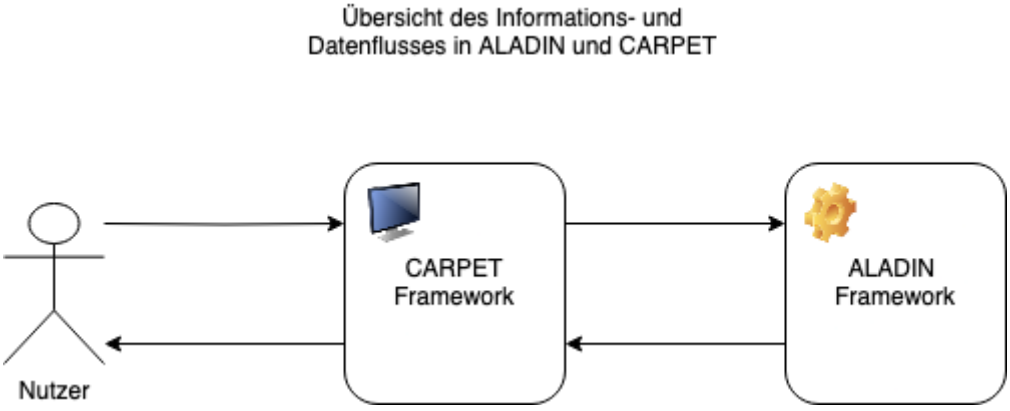


Abbildung 1.

1.2. Projektseminar

Im Rahmen des Projektseminars setzte sich die Gruppe, bestehend aus sechs Studierenden, mit der Entwicklung von Aufgabengeneratoren in ALADIN auseinander. Die Zielsetzung war die Erarbeitung und Umsetzung von Generatoren, wobei die Mehrheit der Teilnehmer selbständig an der Entwicklung eines Generators beteiligt war.

1.2.1. Ziel des Projektseminars

Das Ziel des Projektseminars besteht darin, die Erforschung und Umsetzung neuer Aufgabentypen in der ALADIN-Umgebung. Der Fokus liegt dabei auf die Entwicklung generativer Aufgabentypen. Die Studierenden haben die Möglichkeit aus verschiedenen Fachbereichen wie Musiktheorie, Chemie, Regular Expression, Spatial SQL, Endliche Automaten, Entscheidungsbäume, Hierarchische Clusteranalyse und Phrase Structure Trees zu wählen.

1.2.2. Ablauf des Projektseminars

1. **Auswahl eines Aufgabentyps**
   Die Studierenden wählen zu Beginn des Projektseminars den Aufgabentypen aus, den sie für die Entwicklung in ALADIN umsetzen möchten.
2. **Einarbeitung in den Aufgabentyp**
   In den nächsten Wochen wurde ein generelles Verständnis für das Problem entwickelt und fachliches Wissen angeeignet welches später helfen sollte die Aufgabe in ein Programm umzusetzen.
3. **Konzeption und Implementierung eines Prototypen**
   Folgend wurde zu jedem Konzept ein Prototyp in einer selbst ausgewählten Programmiersprache implementiert. Das Ziel bestand darin ein grundlegendes Verständnis für die Komplexität zu entwickeln und einen Struktur für eine spätere Implementierung in Aladin zu schaffen
4. **Umsetzung in ALADIN**
   Nachdem jeder Studierende die grundlegenden Funktionen seines Prototpyps in einem externen Programm umgesetzt hatte, begannen sie diesen in das vorhandene ALADIN Framework zu integrieren. Voraussetzung dafür war die Einarbeitung in ALADIN und CARPET, sowie das fundamentale Verständnis über deren Zusammenarbeit.
5. **Präsentation des entwickelten Aufgabentyps**
   Am Ende des Semesters wurden alle bearbeiteten Projekte vor den anderen Studierenden vorgestellt. Unter den Augen aller Professoren die im Rahmen des Projektseminars ein Projekt betreut haben, konnten Studierende aus anderen Projekte Fragen zu den erbrachten Arbeiten stellen.

2. Aufgabentyp - Endlicher Automat

2.1. Aufgabenstellung

2.2. Aufgabenbeschreibung

2.3. Anforderung

2.4. Planung

2.5. Technische Umsetzung

2.6. Ergebnisse

2.7. Ausblick

3. Aufgabentyp - Clusteranalyse

3.1. Aufgabenstellung

3.2. Aufgabenbeschreibung

3.3. Anforderung

3.4. Planung

3.5. Technische Umsetzung

3.6. Ergebnisse

3.7. Ausblick

4. Aufgabentyp - Entscheidungsbaum

4.1. Aufgabenbeschreibung

Im Rahmen des Projektseminars wurde ein Aufgabentyp entwickelt, der Studierenden die Berechnung und Erstellung eines Entscheidungsbaums ermöglicht. Die Grundidee besteht darin, dass Studierende anhand eines vorab generierten Datensatzes und eines festgelegten Algorithmus interaktiv einen Entscheidungsbaum für diesen spezifischen Datensatz aufbauen können. Dabei soll es möglich sein, Beschriftungen sowie berechnete Werte für die einzelnen Knoten und Kanten hinzuzufügen.

Beispiel 1. Aufgabenstellung für den Studierenden

> "Erstellen Sie anhand des generierten Datensatzes einen Entscheidungsbaum. Nutzen Sie für die Berechnung den ID3-Algorithmus und für den Aufbau der Lösung den interaktiven Entscheidungsbaum. Tragen Sie die richtigen Beschriftungen und Werte in die Knoten und an den Kanten ein."

Die Aufgabe zielt darauf ab, den Studierenden die Möglichkeit zu bieten, die Berechnung eines Entscheidungsbaums mithilfe eines Algorithmus zu erlernen und zu vertiefen. Die Integration interaktiver Elemente ist darauf ausgerichtet, das Verständnis für den Algorithmus, als auch für den Aufbau eines Entscheidungsbaums zu erleichtern und einen praktischen Lernansatz zu fördern.

4.2. Anforderungen

Für den Aufgabentypen ergeben sich folgende Anforderungen:

1. **Zufällige Datensatzgenerierung**
   Die Generierung eines Datensatzes erfolgt zufällig, wobei sich nur die Berechnungswerte ändern dürfen. Die Struktur des Baumes soll erhalten bleiben.
2. **Datensatzgenerierung basierend auf vorhandenem Baum**
   Die Generierung des Datensatzes erfolgt anhand eines bereits vorgegebenen Entscheidungsbaumes.
3. **Interaktiver Aufbau des Entscheidungsbaums**
   Der Entscheidungsbaum sollte interaktiv aufgebaut werden können, wodurch beliebig viele Knoten und Kanten in unterschiedlichen Tiefen hinzugefügt oder entfernt werden können.
4. **Überprüfung des eingegebenen Entscheidungsbaums**
   Der vom Studierenden aufgebauten Entscheidungsbaum wird auf Richtigkeit sowohl hinsichtlich der eingegebenen Werte als auch der Struktur überprüft.
5. **Flexible Positionierung der Werte**
   Bei der Prüfung des Entscheidungsbaums wird berücksichtigt, dass die Werte der Knoten und Kanten auf der Baumebene an unterschiedlichen Stellen eingetragen werden können.
6. **Berücksichtigung von Mehrfachlösungen**
   Es wird berücksichtigt, dass bei der Überprüfung des Entscheidungsbaums Mehrfachlösungen

möglich sind. Das bedeutet, je nach Berechnung für einen Knoten im Entscheidungsbaum können unterschiedliche Merkmale als korrekte Lösungsmöglichkeit gewertet werden.

7. **Feedback nach erfolgreicher Lösung**
   Nach erfolgreicher Lösung der Aufgabe erhält der Studierende eine Benachrichtigung, dass die Aufgabe erfolgreich gelöst wurde.
8. **Generierung neuer Aufgaben**
   Nach Abschluss der Aufgabe sollte die Möglichkeit bestehen, eine neue Aufgabe mit einem neuen Datensatz zu generieren.

4.3. Planung

4.3.1. Wöchentliche Fortschrittsbesprechung und Umsetzungsplan

Im Rahmen des Projektseminars wurde ein wöchentliches Meeting etabliert. Die Besprechungen fanden einmal in der Woche statt und wurden von Herrn Professor Munkelt und Herrn Christ geleitet. Das Meeting diente für die Evaluierung der gesetzten und erreichten Ziele, die Vorstellung und Diskussion von Ideen und Vorschlägen zur Umsetzung seines jeweiligen Aufgabentyps sowie für Diskussionen über auftretende Probleme und Unklarheiten.

Um die Umsetzung der Aufgaben und Ziele zu gewährleisten, wurden feste Zeiten für die Bearbeitung eingeplant. Zweimal pro Woche waren jeweils mindestens vier Stunden für die Arbeit an den Aufgaben und Zielen der Woche reserviert. Dies diente dazu, um sicherzustellen, dass das Projekt innerhalb der definierten Zeitvorgabe erfolgreich abgeschlossen werden konnte.

4.3.2. Eigenständige Berechnung einer ID3-Algorithmus-Aufgabe

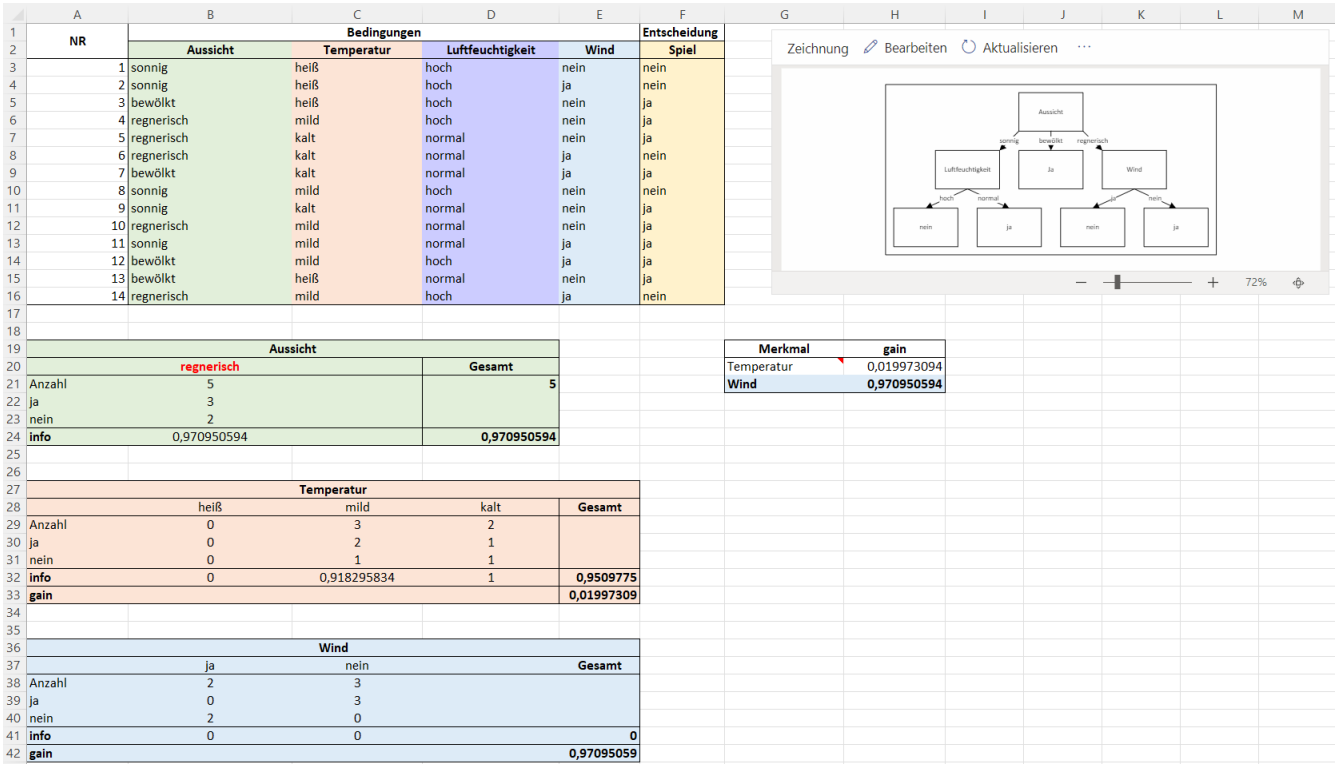
Zu Beginn des Projekts stand die Auseinandersetzung mit dem Aufgabentypen zur Berechnung eines Entscheidungsbaums im Vordergrund. Ein entscheidender Schritt dabei war, die Aufgabe zunächst eigenständig zu lösen. Ziel war es, das persönliche Verständnis für den Aufbau und die wesentlichen Aspekte dieser Art von Aufgabe zu vertiefen. Dies beinhaltete die Analyse, welche Faktoren bei der Konstruktion einer solchen Aufgabe von Bedeutung sind, welche Schritte in der Berechnung notwendig sind, welche Lösungswege existieren und wie das Endresultat, der Aufbau des Entscheidungsbaums, gestaltet sein sollte.

Im wöchentlichen Meeting wurde in Absprache festgehalten, dass der zu entwickelnde Aufgabentyp auf dem ID3-Algorithmus basieren sollte. Dies erforderte zunächst, das Verständnis für die Funktionsweise und Anwendung des ID3-Algorithmus zu erlangen. Weiter wurden Beispielaufgaben durchgerechnet, wobei verschiedene Problemstellungen berücksichtigt wurden, wie zum Beispiel die Berechnung von Aufgaben unter dem Gesichtspunkt von Mehrfachlösungen.

Das selbstständige Lösen der Aufgabe diente somit als Ausgangspunkt für die Ideensammlung für die Umsetzung und Entwicklung des interaktiven Aufgabentyps.

ANMERKUNG	Für die Recherche des ID3-Algorithmus und die Auswahl von Beispielaufgaben zur eigenständigen Bearbeitung wurden die folgenden Quellen herangezogen: 1. https://www.in.th-nuernberg.de/professors/Holl/Personal/DataMining_Ba-

chelor.pdf
2. Literatur: Frochte, J. (2019). Maschinelles Lernen: Grundlagen und Algorithmen in Python. Carl Hanser Verlag.
3. Unterlagen aus dem 4. Semester: Toll, A. (2023). 7. Automatische Generierung von Entscheidungsbäumen. Modul Business Intelligence. Hochschule für Technik und Wirtschaft Dresden.



NR	Bedingungen				Entscheidung
	Aussicht	Temperatur	Luftfeuchtigkeit	Wind	Spiel
1	sonnig	heiß	hoch	nein	nein
2	sonnig	heiß	hoch	ja	nein
3	bewölkt	heiß	hoch	nein	ja
4	regnerisch	mild	hoch	nein	ja
5	regnerisch	kalt	normal	nein	ja
6	regnerisch	kalt	normal	ja	nein
7	bewölkt	kalt	normal	ja	ja
8	sonnig	mild	hoch	nein	nein
9	sonnig	kalt	normal	nein	ja
10	regnerisch	mild	normal	nein	ja
11	sonnig	mild	normal	ja	ja
12	bewölkt	mild	hoch	ja	ja
13	bewölkt	heiß	normal	nein	ja
14	regnerisch	mild	hoch	ja	nein

Zeichnung Bearbeiten Aktualisieren

Aussicht		
regnerisch		Gesamt
Anzahl	5	5
ja	3	
nein	2	
info	0,970950594	0,970950594

Merkmal	gain
Temperatur	0,019973094
Wind	0,970950594

Temperatur				
	heiß	mild	kalt	Gesamt
Anzahl	0	3	2	
ja	0	2	1	
nein	0	1	1	
info	0	0,918295834	1	0,9509775
gain				0,01997309

Wind			
	ja	nein	Gesamt
Anzahl	2	3	
ja	0	3	
nein	2	0	
info	0	0	0
gain			0,97095059

Abbildung 2. Beispiel Berechnungsaufgabe eines Entscheidungsbaums in Excel

4.3.3. Konzeption des interaktiven Aufgabentyps

Nachdem ein grundlegendes Verständnis für die Berechnung von Entscheidungsbäumen erworben war, war der nächste Schritt im Projekt die Entwicklung einer Aufgabenstellung für die Berechnung eines Entscheidungsbaums. Das Ziel bestand darin, wie eine Aufgabenstellung gestaltet werden könnte, um Studierende den interaktiven Aufbau von Entscheidungsbäumen zu ermöglichen. Dieser Schritt beinhaltete die Erstellung von Wireframes, um die Konzeptideen visuell zu erfassen.

Die erste konzeptionelle Idee sah vor, die Aufgabenstellung durch eine Art Menüführung zu strukturieren. Hierbei sollte der Studierende zu Beginn die relevanten Merkmale und deren Ausprägungen eingeben, woraus anschließend ein zufälliger Datensatz generiert werden würde. Basierend auf diesem Datensatz würde der Entscheidungsbaum schrittweise berechnet werden. Die Umsetzung erfolgte durch die schrittweise Anzeige eines Menüs für jede Knoten des zu konstruierenden Baums. In diesem Menü konnte der Studierende die Bezeichnungen der Merkmale und die berechneten Werte für den Informationsgewinn eingeben. Dabei wurde für jeden angezeigten Knoten des Baums das relevante Merkmal ausgewählt. Der iterative Prozess sollte es dem Studierenden ermöglichen, den Entscheidungsbaum schrittweise aufzubauen.

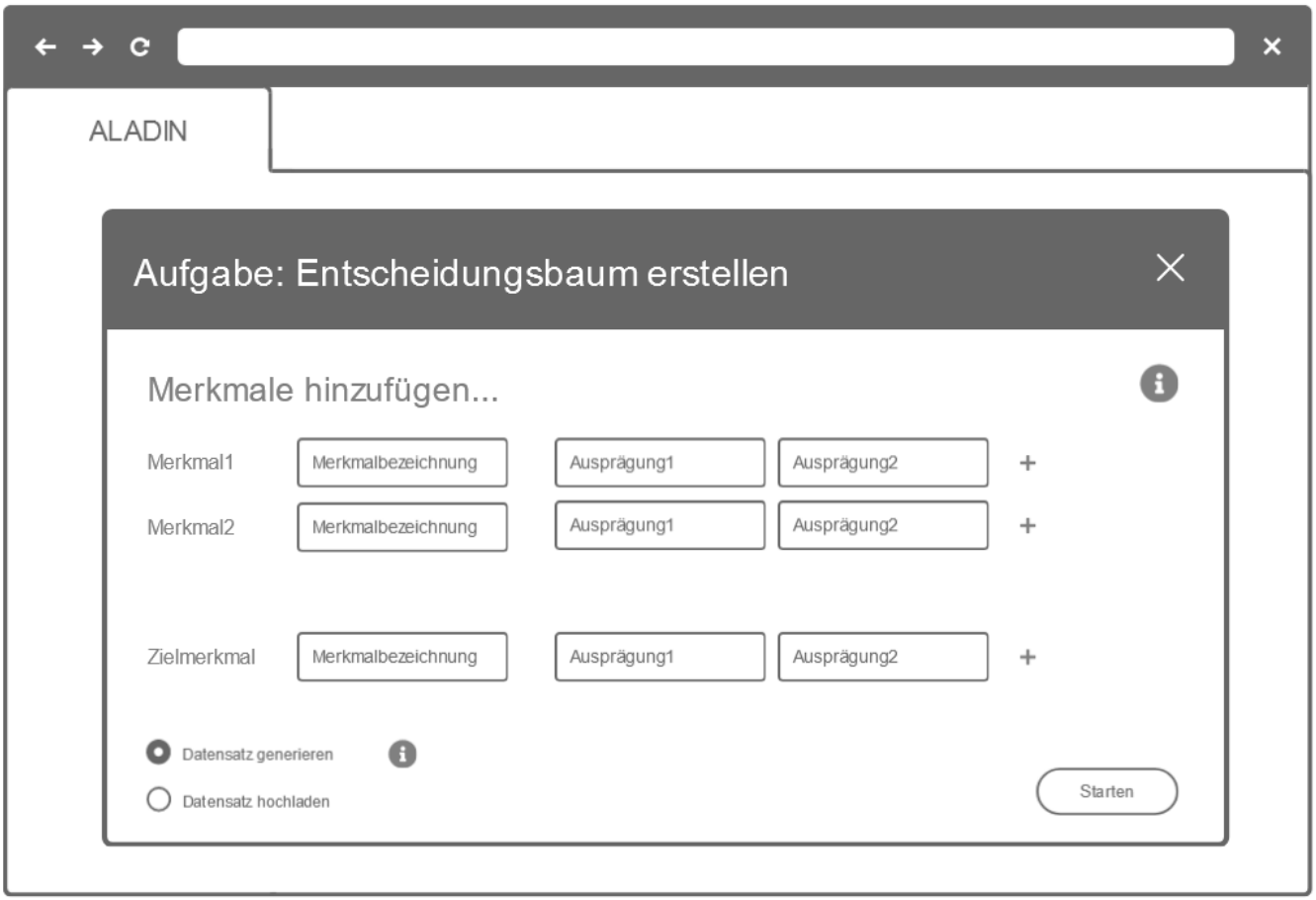


Abbildung 3. Wireframe: Erster Entwurf des Startmenüs für die Aufgabenstellung

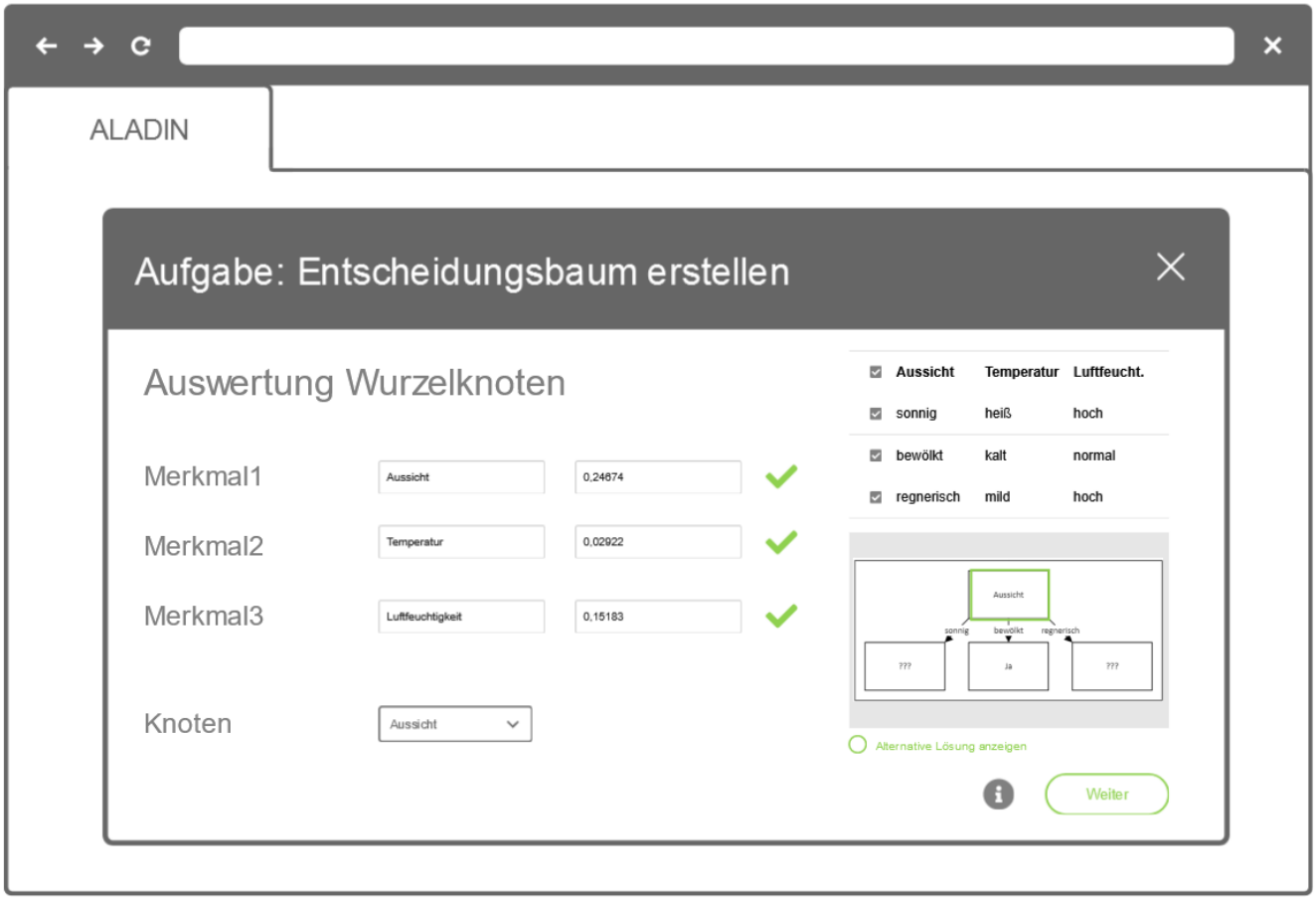


Abbildung 4. Wireframe: Erster Entwurf für die Auswertung der eingetragenen Werte des Wurzelknotens

Die Weiterentwicklung des Aufgabentyps wurde durch eine Idee im Rahmen der wöchentlichen

Besprechung vorangetrieben, dass der Entscheidungsbaum interaktiv aufgebaut werden soll. Dies bedeutet, dass Studierende einen generierten Datensatz zur Berechnung angezeigt bekommen, um die berechnete Lösung in einer Umgebung aktiv als Entscheidungsbaum aufbauen zu können. Dabei soll zu Beginn kein vorgefertigter Baum vorhanden sein. Studierende sollen beliebig viele Knoten hinzufügen, entfernen und alle relevanten Werte wie die Bezeichnung des Merkmals bzw. die Klasse für Blattknoten und berechnete Informationsgewinne eigenständig eintragen können. Die Ausprägungen der Merkmale und die berechneten Entropien können entlang der Kanten des Baumes ergänzt werden. Nachdem der Studierende den Entscheidungsbaum vollständig aufgebaut und alle Werte eingetragen hat, kann er auf "Auswerten" klicken. Dabei werden die Stellen des Entscheidungsbaums angezeigt, die die korrekten Werte enthalten, sowie jene, die falsch sind. Dieser Prozess kann wiederholt werden, bis der Studierenden den Entscheidungsbaum vollständig und korrekt aufgebaut hat. Sobald die Aufgabe gelöst wurde, erscheint eine Meldung für den Studierenden und die Möglichkeit, eine neue Aufgabe zu berechnen. Eine weitere Idee bestand darin, den Studierenden unterstützende Hinweise anzubieten. Sollte beispielsweise Unklarheiten über die Formeln für die Berechnung des Informationsgewinns oder der Entropie bestehen, könnte der Studierende einen Hinweis-Button aktivieren. Nach einem Klick würden dann entsprechende Hinweise zur Formelberechnung angezeigt, um bei der Berechnung des Entscheidungsbaums zu unterstützen.

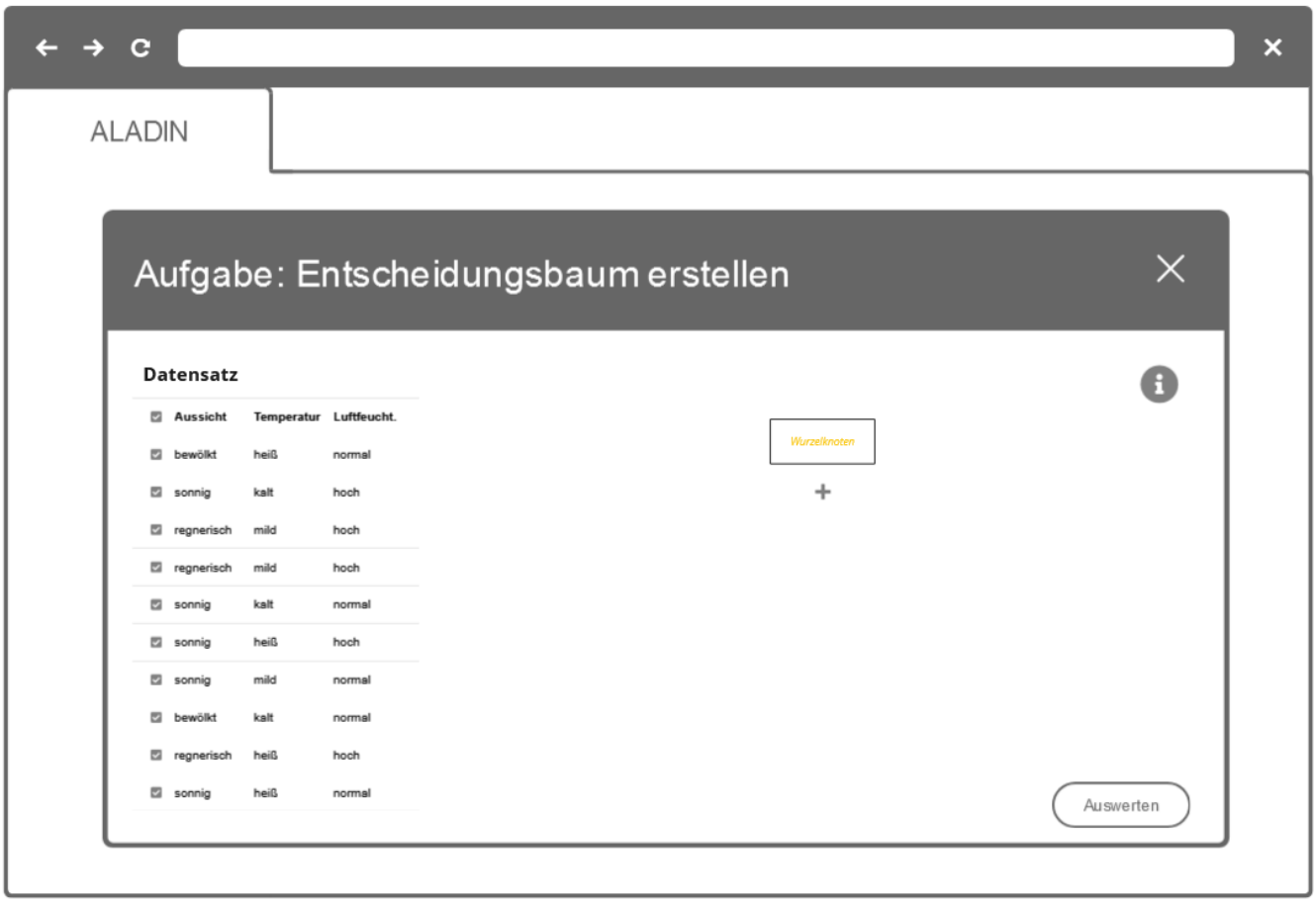


Abbildung 5. Wireframe: Finaler Entwurf der Startansicht mit dem Wurzelknoten des zu konstruierenden Entscheidungsbaums

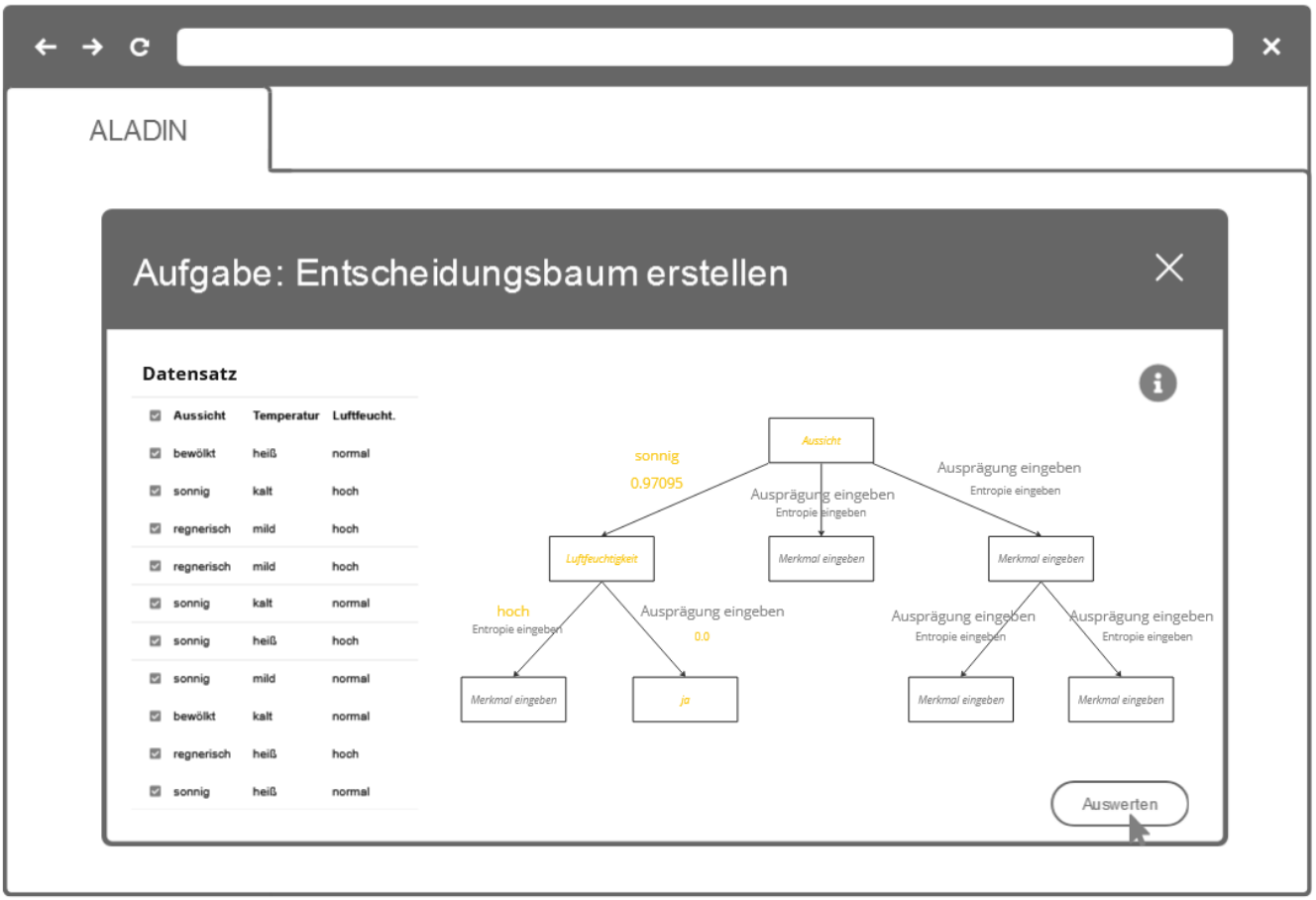


Abbildung 6. Wireframe: Finaler Entwurf für den Aufbau des Entscheidungsbaums und das Einfügen der Werte

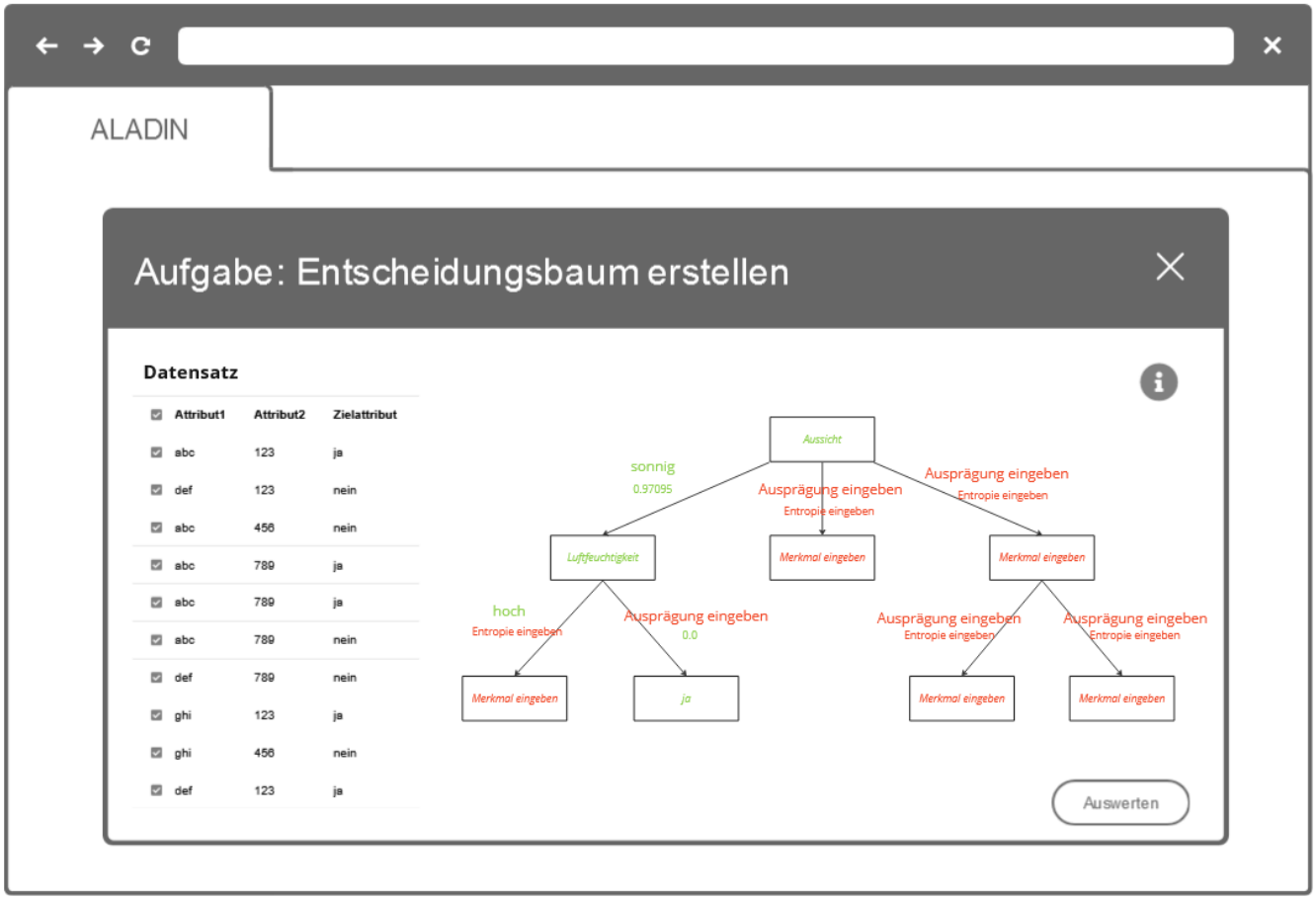


Abbildung 7. Wireframe: Finaler Entwurf für die Auswertung der Eingaben im Entscheidungsbaum

Abbildung 8. Wireframe: Finaler Entwurf für die Anzeige der Meldung über die erfolgreiche Lösung der Aufgabe

4.3.4. Entwicklung und Umsetzung des Prototyps

Nachdem der Entwurf für den Aufgabentypen festgelegt wurde, wurde mit der Umsetzung des Prototyps begonnen. Der Fokus lag zunächst auf der Implementierung des ID3-Algorithmus in der Programmiersprache Python mithilfe der Entwicklungsumgebung Jupyter Notebook. Da zu diesem Zeitpunkt noch keine Erfahrung mit Python und Jypter Notebook vorhanden war, erfolgte zunächst eine Einarbeitung in diese Technologien. Die Entscheidung für Python fiel aufgrund seiner Eignung, komplexe Aufgaben wie die Berechnung von Entscheidungsbäumen einfach zu programmieren. Darüber hinaus wurde in den wöchentlichen Meetings hervorgehoben, dass Python-Programme problemlos in ALADIN integriert werden können, was die Wahl von Python weiter unterstütze.

Anfänglich wurden verschiedene Beispielcodes aus unterschiedlichen Quellen und Bibliotheken für den ID3-Algorithmus ausprobiert und getestet. Im Verlauf der Weiterentwicklung des Algorithmus wurde jedoch deutlich, dass es herausfordernd war, auf die berechneten Werte wie Informationsgewinn und Entropie zuzugreifen, da diese Aspekte in den Beispielcodes und Bibliotheken oft nicht ausreichend berücksichtigt wurden. Infolgedessen wurde der Algorithmus selbst entwickelt, um auf die berechneten Werte sowie die Struktur des Baums zugreifen zu können.

Bei der Entwicklung des eigenen Algorithmus wurde darauf geachtet, dass die berechneten Werte des Entscheidungsbaums gespeichert werden können. Gleichzeitig wurde versucht, die Struktur des Entscheidungsbaums zu speichern, um bei der Überprüfung des eingegebenen Entscheidungsbaums des Studierenden auf die korrekten Werte und Strukturen zugreifen zu können. Dazu wurden auch erste grafische Versuche mit der Bibliothek PyDotPlus unternommen, um Implementie-

rungsmöglichkeiten für die Darstellung eines Entscheidungsbaums zu skizzieren. Diese Versuche beinhalteten die Verwendung verschiedener Datensätze, um sicherzustellen und zu testen, dass der Algorithmus bei unterschiedlichen Datensätzen funktioniert, insbesondere bei solchen, die Mehrfachlösungen ermöglichen.

Tabelle 1. Übersicht über die verwendeten Technologien für die Entwicklung des Algorithmus

Technologie	**Erläuterung**
Jupyter Notebook	Wurde als Entwicklungsumgebung verwendet, für das Ausführen und Testen des geschriebenen Algorithmus
Python	Wurde als Programmiersprache verwendet, für das Schreiben des Algorithmus als Code
Pandas	Verwendete Bibliothek, um die Daten in tabellarischer Form (mit DataFrames) zu organisieren und die Verwendung von Funktionen für unterschiedliche Operationen der Datenbearbeitung
Numpy	Verwendete Bibliothek für die Verwendung von mathematischen Funktionen für die Berechnungen des Algorithmus
PyDotPlus	Verwendete Bibliothek für die grafische Darstellung des Entscheidungsbaums

Nach Abschluss der Algorithmusentwicklung wurde mit der Implementierung des Aufgabentyps als Prototyp begonnen. Die Backend-Entwicklung erfolgte weiterhin mit Python, wobei der zuvor implementierte Algorithmus verwendet wurde. Außerdem wurde Flask, ein Web-Framework für Python verwendet, um die Serveranwendung zu entwickeln. In der Frontend-Entwicklung kamen HTML, CSS und JavaScript für die grundlegende Entwicklung der Benutzeroberfläche zum Einsatz. Zudem wurde die JavaScript-Bibliothek D3.js verwendet, um den Entscheidungsbaum visuell umzusetzen. Basierend auf dem entwickelten Wireframe wurde die Aufgabe so implementiert, dass der Datensatz, aus dem der Entscheidungsbaum berechnet werden sollte, auf der linken Seite angezeigt wird, während rechts der interaktive Entscheidungsbaum dargestellt wird. Oberhalb wird die Aufgabenstellung für den Studierenden angezeigt.

Der interaktive Entscheidungsbaum kann folgendermaßen bedient werden: Durch einen Klick auf einen Knoten wird ein Prompt angezeigt, in dem der Studierende die Bezeichnung des Merkmals eingeben kann. Das gleiche Vorgehen gilt für einen Klick auf eine Kante, wobei hier die Bezeichnung sowie die zugehörige Entropie eingegeben werden können. Um weitere Knoten hinzuzufügen, kann der Studierende auf das ""-Zeichen unterhalb des Knotens klicken und im Prompt die Anzahl der hinzuzufügenden Knoten auf der nächsten Ebene angeben. Ebenso kann er die Anzahl der Knoten reduzieren oder die Knoten auf der Ebene wieder löschen, indem er erneut auf das ""-Zeichen unterhalb des Knotens klickt und die neue Anzahl eingibt oder die Anzahl auf 0 setzt, um die hinzugefügten Knoten zu entfernen. Dabei ist es möglich, auf jeder Ebene des Baumes die Anzahl der Knoten zu verändern. Allerdings werden die zuvor eingetragenen Werte für Knoten und Kanten mit den Standardwerten überschrieben, wenn die Anzahl auf der Ebene verändert wurde. Daher ist es erforderlich, nachträglich diese Werte erneut einzugeben.

Zur Überprüfung der Eingaben des Studierenden wird auf den Button "Speichern" geklickt. Dabei werden alle Knoten und Kanten rot eingefärbt, welche falsch sind. Hier werden folgende Prüfungen durchgeführt:

- **Schritt 1**: Generelle Überprüfung, ob die Eingabe der Bezeichnung korrekt ist. Zum Beispiel, ob die Bezeichnung des Merkmals vorhanden ist.
- **Schritt 2**: Um unterschiedliche Aufbaumöglichkeiten des Entscheidungsbaums zu ermöglichen, wird überprüft, ob der eingegebene Wert auf der richtigen Ebene eingetragen wurde.
- **Schritt 3**: Wenn die Eingabe korrekt ist und sich auf der richtigen Ebene befindet, wird überprüft, ob der übergeordnete Knoten bzw. die übergeordnete Kante korrekt ist.

Wenn alle Prüfungen erfolgreich verlaufen sind, wird die Eingabe des Studierenden als korrekt betrachtet, anderenfalls wird sie als falsch bzw. rot angezeigt.

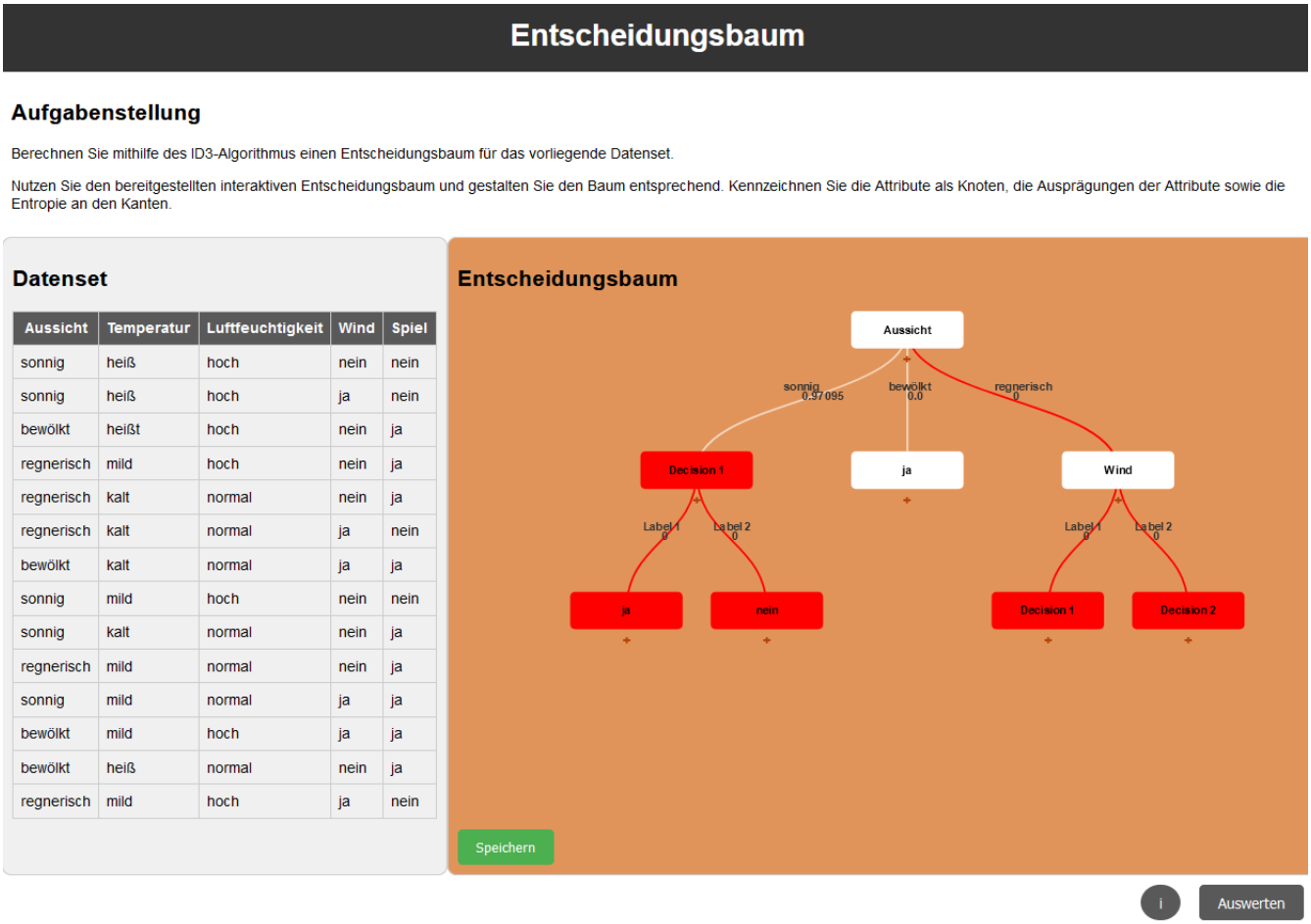
Zur weiteren Kontrolle wurde während des Prototyps eine Ausgabe implementiert, die den möglichen Fehler bei der Eingabe angibt. Um das Debugging zu erleichtern, wurde zwischen Knoten und Kanten unterschieden, um anzuzeigen, wo der Fehler aufgetreten ist. Es wurden die folgenden Prüfungen implementiert:

- **1. Prüfung**: "Label/Entropie nicht richtig"
- **2. Prüfung**: "Parent nicht richtig" - Hierbei handelt es sich um den übergeordneten Knoten bei einer Kante bzw. die übergeordnete Kante bei einem Knoten

Die Implementierung der Überprüfung auf Mehrfachlösungen mittels des ID3-Algorithmus wurde so realisiert, dass die eingegebenen Merkmale des Studierenden als Basis genutzt werden. Wenn während der Berechnung des Entscheidungsbaums an einem bestimmten Knoten mehrere Lösungen möglich sind, wird überprüft, welches Merkmal der Studierende an diesem Knoten eingetragen hat. Wenn die Eingabe mit einer der Lösungsmöglichkeiten übereinstimmt, wird dieses Merkmal für die fortlaufende Berechnung des Entscheidungsbaums verwendet. Andernfalls wird das zuerst berechnete Merkmal als Lösungsmöglichkeit ausgewählt. Durch dieses Vorgehen ist es nicht erforderlich, unterschiedliche Lösungsbäume zu speichern, was die Überprüfung des eingegebenen Entscheidungsbaums vereinfacht und beschleunigt.

Tabelle 2. Übersicht über die verwendeten Technologien für die Entwicklung des Prototyps

Technologie	**Erläuterung**
Backend	**Python** als verwendete Programmiersprache **Flask** als Webframework für Python für die Umsetzung einer Serveranwendung **Pandas** und **Numpy** als verwendete Bibliotheken
Frontend	**HTML**, **CSS** und **JavaScript** für den grundlegenden Aufbau und Umsetzung der Benutzeroberfläche des Aufgabentyps **D3.js** als verwendete JavaScript-Bibliothek zur grafischen Erstellung von Entscheidungsbäumen



Falsche Knoten:

- Node ID: 2 (Decision 1) - Label nicht richtig
- Node ID: 4 (ja) - Parent nicht richtig
- Node ID: 6 (nein) - Parent nicht richtig
- Node ID: 12 (Decision 1) - Label nicht richtig
- Node ID: 14 (Decision 2) - Label nicht richtig

Falsche Kanten:

- Edge ID: 3 (Label 1, 0) - Label/Entropie nicht richtig
- Edge ID: 5 (Label 2, 0) - Label/Entropie nicht richtig
- Edge ID: 9 (regnerisch, 0) - Label/Entropie nicht richtig
- Edge ID: 11 (Label 1, 0) - Label/Entropie nicht richtig
- Edge ID: 13 (Label 2, 0) - Label/Entropie nicht richtig

Abbildung 9. Beispielansicht des Prototyps

4.3.5. Generierung neuer Datensätze

Bei der Generierung eines neuen Datensatzes für die Erstellung einer neuen Berechnungsaufgabe stand die Herausforderung im Vordergrund, die Struktur des Entscheidungsbaums aus der Berechnung des Originaldatensatzes unverändert zu lassen. Dabei sollten ausschließlich an den berechneten Werten wie Informationsgewinne und Entropien Veränderungen vorgenommen werden. Das Ziel bestand darin, neue Datenpunkte auf Grundlage des bereits vorhandenen Entscheidungsbaums zu generieren. Hierbei war es wichtig, die Verteilung der Datenpunkte beizubehalten, sodass auch Merkmale, die im Entscheidungsbaum nicht vorhanden sind, in der Berechnung des Entscheidungsbaums aus dem neuen Datensatz weiterhin nicht berücksichtigt werden. Dadurch sollte die Struktur des Baumes erhalten bleiben.

Bei der Umsetzung wurde die Bootstrap-Sampling-Methode gewählt. Die Bootstrap-Sampling-Methode ist eine statistische Methode, die es ermöglicht, aus einer vorhandenen Stichprobe wiederholt neue Stichproben zu ziehen. Dabei können Statistiken wie der Mittelwert oder andere Kennzahlen berechnet werden. Das Vorgehen beginnt mit einem Gesamtdatensatz, aus dem zufällig

Datenpunkte ausgewählt werden. Dabei können bereits ausgewählte Datenpunkte erneut gewählt werden. Dieser Vorgang wird mehrfach wiederholt, sodass eine Vielzahl von Stichproben erstellt wird.

ANMERKUNG	Für die Recherche der Bootstrap-Sampling-Methode wurden folgende Quellen verwendet: https://novustat.com/statistik-blog/resampling-methoden-bootstrapping-statistik-permutation-test.html https://www.analyticsvidhya.com/blog/2020/02/what-is-bootstrap-sampling-in-statistics-and-machine-learning/

Für die Generierung neuer Datensätze wurde daher die Bootstrap-Sampling-Methode gewählt, da sie eine unkomplizierte und effiziente Möglichkeit bot, basierend auf bereits vorhandenem Datensatz neue Datenpunkte zu generieren. Bei der Umsetzung der Methode war es daher möglich, den bereits bestehenden Entscheidungsbaum zu verwenden, um wiederholt neue Datensätze zu erstellen. Dabei war es möglich, nicht die Struktur des Entscheidungsbaums zu verändern. Die Umsetzung der Methode wurde mit Python realisiert.

4.4. Technische Umsetzung in ALADIN

Nachdem der Prototyp alle erforderlichen Funktionalitäten hatte, bestand der nächste Schritt darin, diese Funktionalitäten in ALADIN zu integrieren und den Aufgabentypen final zu entwickeln. Bei der Umsetzung des Aufgabentypen in ALADIN wurde sich an den bereits implementierten Gozintograph Aufgabentyp zurückgegriffen. Dies ermöglichte einerseits eine erleichterte Einarbeitung und andererseits die Nutzung und Weiterentwicklung bereits vorhandener Komponenten.

4.4.1. Implementierung in ALADIN

Im Backend wurde der ID3-Algorithmus implementiert, um den Lösungsbaum zu berechnen, sowie die Überprüfung des vom Studierenden erstellten Entscheidungsbaums. Die Auswertung des eingegebenen Entscheidungsbaums erfolgt als JSON-Rückgabe an das Frontend, um weitere Bearbeitungen zu ermöglichen. Dabei wurden in ALADIN folgende Daten hinzugefügt oder gendert:

Im Verzeichnis: src\server\tempTaskGraphStorage\tasks\

- **DecisionTree.json**: Festlegung der Komponenten und Schnittstellen für den Aufgabentyp

Im Verzeichnis: src\Tasks\DecisionTrees\

- **dtBuildDecisionTree.py**: Python-Skript, welches von der Wrapperklasse in WrapperValidateDecisionTree.ts ausgerufen wird und zuständig für die Berechnung des Lösungsbaums sowie die Überprüfung des eingegebenen Entscheidungsbaums ist
- **dtSolutionTree.py** : Beinhaltet Funktionen sowohl für den Aufbau des Lösungsbaums als auch für die Überprüfung des eingegebenen Baums
- **dtDataSampling.py**: Python-Skript, welches von der Wrapperklasse in WrapperDataSampler.ts aufgerufen wird und verantwortlich für die Erzeugung neuer Datensätze ist
- **WrapperDataSampler.ts**: Wrapperklasse für das Python-Skript zur Datengenerierung

- **WrapperValidateDecisionTree.ts**: Wrapperklasse für das Python-Skript zur Validierung des Entscheidungsbaums

Im Verzeichnis: src\Tasks\DecisionTrees\datasets

- **spiel.csv**: Ursprünglicher Datensatz, der als Basis für die Generierung eines neuen Datensatzes dient
- **genertaed_dataset.csv**: Neu generierter Datensatz im CSV-Format, der für die Berechnung des Lösungsbaums verwendet wird
- **generated_dataset.json**: Neu generierter Datensatz im JSON-Format, der für die Anzeige auf der Benutzeroberfläche im Tabellenformat dient

Erläuterung: DecisionTree.json
Festgelegte Schnittstellen
Es wurden die Schnittstellen 'generatedSampledData', 'generateGraph' und 'decisionTreeValidator' implementiert:

- **'generatedSampledData'**: Ermöglicht die Übergabe der neu generierten Daten
- **'generateGraph'**: Als Schnittstelle, um den Aufgabentyp zu erstellen
- **'decisionTreeValidator'**: Ermöglicht den Empfang des vom Studierenden eingegebenen Entscheidungsbaums für die weitere Verarbeitung

```
"API": [
        {
            "task": "decisiontree",
            "name": "generateSampledData",
            "httpMethod": "post",
            "params": { "parameters": "object" }
        },
        {
            "task": "decisiontree",
            "name": "generateGraph",
            "httpMethod": "post",
            "params": { "parameters": "object" }
        },
        {
            "task": "decisiontree",
            "name": "decisionTreeValidator",
            "httpMethod": "post",
            "params": { "parameters": "object" }
        }
    ],
  ...
```

Hinzugefügte Komponenten für den Aufgabentyp
Für den Aufgabentyp wurden die Komponenten 'hints', 'TaskConfiguration', 'Output' und DecisionTree integriert:

- **'hints'**: Die Verwendung von 'hints' ermöglicht die Implementierung von Hilfestellungen für den Studierenden, um während der Aufgabenbearbeitung Hilfestellungen anzuzeigen
- **'TaskConfiguration'**: Mithilfe von 'TaskConfiguration' wurde sowohl der "Generieren!"-Button implementiert, um neue Aufgaben zu generieren, als auch eine Aufgabenbeschreibung mit der Bedienungsanleitung des Entscheidungsbaums hinzugefügt
- **'Output'**: Dient dazu, die generierten Daten in Tabellenform auf der Benutzeroberfläche anzuzeigen
- **'DecisionTree'**: Die Komponente zeigt dem Studierenden den interaktiven Graphen an

Erläuterung: dtBuildDecisionTree.py und dtSolutionTree.py
Da in der Datei dtBuildDecisionTree.py Funktionen aus dtSolutionTree.py aufgerufen werden, wird hier ein Überblick darüber gegeben, wie diese Funktionen aufgerufen werden und welche Aktionen dabei durchgeführt werden. In dieser Datei werden spezifische Funktionen aufgerufen, die dazu dienen, den Lösungsentscheidungsbaum aufzubauen und ihn für die Überprüfung der Werte vorzubereiten. Die Validierungsfunktion für den eingegebenen Entscheidungsbaum des Studierenden wird hier aufgerufen und das Ergebnis der Überprüfung wird als JSON-Format umgewandelt und als Rückgabewert an die Wrapperklasse in der Datei WarapperValidateDecisionTree.py zurückgegeben.

Zu Beginn wird der generierte Datensatz aus der CSV-Datei geladen, um daraus das Zielmerkmal, dessen Klassen und die weiteren Merkmale zur Berechnung mit dem ID3-Algorithmus zu filtern. Sobald die Wrapperklasse aus der WrapperValidateDecisionTree.py Listen mit den enthaltenen Knoten und Kanten aus dem eingegebenen Entscheidungsbaum übergibt, werden diese Listen für die Überprüfung vorbereitet, sodass sie die benötigte Struktur aufweisen. Das bedeutet, dass jeder Knoten in der Liste der Knoten einem Tupel entspricht, was auch für alle Kanten in der Liste der Kanten gilt. Sobald dies umgesetzt wurde, wird die Funktion start_build_decisiontree() aufgerufen und die vorbereiteten Listen übergeben.

In der Funktion start_build_decisiontree() wird zunächst der Lösungsentscheidungsbaum mithilfe der Funktion build_decision_tree() berechnet. Der Rückgabewert dieser Funktion entspricht einer Baumstruktur, die alle berechneten Werte enthält.

```
# Beispiel: Ausgabe des Lösungsbaums mit den berechneten Werten
{
  "Aussicht": {
    "entropies": {
      "Aussicht = sonnig": 0.87086,
      "Aussicht = regnerisch": 0.97987,
      "Aussicht = bewölkt": 0.0
    },
    "infogain": {
      "Aussicht": 0.32744
    },
    "sonnig": {
      "Luftfeuchtigkeit": {
        "entropies": {
          "Luftfeuchtigkeit = hoch": 0.0,
          "Luftfeuchtigkeit = normal": 0.0
```

```
        },
        "infogain": {
          "Luftfeuchtigkeit": 0.87086
        },
        "hoch": "nein",
        "normal": "ja"
      }
    },
    "regnerisch": {
      "Wind": {
        "entropies": {
          "Wind = nein": 0.0,
          "Wind = ja": 0.0
        },
        "infogain": {
          "Wind": 0.97987
        },
        "nein": "ja",
        "ja": "nein"
      }
    },
    "bewölkt": "ja"
  }
}
```

Aus diesem Rückgabewert werden die Baumstruktur (decisiontree_structure()) und die berechneten Werte für Informationsgewinn und Entropie abstrahiert (extract_entropy_and_infogain()).

```
# Beispiel: Ausgabe der Struktur des Lösungsbaums
{
  "Aussicht": {
    "sonnig": {
      "Luftfeuchtigkeit": {
        "hoch": "nein",
        "normal": "ja"
      }
    },
    "regnerisch": {
      "Wind": {
        "nein": "ja",
        "ja": "nein"
      }
    },
    "bewölkt": "ja"
  }
}

# Beispiel: Ausgabe des Dictionaries mit den Informationsgewinnen und Entropien
{'entropies': {'Aussicht = sonnig1': {'value': 0.87086, 'level': 1}, 'Aussicht =
regnerisch1': {'value': 0.97987, 'level': 1}, 'Aussicht = bewölkt1': {'value': 0.0,
```

```
'level': 1}, 'Luftfeuchtigkeit = hoch1': {'value': 0.0, 'level': 3}, 'Luftfeuchtigkeit
= normal1': {'value': 0.0, 'level': 3}, 'Wind = nein1': {'value': 0.0, 'level': 3},
'Wind = ja1': {'value': 0.0, 'level': 3}}, 'infogain': {'Aussicht1': {'value':
0.32744, 'level': 0}, 'Luftfeuchtigkeit1': {'value': 0.87086, 'level': 2}, 'Wind1':
{'value': 0.97987, 'level': 2}}}
```

Sobald dies abgeschlossen ist, wird eine Liste erstellt (build_decision_tree_list()), die den vollständigen Baum mit den dazugehörigen Werten darstellen soll. Dabei werden einzelne Tupel erzeugt, die entweder einen Knoten oder eine Kante repräsentieren sollen. Die Bezeichnungen der Kanten und Knoten werden hinzugefügt, ebenso wie die Ebene, auf der sie der Knoten bzw. die Kante befindet. Anschließend werden die übergeordneten Knoten und Kanten hinzugefügt (add_parent_label() und add_child_node()).

```
# Beispiel: Ausgabe der gesamten Liste des Lösungsbaums
[(1, 1, 'sonnig', 'edge', 0, 'Aussicht', 2, 'Luftfeuchtigkeit'),
(3, 3, 'hoch', 'edge', 2, 'Luftfeuchtigkeit', 4, 'nein'),
(5, 3, 'normal', 'edge', 2, 'Luftfeuchtigkeit', 6, 'ja'),
(7, 1, 'regnerisch', 'edge', 0, 'Aussicht', 8, 'Wind'),
(9, 3, 'nein', 'edge', 8, 'Wind', 10, 'ja'),
(11, 3, 'ja', 'edge', 8, 'Wind', 12, 'nein'),
(13, 1, 'bewölkt', 'edge', 0, 'Aussicht', 14, 'ja'),
(0, 0, 'Aussicht', 'node', None, 'None'),
(2, 2, 'Luftfeuchtigkeit', 'node', 1, 'sonnig'),
(4, 4, 'nein', 'node', 3, 'hoch'),
(6, 4, 'ja', 'node', 5, 'normal'),
(8, 2, 'Wind', 'node', 7, 'regnerisch'),
(10, 4, 'ja', 'node', 9, 'nein'),
(12, 4, 'nein', 'node', 11, 'ja'),
(14, 2, 'ja', 'node', 13, 'bewölkt')]
```

Danach wird die Liste in separate Listen für Knoten und Kanten aufgeteilt, um eine bessere und gezieltere Handhabung bei der Überprüfung zu ermöglichen. Zu jeder Liste werden entsprechend die Entropien und Informationsgewinne hinzugefügt (add_entropy() und add_infogain()).

Zum Abschluss wird die Liste in die korrekte Struktur gebracht (prepare_lists()), um sie bei der Überprüfung verwenden zu können. Dies bedeutet, dass die Tupel in der Liste wie folgt strukturiert sind:

- **Tupel in der Lösungs-Knoten-Liste**: (KEY, LEVEL, LABEL, PARENTID, PARENTLABEL, INFOGAIN)
- **Tupel in der Lösungs-Kanten-Liste**: (KEY, LEVEL, LABEL, PARENTID, PARENTLABEL, CHILDID, CHILDLABEL, ENTROPY)

Die Listen solution_nodes und solution_edges dienen als Referenzwerte für die Überprüfung, anhand derer der eingegebene Entscheidungsbaum validiert wird.

```
# Beispiel: solution_nodes Liste
[(0, 0, 'Aussicht', None, 'None', 0.32744),
```

```
(2, 2, 'Luftfeuchtigkeit', 1, 'sonnig', 0.87086),
(4, 4, 'nein', 3, 'hoch', 0),
(6, 4, 'ja', 5, 'normal', 0),
(8, 2, 'Wind', 7, 'regnerisch', 0.97987),
(10, 4, 'ja', 9, 'nein', 0),
(12, 4, 'nein', 11, 'ja', 0),
(14, 2, 'ja', 13, 'bewölkt', 0)]

# Beispiel: solution_edges Liste
[(1, 1, 'sonnig', 0, 'Aussicht', 2, 'Luftfeuchtigkeit', 0.87086),
(3, 3, 'hoch', 2, 'Luftfeuchtigkeit', 4, 'nein', 0.0),
(5, 3, 'normal', 2, 'Luftfeuchtigkeit', 6, 'ja', 0.0),
(7, 1, 'regnerisch', 0, 'Aussicht', 8, 'Wind', 0.97987),
(9, 3, 'nein', 8, 'Wind', 10, 'ja', 0.0),
(11, 3, 'ja', 8, 'Wind', 12, 'nein', 0.0),
(13, 1, 'bewölkt', 0, 'Aussicht', 14, 'ja', 0.0)]
```

Die Funktion validateGraph() vergleicht den vom Studierenden eingegebenen Entscheidungsbaum mit dem berechneten Lösungsbaum. Hierbei werden die Listen der Lösungen (solution_nodes und solution_edges) sowie die erhaltenen Werte aus dem erstellten Entscheidungsbaum (received_nodes und received_edges) übergeben. Als Rückgabewert werden zwei Listen (result_nodes und result_edges) zurückgegeben, die die Auswertungen für jeden eingegebenen Knoten und jede eingegebene Kante enthalten. Hierbei werden lediglich die IDs gespeichert und der Wert 'true', wenn die Eingabe des Studierenden korrekt war, bzw. 'false', wenn die Eingabe falsch war.

```
# Beispiel: result_nodes Liste
[{'key': '1', 'result': False},
{'key': 'key_xlXFsdt', 'result': False},
{'key': 'key_soGlGdt', 'result': False},
{'key': 'key_VEQhpdt', 'result': False},
{'key': 'key_B2MDrdt', 'result': False},
{'key': 'key_rEtVGdt', 'result': False},
{'key': 'key_NiSx0dt', 'result': False},
{'key': 'key_dDTWldt', 'result': True},
{'key': 'key_OmwAkdt', 'result': True},
{'key': 'key_Gyegzdt', 'result': False},
{'key': 'key_xl4ocdt', 'result': False}]

# Beispiel: result_edges Liste
 [{'key': 'geid_243_0', 'result': False},
 {'key': 'geid_243_1', 'result': False},
 {'key': 'geid_243_2', 'result': False},
 {'key': 'geid_243_3', 'result': False},
 {'key': 'geid_243_4', 'result': False},
 {'key': 'geid_243_5', 'result': False},
 {'key': 'geid_243_6', 'result': False},
 {'key': 'geid_243_7', 'result': False},
 {'key': 'geid_243_8', 'result': False},
 {'key': 'geid_243_9', 'result': False}]
```

Abschließend wird überprüft, ob der vom Studierenden erstellte Entscheidungsbaum korrekt und vollständig ist. Im Fall, dass der Entscheidungsbaum vollständig richtig aufgebaut wurde, wird der Status von 'finished' auf 'true' gesetzt. Andernfalls bleibt der Status auf 'false'. Die Listen 'result_nodes' und 'result_edges', der Status von 'finished' sowie die berechneten Lösungswerte im Dictionary 'entropy_infogain_dict' werden in das JSON-Format umgewandelt und als Rückgabewert an die Wrapperklasse gesendet.

```
import sys
import ast
import json
import pandas as pd
import numpy as np
from dtSolutionTree import *

data_path_csv = 'src/Tasks/DecisionTrees/datasets/generated_dataset.csv'
dataframe = pd.read_csv(data_path_csv)

target_attribute = dataframe.columns[-1]
classes = dataframe[target_attribute].unique()
attributes = dataframe.columns[:-1]

def start_build_decisiontree(received_nodes, received_edges, classes):

    dt = build_decision_tree(dataframe, attributes, target_attribute, 0,
received_nodes)
    decision_tree_structure_dict = decisiontree_structure(dt)
    entropy_infogain_dict = extract_entropy_and_infogain(dt)

    tree_list = []
    build_decision_tree_list(decision_tree_structure_dict, tree_list=tree_list)

    tree_list = add_parent_label(tree_list)
    tree_list = add_child_node(tree_list)

    nodes_list = [item for item in tree_list if item[3] == 'node']
    edges_list = [item for item in tree_list if item[3] == 'edge']

    edges_list = add_entropy(edges_list, entropy_infogain_dict)
    nodes_list = add_infogain(nodes_list, entropy_infogain_dict)

    solutions_nodes, solutions_edges = prepare_lists(nodes_list, edges_list)
    results_nodes, results_edges = validateGraph(solutions_nodes, solutions_edges,
received_nodes, received_edges, classes)

    correctNodesCount = 0
    correctEdgesCount = 0

    for r in results_nodes:
        if r['result'] == True:
            correctNodesCount += 1
```

```
    for r in results_edges:
        if r['result'] == True:
            correctEdgesCount += 1

    if(len(solutions_nodes) == correctNodesCount and len(solutions_edges) ==
correctEdgesCount and len(solutions_nodes) == len(received_nodes) and
len(solutions_edges) == len(received_edges)):
        result = { 'nodes': results_nodes, 'edges': results_edges, 'finished': True,
'entropies': entropy_infogain_dict }
    else:
        result = { 'nodes': results_nodes, 'edges': results_edges, 'finished': False,
'entropies': entropy_infogain_dict }

    return (json.dumps(result))

# Erhalt der Nodes- und Edges-Liste aus WrapperValidateDecisionTree.ts
received_nodes_list, received_edges_list = sys.argv[1], sys.argv[2] if len(sys.argv)
<= 3 else (None, None)

modified_nodes_list_1 = ast.literal_eval(received_nodes_list)
modified_nodes_list_2 = [tuple(inner) for inner in modified_nodes_list_1]
current_nodes_list = [(x[0], x[1], 0 if x[2] is None else x[2], x[3], x[4]) for x in
modified_nodes_list_2]

modified_nodes_list = ast.literal_eval(received_edges_list)
current_edges_list = [tuple(inner) for inner in modified_nodes_list]

print(start_build_decisiontree(current_nodes_list, current_edges_list, classes))
```

Erläuterung: dtDataSampling.py
In der dtDataSampling.py wird die Datengenerierung implementiert. Zunächst wird der originale Datensatz geladen, um neue Stichproben zu generieren. Der Pfad, unter dem der neu generierte Datensatz gespeichert werden soll, wird mit 'src/Tasks/DecisionTrees/datasets/generated_dataset.csv' angegeben.

bootstrap_sample() und generate_data_point() - Funktion
Diese Funktionen dienen dazu, Bootstrap-Samples des Datensatzes mit bootstrap_sample() zu erstellen und Datenpunkte basierend auf dem Entscheidungsbaum mit generate_data_point() zu generieren.

check_generated_tree() - Funktion
Diese Funktion überprüft, ob die Struktur des Baumes aus dem generierten Datensatz der originalen Baumstruktur entspricht. Dies stellt sicher, dass bei unterschiedlichen Strukturen der Datensatz erneut generiert wird, bis er die gleiche Struktur wie der Originalbaum enthält.

sample_data() - Funktion
Diese Funktion ruft die Datengenerierung auf. Die while-Schleife sorgt dafür, dass der neue Datensatz generiert wird, solange er nicht der originalen Baumstruktur des Originalbaums entspricht. Mit 'num_samples' wird die Anzahl der Batches festgelegt, wie oft Daten generiert werden sollen.

Die Größe des Batches entspricht der Größe des Ursprungsdatensatzes. Der Aufruf von pd.concat(...) erstellt ein neues DataFrame aus der Pandas-Bibliothek, welches die neu generierten Daten strukturiert. Die Funktion concat() wird verwendet, um mehrere Batches zusammenzuführen und einen vollständig neuen Datensatz zu erhalten.

Nach der Generierung des Datensatzes wird dieser sowohl im JSON-Format gespeichert, um ihn als Tabelle auf der Benutzeroberfläche anzuzeigen, als auch als CSV-Datei, um die Berechnung der Baumstruktur aus dem generierten Datensatz zu ermöglichen.

```
import pandas as pd
import numpy as np
from dtSolutionTree import *

data_path_csv = 'src/Tasks/DecisionTrees/datasets/spiel.csv' # Originaler Datensatz
generated_data_path_csv = 'src/Tasks/DecisionTrees/datasets/generated_dataset.csv' #
Generierter Datensatz

data = pd.read_csv(data_path_csv)
df = pd.DataFrame(data)

# Beispiel: Lösungsbaum-Struktur von spiel.csv
# decision_tree = {'Aussicht': {'sonnig': {'Luftfeuchtigkeit': {'hoch': 'nein',
'normal': 'ja'}},
#                               'bewölkt': 'ja',
#                               'regnerisch': {'Wind': {'nein': 'ja', 'ja': 'nein'}}}}

decision_tree_original = get_decisiontree_structure(data_path_csv)

# Prüfung der Baumstruktur aus dem generierten Datensatz
def check_generated_tree(original_tree, generated_tree):
    original_json = json.loads(json.dumps(original_tree, sort_keys=True))
    generated_json = json.loads(json.dumps(generated_tree, sort_keys=True))
    return original_json == generated_json

# Bootstrap-Sampling-Funktion
def bootstrap_sample(data):
    # samplen der Datensätze in gleicher größe wie die original Daten
    sample = data.sample(n=len(data), replace=True)
    return sample

# Funktion zum Generieren von Datenpunkten basierend auf dem Entscheidungsbaum
def generate_data_point(tree):
    current_node = tree
    while isinstance(current_node, dict):
        feature = list(current_node.keys())[0]
        values = list(current_node[feature].keys())
        selected_value = np.random.choice(values)
        current_node = current_node[feature][selected_value]
    return current_node
```

```
def sample_data():
    while True:
        num_samples = 5
        data = pd.concat([bootstrap_sample(df.apply(generate_data_point, axis=1)) for
_ in range(num_samples)], ignore_index=True)

        # als JSON speichern für die Ausgabe als Tabelle

data.to_json('src/Tasks/DecisionTrees/datasets/generated_dataset.json',orient='records
',lines=True)

        # als CSV speichern für die Berechnung des Entscheidungsbaums
        data.to_csv('src/Tasks/DecisionTrees/datasets/generated_dataset.csv',
index=False)

        # Entscheidungsbaum für die generierten Daten erstellen
        decision_tree_generated = get_decisiontree_structure(generated_data_path_csv)

        # Überprüfen, ob die generierten Daten den originalen Daten die gleiche
Baumstruktur entsprechen
        if check_generated_tree(decision_tree_original, decision_tree_generated):
            print("Die generierter Datensatz entspricht der Struktur des originalen
Entscheidungsbaum.")
            break

sample_data()
```

Erläuterung: WrapperDataSampler.ts
Die Datei WrapperDataSampler.ts enthält die Wrapperklasse, die in TypeScript geschrieben ist und dazu dient, die Ausführung des Python-Skripts dtDataSampling.py zu erleichtern.

runPythonScript() - Funktion
Diese Funktion ermöglicht die Ausführung eines Python-Skripts mit übergebenen Argumenten. Hierbei wird die 'exec()'-Funktion aus dem 'child_process' Modul verwendet. Der Shell-Befehl für die Skriptausführung wird erstellt und bei erfolgreicher Ausführung wird die Standardausgabe zurückgegeben. Im Fehlerfall werden entsprechende Fehlermeldungen oder Standardfehlerausgaben behandelt und zurückgegeben.

readJsonFile() - Funktion
Diese Funktion liest eine JSON-Datei und gibt die geparsten Daten als Array von Objekten zurück. Sie filtert leere Zeilen und behandelt Lesefehler sowie Parsing-Ausnahmen. Die geparsten JSON-Daten werden als Promise zurückgegeben.

DataSamplingGenerator() - Funktion
Dies ist die Hauptfunktion, welche das Python-Skript dtDataSampling.py ausführt, die JSON-Datei einliest und eine Tabelle aus den Daten erstellt. Dabei werden mögliche Fehler bei der Ausführung des Skripts oder beim Lesen der JSON-Datei behandelt.

Erläuterung: WrapperValidateDecisionTree.ts
Die Datei WrapperValidateDecisionTree.ts enthält die Wrapperklasse, die in TypeScript geschrieben

ist, um die Ausführung des Python-Skripts dtBuildDecisonTree.py zu erleichtern.

runPythonScript() - Funktion
Diese Funktion ermöglicht die Ausführung eines Python-Skripts über die Shell und nutzt die 'exec()'-Funktion aus dem 'child_process' Modul. Fehler, die während der Ausführung auftreten, werden über ein Promise behandelt, um eine asynchrone Ausführung zu ermöglichen.

createNodesList() und createEdgesList() - Funktion
Diese Funktion konvertiert eine Liste von Knoten und Kanten aus dem Graphen in ein Tupel-Format mit einer festgelegten Struktur, die für die spätere Überprüfung der Werte relevant ist.

- **Die Struktur eines Knoten-Tupels**: (KEY, LEVEL, LABEL, PARENTEDGE, INFOGAIN)
- **Die Struktur eines Kanten-Tupels**: (KEY, LEVEL, LABEL, PARENTKEY, PARENTNODELABEL, CHILDKEY, CHILDNODELABEL, ENTROPY)

DecisionTreeValidator() - Funktion
Dies ist die Hauptfunktion. Aus dem übergebenen Entscheidungsbaum an diese Funktion werden die Knoten- und Kantenlisten erzeugt, die für die Überprüfung übergeben werden sollen. Die Daten werden für die Übergabe so vorbereitet, dass sie als JSON-Daten in Zeichenketten umgewandelt werden. Hierbei werden die Zeichenketten mit Hochkommas geparst, um eine korrekte Übertragung der Daten als Shell-Befehl zu ermöglichen.

Das Python-Skript dtBuildDecisionTree.py wird mit den übergebenen Listen als Argumente ausgeführt und erhält die Werte der Auswertung des Entscheidungsbaums zurück. Diese werden anschließend als JSON-Format geparst und als Rückgabewert an das Frontend CARPET weitergegeben.

4.4.2. Implementierung in CARPET

Bei der Frontend-Umsetzung in CARPET wurde sich an dem Gozintographen Aufgabentyp orientiert. Da dieser Aufgabentyp bereits eine Graphenimplementierung enthielt, wurden die entsprechenden Komponenten modifiziert und erweitert, um eine visuelle Darstellung des Entscheidungsbaums zu ermöglichen.

In CARPET wurden folgende Dateien hinzugefügt bzw. geändert:

- Neue Komponente für die Erstellung eines interaktiven Entscheidungsbaums: **DecisionTree.vue** (im Verzeichnis: src\components\taskComponents\decisiontree\DecisionTree)
- **Änderungen in Helperfunctions.ts**: Weitere Funktionen wurden hinzugefügt als Unterstützung für die Umsetzung des interaktiven Entscheidungsbaums
- **Änderungen in taskGraph.ts**: Eine Schnittstelle wurde implementiert, um den bearbeiteten Graphen des Studierenden an das Backend (ALADIN) zu übertragen

Erläuterung: DecisionTree.vue
Bei der Umsetzung der neuen Komponenten DecisionTree.vue wurde die sich an der bereits existierenden DOTGraph-Komponente orientiert, die bereits im Gozintographen-Aufgabentyp verwendet wurde. Die DOTGraph-Komponente stellte eine Grundlage dar, um eine ähnliche Visualisierung eines Graphen für einen interaktiven Entscheidungsbaum zu implementieren. Dadurch konnte

bereits bestehender Code wiederverwendet und erweitert werden.

Graphology
Als Erweiterung der Komponente wurde die Bibliothek "Graphology" verwendet. Die Bibliothek bietet eine Sammlung an Funktionen, um die Erstellung und Verwaltung von Graphen zu erleichtern. Dadurch war es möglich, für den Entscheidungsbaum eine flexible Struktur zu schaffen. Die Anwendungen der Funktionen und Event-Handler ermöglichten die einfache Durchführung von Manipulationen am Graphen, wie beispielsweise das Hinzufügen oder Entfernen von Knoten und Kanten.

Neuen Graphen anlegen und IDs speichern
Zu Beginn wird ein neuer Graph erstellt und ein Array zur Speicherung von IDs für jeden Knoten und für jede Kante. Dadurch wird die Überprüfung auf Vorhandensein einer ID ermöglicht, wenn eine neue ID generiert wird.

```
let graph = new Graph();
let ids: Array<String> = [];
```

Funktion: idGenerator()
Diese Funktion generiert automatisch eindeutige Zeichenketten als IDs für Knoten und Kanten im Graphen. Dabei wird überprüft, ob die generierte ID bereits vorhanden ist. Diese werden in das zuvor angelegte Array gespeichert.

```
const idGenerator = () => {
  let s = "";
  do {
    s = "key_" + createRandomString() + "dt";
  } while (ids.includes(s));
  ids.push(s);
  return s;
}
```

Wurzelknoten erstellen
Diese Funktion hat den Zweck, den ersten Knoten (Wurzelknoten) zu erstellen. Dabei werden der erstellte Graph und ein Schlüsselwert für den Wurzelknoten übergeben. Es wurde festgelegt, dass der Wurzelknoten im Graphen immer den Schlüsselwert '1' hat, um eine eindeutige Identifikation zu gewährleisten. Schlüsselwerte (Keys) sind erforderlich für die Nutzung der Funktionen aus der Graphology-Bibliothek, um eindeutig Knoten und Kanten anzusteuern.

```
function createRootNode(graph, rootNodeKey) {
  graph.addNode(rootNodeKey, {
    id: idGenerator(),
    label: "Wurzelknoten\n(Info)",
    name: "Wurzelknoten",
    infogain: -1.0,
    level: 0,
    parentedge: "None"
```

```
  });
}
```

Event: Anklicken des "Generieren!"-Buttons
Nach dem Klick auf den 'Generieren!'-Button wird überprüft, ob bereits ein Wurzelknoten vorhanden ist. Falls nicht, wird ein neuer Wurzelknoten erzeugt. Wenn bereits ein Knoten existiert, wird der aktuelle Entscheidungsbaum gelöscht und ein neuer Wurzelknoten erstellt. Dadurch wird gewährleistet, dass beim Generieren eines neuen Datensatzes der Entscheidungsbaum entsprechend zurückgesetzt wird, um die neue Aufgabe von vorne beginnen zu können.

```
setTimeout(() => {
  var generateButton = document.querySelector('.button');
  generateButton.addEventListener('click', function() {
    const rootNodeKey = '1';
    let nodeExists = graph.hasNode(rootNodeKey);
    if (!nodeExists) {
      createRootNode(graph, rootNodeKey);
    } else {
      alert('Neuer Datensatz wurde generiert!');
      graph.clear();
      createRootNode(graph, rootNodeKey);
    }
  });
}, 100);
```

Hinzufügen von Attributwerten: parentedge und childnode
Die Funktion setParentEdge() fügt die eingehenden Kanten zu einem Knoten hinzu. Dies ermöglicht spätere Überprüfungen, um sicherzustellen, dass der Knoten eindeutig zugeordnet werden kann. Gleiches gilt für die Funktion setParentChildNode(). Diese Funktion fügt für jede Kante den Startknoten ('parentnode') und den Zielknoten ('childnode') hinzu.

```
function setParentEdge(key: any) {
  graph.findNode((node, attributes) => {
    if (key == node) {
      graph.findInEdge((edge, attributes, source, target) => {
        if (node == target) {
          graph.setNodeAttribute(node, 'parentedge', attributes['name']);
        }
      });
    }
  });
}

function setParentChildNode(key: any) {
  let nodeName = graph.getNodeAttribute(key, 'name')
  graph.findEdge((edge, attributes, source, target) => {
    if (key == source) {
      graph.setEdgeAttribute(edge , 'parentnode', nodeName)
```

```
    }
  });
  graph.findEdge((edge, attributes, source, target) => {
    if (key == target) {
      graph.setEdgeAttribute(edge , 'childnode', nodeName)
    }
  });
}
```

Farbige Hervorhebung der Knoten und Kanten
Die Funktionen colorNodes() und colorEdges() haben den Zweck, Knoten und Kanten farblich hervorzuheben. Nach der Überprüfung des Entscheidungsbaums im Backend und der Übermittlung der Auswertungen an das Frontend werden zu jeder Kante und zu jedem Knoten entsprechend ihren Werten (true oder false) in den Farben Grün oder Orange eingefärbt. Hierbei wird die Farbe als zusätzliches Attribut entsprechend hinterlegt.

```
function colorNodes(key: any, c: boolean) {
  if (c) { graph.setNodeAttribute(key, 'color', 'green'); }
  else { graph.setNodeAttribute(key, 'color', '#f1ad2d'); }
}

function colorEdges(key: any, c: boolean) {
  if (c) { graph.setEdgeAttribute(key, 'color', 'green'); }
  else { graph.setEdgeAttribute(key, 'color', '#f1ad2d'); }
}
```

Übermittelte Auswertungen der Knoten und Kanten aus dem Backend überprüfen
In der Funktion validateGraph() erfolgt die Überprüfung der Auswertungen, die vom Backend übergeben wurden. Hier werden die Funktionen zum Einfärben der Knoten und Kanten aufgerufen. Zudem wird überprüft, ob der Entscheidungsbaum bzw. der Graph bereits vollständig gelöst wurde. Diese Information wird aus den Lösungswerten der Auswertung ('parsendResult['status']') herausgelesen. Wenn dieser Wert den Wert true besitzt, gilt die Aufgabe als vollständig gelöst und es wird eine Meldung für den Studierenden angezeigt.

```
function validateGraph(parsedResult: any){
  const solutionNodesList = parsedResult['solution_nodes_list'];
  const solutionEdgesList = parsedResult['solution_edges_list'];
  const status = parsedResult['status'];
  for (let i = 0; i < solutionNodesList.length; i++) {
colorNodes(solutionNodesList[i]['key'], solutionNodesList[i]['result']); }
  for (let i = 0; i < solutionEdgesList.length; i++) {
colorEdges(solutionEdgesList[i]['key'], solutionEdgesList[i]['result']); }
  if (status) {
    setTimeout(() => {
      console.log("Finished: ", status);
      alert("Herzlichen Glückwunsch!\nSie haben die Aufgabe vollständig gelöst!");
    }, 200);
  }
```

```
  renderIfGraph();
}
```

Event-Handler bei Änderungen der Knoten und Kanten
Die Event-Handler in dieser Implementierung sind aus der Graphology-Bibliothek und ermöglichen die Reaktion auf Veränderungen im Graphen. Im folgenden Beispiel wird dies bei einer Attributsänderung eines Knotens verdeutlicht, wobei das Prinzip auf andere Event-Handler übertragbar ist. Insbesondere ist der Event-Handler von Bedeutung, wenn die Attributwerte 'label' (Knotenbeschriftung), 'parentedge' (eingehende Kante des Knotens) und 'infogain' (Informationsgewinn für den Knoten) verändert werden. Daher erfolgt zunächst eine Überprüfung, ob das geänderte Attribut eines dieser Werte ist. Falls nicht, wird die Funktion vorzeitig verlassen. Andernfalls wird der aktuelle Zustand des Graphen an das Backend gesendet, welches den vom Studierenden eingegebenen Graphen überprüft. Nachdem der Graph übermittelt wurde, wird die Auswertung als JSON unter 'parsedResult' gespeichert und für weitere Verarbeitung bereitgestellt.

Die folgenden Event-Handler wurden für diese Anwendung genutzt:

- **'nodeAttributesUpdated'**: bei Änderungen der Attributwerte 'label', 'parentedge' und 'infogain' eines Knotens
- **'edgeAdded'**: bei Hinzufügen einer Kante
- **'edgeAttributedUpdated'**: bei Änderungen der Attributwerte 'label', 'entropy', 'parentnode' und 'childnode' einer Kante
- **'nodeDropped'**: bei Entfernen eines Knotens

```
graph.on('nodeAttributesUpdated', function({key, type, attributes, name}) {
  if (name !== 'label' && name != 'parentedge' && name != 'infogain')
    return;

  unref(storeObject).store.dispatch('fetchSolution', graph.export())
    .then(result => {
      const parsedResult = JSON.parse(result);
      validateGraph(parsedResult);
      console.log("solution: ", parsedResult)
      renderIfGraph();
    });
});
```

Überprüfung der Eingabewerte für den Informationsgewinn und die Entropie
Die Funktion isValidInputEntropy() und isValidInputInfo() wurden implementiert, um die Eingabewerte für die Entropie an den Kanten und den Informationsgewinn an den Knoten zu validieren. Ziel ist es sicherzustellen, dass die übergebenen Werte vom Typ Float sind, um mögliche Komplikationen bei späteren Überprüfungen zu vermeiden. Bei der Validierung des Informationsgewinns wird darüber hinaus geprüft, ob die Eingabe entweder vom Typ Float oder vom Typ String mit dem spezifischen Stringwert 'decision'. Nur der Stringwert 'decision' ist zulässig, um Knoten zu identifizieren, die Klassen repräsentieren.

```
  function isValidInputEntropy(input: string): boolean {
    const floatValue = parseFloat(input);
    if (!isNaN(floatValue)) {
      return true;
    }
    else {
      return false;
    }
  }

  function isValidInputInfo(input: string | number): string | number | null {
  if (typeof input === 'number') {
    return parseFloat(input.toFixed(5));
  } else if (typeof input === 'string') {
    if (input.toLowerCase() === 'decision') {
      return 'decision';
    }
    const parsedFloat = parseFloat(input);
    return isNaN(parsedFloat) ? null : parseFloat(parsedFloat.toFixed(5));
  }
  return null;
}
```

Interaktionen mit dem Graphen

Um den Studierenden eine interaktive Bedienung des Entscheidungsbaums bzw. des SVG-Graphen zu ermöglichen, wurden Event-Handler aus der D3.js-Bibliothek für die Knoten und Kanten implementiert. Dabei werden sowohl Links- als auch Rechtsklicks beim Anklicken eines Knotens abgefangen. Bei Kanten wird ausschließlich auf Rechtsklick reagiert.

```
setTimeout(() => {
    d3.selectAll('svg .node').on('click', null);
    d3.selectAll('svg .node').on('contextmenu', null);
    d3.selectAll('svg .edge').on('contextmenu', null);
    ...
```

Rechtsklick auf eine Kante - Kantenbeschriftung und Entropie ändern

Wenn eine Kante durch Rechtsklick ausgewählt wird, öffnet sich ein Prompt zur Eingabe der Kantenbeschriftung. Wenn die Eingabe für die Kantenbeschriftung leer ist, wird der Prompt geschlossen, ohne Änderungen an der Kante vorzunehmen. Falls jedoch eine Kantenbeschriftung eingegeben wurde, erscheint ein weiterer Prompt zur Eingabe der Entropie an der Kante. Die eingegebene Entropie wird auf seine Gültigkeit überprüft. Dies wird so lange wiederholt, bis ein gültiger Wert eingegeben wurde. Anschließend werden diese Werte als Attribute für die Kante gespeichert. Das Attribut 'label' fungiert als zusammengesetzter String, der die Kantenbeschriftung und die Entropie enthält und im Graphen angezeigt wird. Das Attribut 'name' enthält ausschließlich die Kantenbeschriftung, was für spätere Überprüfungen erforderlich ist.

```
// Kantenbeschriftung und Entropie ändern
```

```
d3.selectAll('svg .edge').on('contextmenu', (e) => {
      let edgeName = prompt("Kantenbeschriftung hinzufügen:", "");
      let edgeEntropyInput;
      edgeName = normalizeString(edgeName);

      if (edgeName == null || edgeName == "") {
        return;
      } else {
        do {
          edgeEntropyInput = prompt("Entropy hinzufügen:", "");
        } while (!isValidInputEntropy(edgeEntropyInput));
      }

      const edgeLabel = edgeName + "\n" + "(" + edgeEntropyInput + ")"
      const edgeId = e.target.parentNode.id;
      const filteredEdge = graph.filterEdges((n, a) => a.id == edgeId);

      if (filteredEdge.length <= 0 || edgeLabel == null || edgeLabel == "")
        return;

      const edgeEntropy = parseFloat(edgeEntropyInput);
      graph.setEdgeAttribute(filteredEdge[0], 'label', edgeLabel);
      graph.setEdgeAttribute(filteredEdge[0], 'name', edgeName);
      graph.setEdgeAttribute(filteredEdge[0], 'entropy', edgeEntropy);

      setTimeout(() => {
        let parentEdge;
        graph.findEdge((edge, attributes, source, target) => {
          if(filteredEdge[0] == edge){
            parentEdge = target;
          }
        });
        setParentEdge(parentEdge);
      }, 300);
    });
    ...
```

Rechtsklick auf einen Knoten - Knotenbeschriftung und Informationsgewinnwert ändern
Bei einem Rechtsklick auf einen Knoten öffnet sich ein Prompt, welcher zur Eingabe der Knotenbeschriftung auffordert. Wenn die Eingabe leer ist, wird die Funktion vorzeitig beendet und es erfolgt keine Änderungen am Knoten. Andernfalls erscheint ein weiterer Prompt, der nach einer Eingabe des Informationsgewinns für den Knoten auffordert. Der eingegebene Wert wird überprüft. Bei einer ungültigen Eingabe wird erneut der Prompt angezeigt, bis ein gültiger Wert eingegeben wird. Anschließend werden die eingegebenen Werte als Attributwerte des Knotens gespeichert. Das Attribut 'label' fungiert als zusammengesetzter String, der die Beschriftung des Knotens und den Informationsgewinn enthält und für die Anzeige im Graphen verwendet wird. Das Attribut 'name' wird für die Knotenbeschriftung verwendet und ist für die späteren Überprüfungen relevant.

```
// Knotenbeschriftung und Informationsgewinn ändern
```

```
d3.selectAll('svg .node').on('contextmenu', (e) => {
    let nodeName = prompt("Label für Knoten ändern:", "");
    let nodeInfoGainInput: string | number;
    let nodeInfoGain;

    nodeName = normalizeString(nodeName);
    const nodeId = e.target.parentNode.id;
    const node = graph.filterNodes((n, a) => a.id == nodeId);

    if (node.length <= 0 || nodeName == null || nodeName == "") {
    return;
    } else {
    do {
        nodeInfoGainInput = prompt("Informationsgewinn hinzufügen:", "");
        nodeInfoGain = isValidInputInfo(nodeInfoGainInput);
    } while (nodeInfoGain === null);
    }

    graph.setNodeAttribute(node[0], 'name', nodeName);
    graph.setNodeAttribute(node[0], 'infogain', nodeInfoGain);
    graph.setNodeAttribute(node[0], 'label', nodeName + "\n" + "(" + nodeInfoGain +
")");

    setTimeout(() => {
    setParentChildNode(node);
    }, 200);
});
...
```

Linksklick auf einen Knoten - Neuer Knoten hinzufügen und Knoten entfernen
Bei einem Linksklick auf einen Knoten können zwei Funktionalitäten ausgelöst werden:

1. **Linksklick mit Eingabe einer Beschriftung für den Knoten**: Ein neuer Knoten wird unterhalb des angeklickten Knotens hinzugefügt
2. **Linksklick ohne Eingabe einer Beschriftung für den Knoten**: Der angeklickte Knoten wird entfernt

Bei einem Linksklick auf einen Knoten öffnet sich ein Prompt, um die Beschriftung des neu hinzuzufügenden Knotens einzugeben. Dabei wird überprüft, ob eine Eingabe vorhanden ist. Wenn die Eingabe leer ist, wird die Funktion zum Entfernen des angeklickten Knotens ausgeführt. Andernfalls öffnet sich ein weiterer Prompt, in dem der Informationsgewinn eingegeben werden kann. Es folgt die Erstellung eines neuen Knotens mit den eingegebenen Attributwerten sowie die Erzeugung einer neuen Kante, die vom angeklickten Knoten ausgeht und zum neuen hinzugefügten Knoten führt. Für die Kante werden entsprechende Attributwerte hinzugefügt, die zunächst Standardwerte beinhalten.

```
// Neuen Knoten hinzufügen/ Knoten entfernen
d3.selectAll('svg .node').on('click', (e) => {
    let nodeName = prompt("Neuer Knoten:", "");
```

```
    let nodeInfoGainInput: string | number;
    let nodeInfoGain;

    nodeName = normalizeString(nodeName);
    const nodeId = e.target.parentNode.id;
    const node = graph.filterNodes((n, a) => a.id == nodeId);

    if (node.length <= 0)
    return;

    if (nodeName == null || nodeName == "") {
        if (node[0] == '1')
            return;
        else if (graph.degree(node[0]) > 1) {
            alert('Es dürfen nur Blattknoten gelöscht werden!');
            return;
        }
        graph.dropNode(node[0]);
        return;

        } else {
            do {
            nodeInfoGainInput = prompt("Informationsgewinn hinzufügen:", "");
            nodeInfoGain = isValidInputInfo(nodeInfoGainInput);
        } while (nodeInfoGain === null)
    }

    // Neuer Knoten wird erstellt
    let newNode = graph.addNode(idGenerator(), {
    id: idGenerator(),
    label: nodeName + "\n" + "(" + nodeInfoGain + ")",
    name: nodeName,
    infogain: nodeInfoGain,
    level: graph.getNodeAttribute(node[0], 'level') + 2,
    parentedge: "None",
    });

    setTimeout(() => {
    // Neue Kante wird erstellt
    graph.addEdge(node[0], newNode, {
        id: idGenerator(),
        label: "Label\n(Entropie)",
        name: "Label",
        entropy: -1.0,
        parentnode: graph.getNodeAttribute(node[0], 'name'),
        childnode: graph.getNodeAttribute(newNode, 'name'),
        level: graph.getNodeAttribute(node[0], 'level') + 1,
    });
    setParentEdge(newNode);
    }, 200);
});
```

```
}, 500);
```

Neue Funktionen bei HelperFunctions.ts
Um die Generierung von Schlüsselwerten und IDs für Knoten und Kanten zu unterstützen und auch die Normalisierung von Strings bei Eingabewerten zu ermöglichen, wurden entsprechende Hilfsfunktionen implementiert:

createRandomString()
Diese Funktion erzeugt eine zufällige Zeichenkette. Die Länge der Zeichenkette ist standardmäßig auf fünf festgelegt. In der while-Schleife wird ein zufälliges Zeichen aus dem vorherigen Zeichenpool 'characters' ausgewählt und zu der Zeichenkette hinzugefügt. Sobald die Zeichenkette eine Länge von fünf erreicht, wird die als Rückgabewert zurückgegeben.

normalizeString()
Diese Funktion normalisiert einen Zeichenkettenwert mithilfe von regulären Ausdrücken. Standardmäßig ersetzt sie mehrere aufeinanderfolgende Leerzeichen durch ein einzelnes Leerzeichen. Der Parameter 's' ist der Eingabestring, der normalisiert werden soll. Der Parameter 'regex' gibt an, welche Zeichen ersetzt werden sollen.

```
const createRandomString = (length: number = 5) => {
  const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
  let s = "";

  while (s.length < length)
    s += characters.charAt(Math.floor(Math.random() * characters.length));
  return s;
}

const normalizeString = (s: string | null, regex: string = "\\s+") => {
  if (s == null)
    return null;
  return s.replace(new RegExp(regex, "gi"), "_").trim();
}
```

Neue Schnittstelle bei taskGraph.ts
Diese Funktion initiiert eine asynchrone POST-Anfrage an das Backend und empfängt die entsprechenden Antwortdaten. Der Payload der Anfrage setzt sich zusammen aus dem Graphen, welcher vom Studierenden auf der Benutzeroberfläche aufgebaut wurde, dem Typ 'decisiontree', sowie einer Anweisung 'decisionTreeValidator'. Letztere ruft die entsprechende Funktion im Backend auf und übergibt den Graphen zur Überprüfung der enthaltenen Werte. Die Funktion dient entsprechend als Schnittstelle für die Übertagung des eingegbenen Graphen an das Backend.

```
fetchSolution: async ({}, payload: Object) => {
    const result = await axios({
      method: 'POST',
      url: 'http://localhost:8000/api/decisiontree/decisionTreeValidator',
      headers: {
        "Content-Type": "application/json"
```

```
      },
      data: JSON.stringify({ graph: payload, type: 'decisiontree', instruction:
'decisionTreeValidator' })
    });
    return result.data;
  }
```

4.5. Ergebnisse

4.5.1. Aktueller Stand

Die Umsetzung des Aufgabentyps konnte in ALADIN umgesetzt werden, wodurch es für die Studierende nun möglich ist, anhand von generierten Datensätzen einen Entscheidungsbaum zu berechnen und diesen auf interaktive Weise aufzubauen und zu lösen.

Tabelle 3. Übersicht über die erfüllten/nicht erfüllten Anforderungen

Anforderung	**erfüllt/nicht erfüllt**
Zufällige Datensatzgenerierung	erfüllt
Datensatzgenerierung basierend auf vorhandenem Baum	erfüllt
Interaktiver Aufbau des Entscheidungsbaums	erfüllt
Überprüfung des eingegebenen Entscheidungsbaums	erfüllt
Flexible Positionierung der Werte	erfüllt
Berücksichtigung von Mehrfachlösungen	erfüllt
Feedback nach erfolgreicher Lösung	erfüllt
Generierung neuer Aufgaben	erfüllt
Hinweis, um welchen Fehler es sich bei der Eingabe handelt	nicht erfüllt (bzw. nur beim Prototypen umgesetzt)

Sobald der Studierende auf den Aufgabentypen "DecisionTree" auf der Startseite von ALADIN klickt, erscheinen drei Felder. Das linke Feld enthält die Aufgabenstellung sowie Hinweise zur Bearbeitung und Bedienungshinweise für die Benutzung des Entscheidungsbaums. Auf der rechten Seite erscheint der generierte Datensatz, der für die Berechnung verwendet wird. Unterhalb beider Felder befindet sich der Bereich, in dem der Studierende den Entscheidungsbaum erstellen kann.

Erstellung einer Aufgabe
Wenn der Studierende auf den "Generieren"-Button klickt, wird ein Datensatz und ein Wurzelknoten erzeugt. Bei erneutem Klicken auf den Button wird ein neuer Datensatz und ein neuer Wurzelknoten generiert. Dies bedeutet, dass der bisher aufgebaute Entscheidungsbaum entsprechend entfernt wird.

Abbildung 10. Ansicht des Aufgabentyps bei der Erstellung einer neuen Aufgabe

Knoten und Kanten bearbeiten

Durch Rechtsklick auf einen Knoten können sowohl die Beschriftung als auch der Informationsgewinn des Knotens geändert werden. Ein Prompt erscheint für die Eingabe der Beschriftung (Label) des Knotens, gefolgt von einem weiteren Prompt für die Eingabe des Informationsgewinns. Das gleiche Vorgehen gilt auch für die Änderung der Beschriftung und Entropien an den Kannten. Wenn keine Eingabe für die Beschriftung des Knotens erfolgt, wird der Knoten aus dem Baum entfernt, sofern es sich um einen Blattknoten handelt. Falls der Knoten ein Blattknoten ist und eine Klasse eingetragen werden soll, wird beim Prompt für die Eingabe des Informationsgewinns die Bezeichnung "decision" eingetragen, um hervorzuheben, dass es sich um eine Klasse und kein Merkmal handelt.

Abbildung 11. Anzeige eines Prompts für die Bearbeitung eines Knotens

Neue Knoten hinzufügen

Durch Linksklick auf einen Knoten kann ein neuer Knoten direkt unterhalb des ausgewählten Knotens erstellt werden. Ein Prompt wird angezeigt, um die Beschriftung und den Informationsgewinn des neuen Knotens einzugeben. Dabei wird automatisch eine Kante zum neuen Knoten erstellt, der Standardwerte enthält, welche nachträglich geändert werden können.

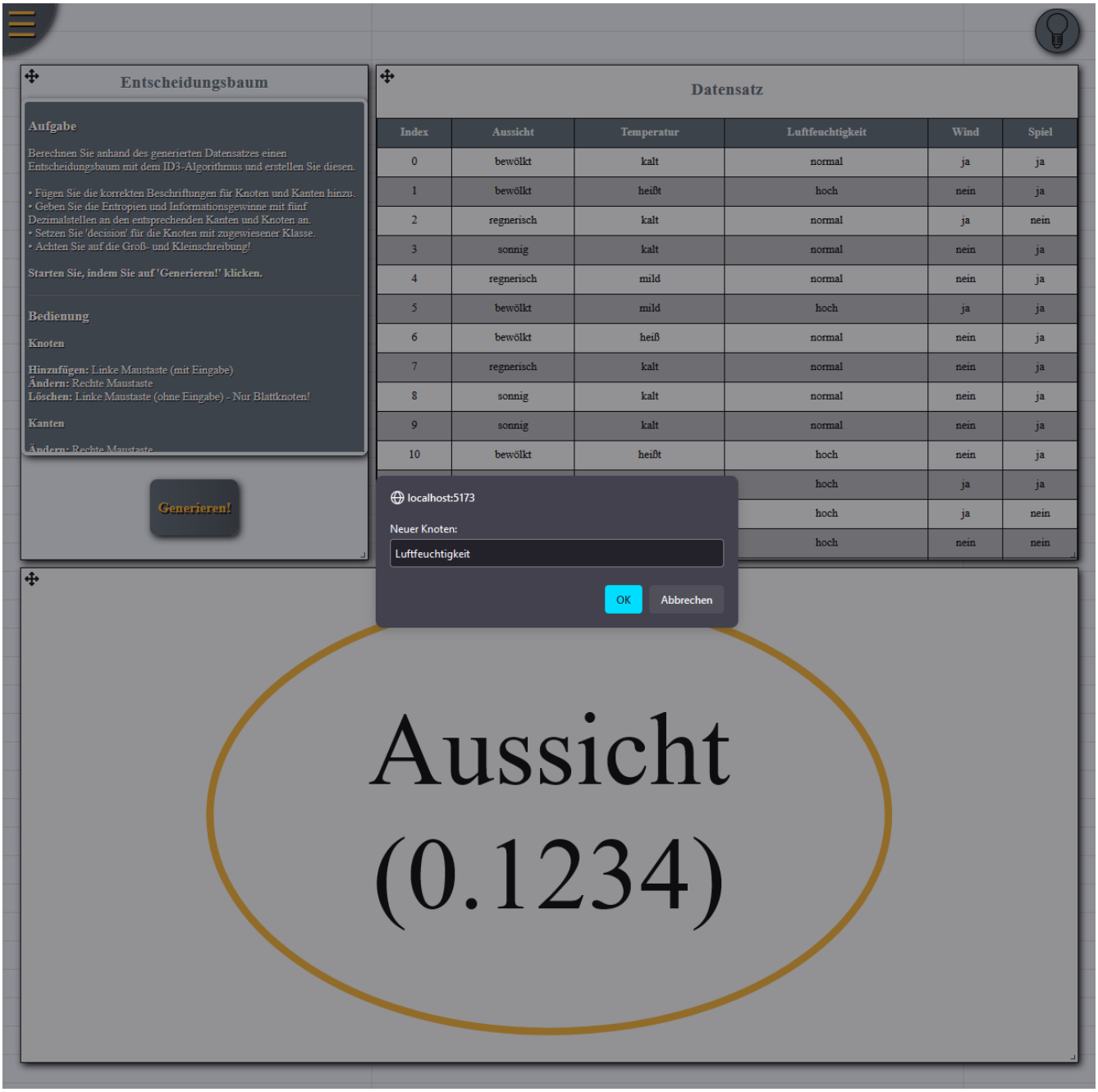


Abbildung 12. Anzeige eines Prompts für das hinzufügen eines neuen Knotens

Prüfung der Eingaben

Nachdem der Studierende eine Eingabe getätigt hat, erfolgt automatisch eine Überprüfung. Bei korrekter Eingabe am Knoten oder an der Kante wird das entsprechende Objekt grün hervorgehoben. Andernfalls bleibt der Knoten bzw. die Kante orange, um auf einen Fehler hinzuweisen.

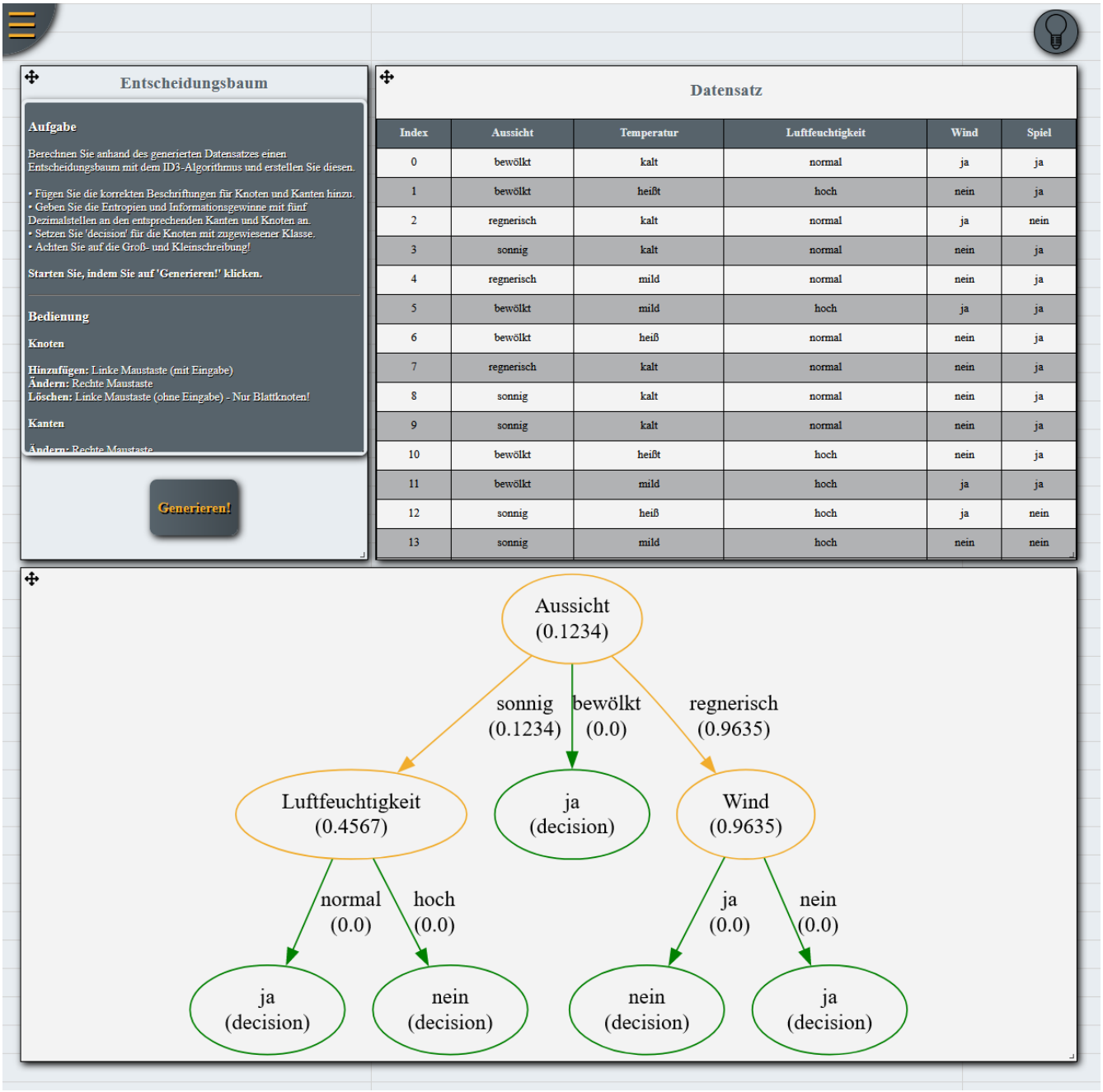


Abbildung 13. Prüfung des Entscheidungsbaums durch farbliche Hervorhebungen

Meldung bei vollständiger Lösung der Aufgabe

Wenn der Studierende den Entscheidungsbaum korrekt und vollständig aufgebaut hat, wird eine Meldung angezeigt, die den erfolgreichen Abschluss der Aufgabe bestätigt.

Abbildung 14. Meldung an den Studierenden bei erfolgreiche Lösung der Aufgabe

Hilfestellungen für den Studierende

Wenn der Studierende Unterstützung bei der Lösung der Aufgabe benötigt, gibt es oben rechts eine Glühbirne. Durch einen Klick darauf erhält der Studierende Hinweise für die Bearbeitung der Aufgabe. Die Hinweise umfassen das Vorgehen des ID3-Algorithmus, Informationen zur Entropie sowie die Berechnungsformeln für den Informationsgewinn und die Entropie.

Abbildung 15. Anzeige einer Hilfestellung für das Vorgehen des ID3-Algorithmus

4.5.2. Aufgetretene Probleme

Die Umsetzung des neuen Aufgabentyps in ALADIN stellte eine Herausforderung dar, insbesondere bei der anfänglichen Schwierigkeit beim Verständnis des Frameworks. Das Fehlen einer vorhandenen Dokumentation machte den Einstieg schwierig und erschwerte somit die Einarbeitung.

Da das Verständnis nicht ausreichend vorhanden war, insbesondere bei der Handhabung der Datenübertragung zwischen Frontend (CARPET) und Backend (ALADIN), wurde um das Hindernis zu umgehen, eigenständig eine Schnittstelle entwickelt, um die notwendigen Daten, in diesem Fall die Übertragung des aufgebauten Entscheidungsbaums vom Studierenden, übertragen zu können.

Aufgrund der zeitintensiven Einarbeitung konnten einige geplante Erweiterungen des Aufgabentyps nicht umgesetzt werden. Zu den nicht realisierten Erweiterungen gehören unter anderem die Klassifizierung neuer Datensätze sowie die Anwendung des Pruning-Verfahrens nach dem Aufbau des Entscheidungsbaums als erweiterte Aufgabe.

4.6. Ausblick

Durch die bisherige Entwicklung des Aufgabentypen wurde eine interaktive Plattform geschaffen, die es Studierenden ermöglicht, den berechneten Entscheidungsbaum mit dem ID3-Algorithmus zu üben.

Um den Lernumfang und die Anwendungsvielfalt zu erweitern, bietet das Projekt zahlreiche Möglichkeiten zur Erweiterung:

1. **Anzeigen des Lösungsbaums bzw. weitere mögliche Lösungsoptionen**
   Eine Erweiterung des Aufgabentyps könnte darin bestehen, dass nach Lösen einer Aufgabe der Lösungsbaum mit angezeigt wird. Insbesondere bei Aufgaben, bei denen Mehrfachlösungen vorhanden sind, könnte dies für die Studierenden interessant sein. Dadurch wird nicht nur die individuelle Lösung betrachtet, sondern auch alternative Lösungsoptionen. Dies könnte ein umfassenderes Verständnis für die Berechnung verschiedener Entscheidungsbaumstrukturen ermöglichen, die durch den ID3-Algorithmus erzeugt werden.
2. **Hinweise zu den Fehlern im Baum dem Studierenden angeben**
   Eine weitere mögliche Erweiterung könnte die Implementierung von Hinweisen zu Fehlern bei den Eingaben und dem Aufbau des Entscheidungsbaums sein. Dadurch werden Studierende nicht nur darauf hingewiesen, an welcher Stelle ein Fehler aufgetreten ist, sondern erhalten auch eine Erklärung, warum der Fehler aufgetreten ist. Dies kann den Lerneffekt der Studierenden weiter fördern. Erste Ansätze zur Umsetzung von Hinweisen zu den Fehlern wurden bereits beim Prototypen entwickelt.
3. **Pruning-Verfahren als Erweiterung des Aufgabentyps**
   Nachdem ein Entscheidungsbaum aufgebaut wurde, könnte eine Erweiterung des Aufgabentyps ermöglichen, den Baum mithilfe verschiedener Pruning-Verfahren zu "stutzen". Dies könnte Studierenden das Verständnis dafür vermitteln, wie Pruning-Techniken angewendet werden können, um übermäßig komplexe Bäume zu vereinfachen.
4. **Klassifizierung neuer Datensätze als Erweiterung des Aufgabentyps**
   Eine mögliche Fortführung und Erweiterung des Aufgabentyps könnte darin bestehen, nachdem die Studierenden den Entscheidungsbaum durch ein Pruning-Verfahren gestutzt haben, neue Datensätze anhand des gegebenen Entscheidungsbaums zu klassifizieren. Dadurch könnte auch die praktische Anwendung des Entscheidungsbaums bei den Studierenden hervorgehoben werden.
5. **Implementierung weiterer Algorithmen**
   Um den Lernumfang zu erweitern, könnten neben dem ID3-Algorithmus zusätzliche Algorithmen implementiert werden. Dies würde den Studierenden die Möglichkeit bieten, verschiedene Ansätze zur Berechnung von Entscheidungsbäumen zu üben. Beispiele für weitere Algorithmen könnten der C4.5-Algorithmus sein, eine Weiterentwicklung des ID3-Algorithmus oder der Random-Forest-Algorithmus.
6. **Vielfältige Aufgabenthemen**
   Statt nur mit einem vorgegebenen Datensatz zu arbeiten, könnte aus verschiedenen "Themengebieten" Datensätzen ausgewählt werden. Dies würde die Generierung unterschiedlicher Aufgabenstellungen und den Schweregrad ermöglichen, wie z.B. die Kreditwürdigkeitsprüfung oder andere relevante Szenarien. Dies fördert sowohl die Vielseitigkeit des Lernmaterials als auch ermöglicht es, unterschiedliche Berechnungsfälle zu üben.

5. Aufgabentyp - Regex Puzzle

5.1. Aufgabenstellung

5.2. Aufgabenbeschreibung

5.3. Anforderung

5.4. Planung

5.5. Technische Umsetzung

5.6. Ergebnisse

5.7. Ausblick

6. Aufgabentyp - Chemie

6.1. Aufgabenstellung

Die Aufgabe und das dazugehörige Programm wurden in Zusammenarbeit mit Prof. Dr. Teichert der Technischen Universität Chemnitz entwickelt. Maßstäbe und Begrenzungen der Aufgabenstellung richten sich inhaltlich nach seinen Anforderungen und Wünschen.

6.2. Aufgabenbeschreibung

Die Aufgabe erforderte es, ein Programm zu erstellen, welches aus einer endlichen, vorgegebenen Liste zufällig Stoffe auswählte und diese unter Einhaltung der chemischen Regeln in einer Reaktion darstellte. Diese Reaktion basiert auf dem Konzept der elektrophilen aromatischen Substitution und das Ziel besteht darin, dass der Student reaktionsbeschreibende Eigenschaften nennen und das fertige Reaktionsprodukt in einem implementierten Editor zeichnen muss. Diese zwei Aufgabenteile sollen, anschaulich und intuitiv in der Verwendung, dargestellt werden.

6.2.1. Chemischer Hintergrund

Die elektrophile Substitution gehört zur Gruppe der Substitutionsreaktionen in der organischen Chemie. Ein Atom oder eine Atomgruppe wird durch ein elektrophiles Teilchen ersetzt. Ein Elektrophil hat eine positive Ladung oder Teilladung und ist daher bestrebt, Elektronen aufzunehmen (elektronenliebend). Das Programm beinhaltet ausschließlich die Reaktion mit Benzolringen, da diese eine der häufigsten Reaktionen dieses Typs sind. Dabei wird der Benzolring von elektrophilen Teilchen angegriffen und ein Wasserstoffatom verdrängt. Solch eine Reaktion beinhaltet abgesehen von Benzolring drei weitere Ausgangsstoffe. Die Reagenz (X) ist der Stoff der Reaktionsbedingung, in diesem Beispiel handelt es sich um eine Bromierung, folglich ist die Reagenz ein Brommolekül. Der Katalysator spaltet besagte Reagenz auf, um Reaktionen zu ermöglichen, in diesem Fall ist dies der Katalysator FeBr3. Der letzte Ausgangsstoff ist der Erstsubstituent ®, dieser ist sehr variabel und kann ein Ion oder ein ganzes Molekül sein. In der Reaktion reagiert sowohl der Erstsubstituent, als auch die Regenz an den Benzolring. Die Reagenz reagiert relativ zum Erstsubstituenten an einer von drei möglichen Positionen: Para, Meta, oder Ortho. Das ist abhängig von der Reagenz selbst, dem Erstsubstituenten, sowie jeglichen Reaktionsbedingungen.

6.3. Anforderung

Die wichtigste Anforderung ist das intuitive Navigieren. Eine benutzerfreundliche Benutzeroberfläche sollte eine Navigation bieten, die für den Benutzer leicht verständlich ist. Dies umfasst klar gekennzeichnete Aufgabenfelder und Navigationselemente, die dem Benutzer eine nahtlose Interaktion mit dem System ermöglichen. Ein weiterer wichtiger Aspekt ist die wiederholbare Nutzung des Systems. Die Benutzeroberfläche sollte so gestaltet sein, dass Benutzer das System wiederholt nutzen können. Dies erfordert eine konsistente Strukturierung der Benutzeroberfläche und eine klare Darstellung von Funktionen und Optionen, sodass eine Nutzung ohne ständiges Neuladen der Seite erfolgen kann. Die Verständlichkeit der Aufgabenstellung ist ebenfalls entscheidend. Die Aufgabenstellung und Anweisungen innerhalb des Systems müssen klar und verständlich formuliert sein, um Missverständnisse zu vermeiden und den Benutzern eine klare Orientierung zu bieten.

Des Weiteren muss das System einen Editor bereitstellen, der es den Benutzern ermöglicht, die chemische Reaktion visuell darzustellen. Dieser Editor muss benutzerfreundlich sein und eine einfache Bearbeitung der Reaktion ermöglichen. Benutzer müssen in der Lage sein, die Eigenschaften der Reaktion in das System einzugeben. Daher benötigt die Benutzeroberfläche eine Schnittstelle, die es den Benutzern ermöglicht, die relevanten Informationen präzise und korrekt einzugeben. Zusätzlich sollte das System eine Funktion zum Zeichnen des Reaktionsprodukts durch die Benutzer bereitstellen. Dies ermöglicht es den Benutzern, ihr Verständnis der Reaktion auf visuelle Weise zu demonstrieren. Schließlich muss das System die eingegebenen Informationen validieren, um sicherzustellen, dass sie mit den vorgegebenen Lösungen übereinstimmen. Dadurch wird sichergestellt, dass die Benutzer korrekte und akkurate Informationen eingeben, um ein direktes Feedback zu ihrem Wissensstand zu erhalten. Diese Anforderungen bilden die Grundlage für die Gestaltung und Entwicklung der Benutzeroberfläche, um eine optimale Benutzererfahrung zu gewährleisten.

6.4. Planung

Bei der Planung des IT-Projekts müssen verschiedene Aspekte berücksichtigt werden, die eng mit den chemischen Regeln und den Vorgaben von Prof. Teichert zusammenhängen. Diese Regeln dienen als Grundlage für die Entwicklung des Programms und müssen entsprechend umgesetzt werden. Dies kann durch die Implementierung von Algorithmen und Logik geschehen, die die chemischen Prinzipien abbilden und verarbeiten. Um auf die von Prof. Teichert bereitgestellten Daten zugreifen zu können, ist es notwendig, Schnittstellen zu implementieren, die es dem Programm ermöglichen, diese Daten abzurufen und zu verarbeiten. Dies kann beispielsweise durch JSON-Datei geschehen, hierbei werden die Daten in jenen Dateien gespeichert und später, wenn abgerufen von Aladin wieder ausgelesen. Unser Framework benötigt ein bestimmtes Datenformat, um effizient mit den Daten umgehen zu können. Ein strukturiertes Format wie JSON oder XML könnte verwendet werden, um die Informationen über chemische Reaktionen und Eigenschaften klar und einheitlich darzustellen. Die Benutzereingaben müssen durch eine benutzerfreundliche Oberfläche gestaltet werden, die klare Anweisungen und Eingabefelder bietet, um die erforderlichen Informationen von den Benutzern zu erhalten. Dies kann durch die Verwendung von Formularen, Dropdown-Menüs oder anderen interaktiven Elementen geschehen. Für die Darstellung und Zeichnung chemischer Reaktionen empfehlen wir den Einsatz des JSME Editors, da dieser von Prof. Teichert empfohlen wird und bereits von den Studenten vertraut ist. Dieser Editor erfüllt alle relevanten Anforderungen und bietet eine benutzerfreundliche Erfahrung. Die Implementierung eines externen Editors wie JSME erfordert möglicherweise die Integration entsprechender Schnittstellen oder Plugins in das Framework, um die Interaktion zwischen dem Programm und dem Editor zu ermöglichen. Dies kann durch die Nutzung von APIs oder anderen Integrationstechniken erfolgen, um eine reibungslose Zusammenarbeit zwischen beiden Komponenten sicherzustellen.

6.5. Technische Umsetzung

Zunächst folgte die Umsetzung eines Prototypen in Java, dieser diente zunächst dazu, die Aufgabe grundlegend zu modellieren, um die Komplexität zu erfassen. Die Interaktion folgte hier zunächst rein über Konsolen In- und Outputs, welche jedoch im Rahmen des Prototypen ausreichend waren, um erste Funktionen implementieren und testen zu können. Anschließend folgte die Analyse des vorhandenen Backends Aladin und dem dazugehörigen Frontend Carpet. Die zugrundeliegenden Programmiersprachen sind Typescript, Vue, sowie Json, dies überschneidet sich jedoch nicht mit der Programmiersprache des erstellten Prototypen. Nach ersten Einschätzungen wäre eine direkte

Implementierung der Aufgabenlogik in Front- und Backend eine komplexere Lösung als ein Aufruf des Java Programm, welches dann Inhalte entsprechend wiedergibt. Der Prototyp wurde anschließend entsprechend angepasst: Als Input erfolgt das Einlesen einer Json-Datei mit dem festen Namen “Substituenten.json”, diese befindet sich im Root-Verzeichnis der Software, in der sie eingefügt wurde. Weiterhin gibt es eine Datei “Reagenz.json”, welches sich ebenfalls im Root-Verzeichnis befinden muss. Der Prototyp liest den Inhalt dieser Datei aus, erstellt Parameter einer zufälligen Aufgabe, sowie deren Lösung und speichert alle benötigten Inhalte in einer Datei namens “Antwort.json”. Diese Datei wird ebenfalls im Root-Verzeichnis erstellt. Das gesamte Programm liegt als Jar-Datei vor und kann, sollten alle Dateien entsprechend abgelegt sein, in jeder Ordnerstruktur aufgerufen werden. Diese Jar-Datei wurde dann entsprechend im Backend von Aladin im Ordner Tasks unter Chemistry eingefügt. Sie wird dann über das Backend aufgerufen, es werden neue Daten generiert und in Antwort.json gespeichert, aus welcher diese dann wieder ausgelesen und verwendet werden können. Wenn die Oberfläche von Aladin aufgerufen wird und der Nutzer die Kategorie Chemie auswählt, werden die benötigten Komponenten zusammengestellt und angezeigt. Dazu gehören das Aufrufen der Jar-Datei und das Verarbeiten ihrer Ausgabe, das Anzeigen der Aufgabenstellung und das Einbinden des JSME Editors. Dieser wird jedes mal dynamisch aufgerufen und in mehreren Ausführungen angezeigt.

Um dem Nutzer die Aufgabenstellung zu verdeutlichen, wird auf der ersten Seite dem Nutzer lediglich ein Feld angezeigt, in welchem die Reaktion mit den dazugehörigen Reaktionsstoffen dargestellt wird. Dabei handelt es sich um einen JSME Editor, der mittels mehrer Parameter und einem String aufgerufen wird. Die Parameter dienen dazu, alle Schaltflächen für die Bearbeitung zu entfernen und den Editor somit auf Read-only einzustellen. Der String enthält einen komplexen “SMILES” Code, der vom Editor als Reaktionsgleichung erkannt und angezeigt wird. Dieser wurde ebenfalls von der Jar Datei erzeugt. Weiterhin gibt es eine Komponente, die zwei Dropdown Felder enthält. Der Nutzer muss zwei Teilaufgaben lösen, indem er die richtigen Elemente in den Feldern auswählt und anschließend seine Antworten überprüfen lässt. Erst wenn diese Benutzereingabe durch Carpet validiert wurde, erhält der Nutzer die Möglichkeit, auf die nächste Seite zu wechseln.

Auf der zweiten Seite wird der Nutzer aufgefordert, das chemische Hauptprodukt selbst darzustellen. Dazu wird hier erneut die Reaktionsgleichung mit einem Read-only Editor angezeigt. Weiterhin gibt es einen zweiten Editor, dieser ist jedoch leer und enthält alle benötigten Schaltflächen, damit der Nutzer verschiedene chemische Stoffe zeichnen kann. Jede Benutzereingabe sendet den Inhalt der JSME Editors als String an Carpet, wo dieser String mit der zuvor durch die Jar-Datei erstellten Lösung abgeglichen wird. Erst nach Eingabe der korrekten Lösung erhält der Nutzer eine visuelle Bestätigung und kann zurück zur Startseite von Aladin kehren.

6.6. Ergebnisse

Anforderung	**Status**
Geben Sie den Ort der Zweitsubstitution in einer elektrophilen aromatischen Substitution an!	gelöst (mittels eines Dropdown Feldes welches die Eingabe des Studenten erwartet)
Zeichnen Sie das zu erwartende Hauptprodukt.	gelöst (mittels einbinden des JSME-Editors)
Erwarten Sie eine Überreaktion des Erstprodukts unter den gegebenen Reaktionsbedingungen	gelöst (mittels eines Dropdown Feldes welches die Eingabe des Studenten erwartet)

Anforderung	Status
Validierung der, von dem Studenten, eingegeben Daten	teilweise gelöst (mittels Validierungsfunktionen zum Abgleich der Eingaben mit der richtigen Lösung)

Die Umsetzung in Aladin und Carpet stellte den umfangreichsten Teil unserer Arbeit dar. Dabei war die Unterstützung von Herrn Christ von entscheidender Bedeutung, um ein fundiertes Verständnis für die beiden Frameworks zu entwickeln. Seine Kenntnisse halfen uns, die Herausforderungen zu bewältigen und die Funktionalitäten effektiv zu nutzen. Durch die Strukturierung der Probleme in zwei Bereiche konnten wir die Aufgaben klar unterteilen und effizient bearbeiten. Diese Herangehensweise ermöglichte eine gezielte Umsetzung und trug maßgeblich zum Erfolg des Projekts bei. Positiv betrachtet ermöglichten Aladin und Carpet eine umfassende Realisierung unserer Ziele und boten vielfältige Möglichkeiten zur Gestaltung und Implementierung.

6.6.1. Probleme

Während des Projekts traten verschiedene Probleme und Herausforderungen auf. Die Einarbeitung in das bestehende Framework war mit einer wesentlich höheren Komplexität verbunden, als zu Beginn des Projekts angenommen. Die Implementierung war sehr zeitaufwändig und erforderte intensive Unterstützung durch Herrn Christ. Darüber hinaus gab es während der Umsetzung einige technische Schwierigkeiten, die den Fortschritt zeitweise verlangsamten.

6.7. Ausblick

Das Programm stellt eine akzeptable Basis für eine Weiterentwicklung dar. Die Implementierung in Java besitzt geringe Komplexität und lässt sich schnell überarbeiten und entsprechend anpassen. Weiterhin sind alle notwendigen Komponenten in Aladin implementiert, sowie alle Schnittstellen, die dazu dienen, eine lauffähige Oberfläche zu erzeugen. Eine Erweiterung, die bereits teilweise implementiert ist, ist die Erweiterung um verschiedene mögliche Reagenzien. Das Programm arbeitet bereits mit einer Json-Datei, in welcher besagte Stoffe übergeben werden, jedoch sind diese aus chemischer Sicht nicht vollständig korrekt, sondern wurden lediglich aus informatischer Sicht für Testzwecke verwendet. Sollte eine Variation der Reagenz in Zukunft hinzugefügt werden, so müsste die Jar-Datei inhaltlich, chemisch korrekt, angepasst werden, um korrekte Ergebnisse liefern zu können. Weiterhin könnte der Nutzer auch weitere Reaktionsparameter, auch im Zusammenhang mit weiteren Reagenzien, abgefragt werden. Dazu müsste die Anzeige der Aufgabenstellung entsprechend verändert und eine Eingabe für den Nutzer geschaffen werden. Dazu bietet sich die bereits existierende Komponente, die die Dropdown Felder enthält an, es gibt jedoch auch eine Reihe weiterer Komponenten, die für solche Zwecke eine potentielle Lösung darstellen könnten.

7. Reflexion

7.1. Jonas Hölzel

7.2. Norman Sebastian Arnold

7.3. Tanja Dietrich

Im Rahmen des Projektseminars konnte ich wertvolle Erfahrungen sammeln und meine Kenntnisse vertiefen, im Hinblick auf die eigenständige Entwicklung einer generativen Aufgabe unter Berücksichtigung vorgegebener Anforderungen. Die Konzeptionsphase erwies sich für mich als entscheidender Schritt, in dem ich lernen musste, wie die Strukturierung eines Aufgabentyps für die Berechnung eines Entscheidungsbaums erfolgt und welche Komponenten dabei erforderlich sind. Im Rückblick erkenne ich, dass eine strukturierte Vorgehensweise für die Umsetzung in ALADIN ebenso hilfreich gewesen wäre, um den Aufgabentyp technisch effizient umzusetzen. Der implementierte Code hätte durch eine klare Strukturierung von Klassen und Funktionen deutlich übersichtlicher und verständlicher gestaltet werden können. Diese Erkenntnis nehme ich als wertvolle Lektion für kommende Projekte aus dem Projektseminar mit.

Die Einarbeitung in ALADIN stellte eine Herausforderung dar, besonders aufgrund meiner begrenzten Programmierkenntnisse. Die anfängliche Einführung ermöglichte zwar einen groben Einblick in die Implementierung unserer Aufgabentypen in ALADIN, jedoch fehlte mir eine umfassende Dokumentation, die eine selbstständige und gezielte Einarbeitung in das Framework ermöglicht hätte. Dies führte zu hohem Zeitaufwand in der Einarbeitung, insbesondere zu Beginn der Implementierung, um das erforderliche Verständnis aufzubauen. Erst gegen Ende des Semesters klärte sich ansatzweise das Konzept, was schließlich die Umsetzung der Funktionalitäten in ALADIN ermöglichte.

Ein weiterer Aspekt war die Einzelarbeit im Projektseminar, da es besonders zu Beginn des Semesters kaum zu einem Austausch innerhalb der Gruppe kam. Probleme wurden eigenständig und allein bewältigt. Ein verstärkter Austausch und gemeinsame Problemlösungen hätten aus meiner Sicht zu einer produktiveren Arbeitsweise führen können. Erst gegen Ende des Semesters wurde die Gruppendynamik besser genutzt, insbesondere bei der gemeinsamen Bearbeitung der Präsentation und des Projektberichts. Eine mögliche Idee wäre, dass für die Entwicklung eines Aufgabentyps möglicherweise mindestens zwei Studierende zusammenarbeiten. Dadurch hätte nicht nur die Effektivität der Aufgabenbewältigung verbessert werden können, sondern es hätte auch die Möglichkeit eröffnet, Problemstellungen, insbesondere bei der Implementierung in ALADIN gemeinsam zu lösen, wie auch verschiedene Umsetzungsansätze zu diskutieren und mögliche weitere Aufgabenstellungen umzusetzen.

Trotz der Herausforderungen war die Teilnahme am ALADIN Projektseminar eine wertvolle und lehrreiche Erfahrung. Die gewonnenen Erkenntnisse werden mir bei zukünftigen Projekten von großem Nutzen sein.

7.4. Alessandra Ruff

7.5. Julius Wyrembek

7.6. Vincent Weise

8. Ausblick

8.1. Integration von weiteren interdisziplinären Aufgaben

ALADIN kann um zusätzliche, vielfältige Aufgabentypen erweitert werden. Den Themenbereichen, die ALADIN umfasst, sind keine Grenzen gesetzt. So wären Aufgaben aus Fachbereichen der Modellierung, Geoinformatik, Musik, sowie vielen weiteren möglich. Diese könnten in Zukunft hinzugefügt werden, um die Vielfalt von dem, was mit ALADIN möglich ist, weiter zu erhöhen.

8.2. KI Erweiterung

Durch den gezielten Einsatz von KI, kann auch das ALADIN-Framework profitieren. In ALADIN könnte ein KI-Einsatz zum Beispiel bedeuten, dass die gelösten Aufgaben von der KI ausgewertet werden, oder neue Aufgaben durch die KI generiert werden. Außerdem kann das Feedback der Nutzer, wenn bereits integriert, ausgewertet werden um die Qualität der Aufgaben zu erhöhen, aber auch die Lösungshilfen könnten dynamisch mit KI-Einsatz generiert werden.

8.3. Tracking des Lernfortschritts

Damit der Lernprozess für jeden Studierenden individualisiert werden kann, wäre es erdenklich, für jede Aufgabe Punkte zu verteilen und den Lernfortschritt zu speichern. So könnten Statistiken erstellt werden, die dem Studierenden helfen seine Problemfelder zu identifizieren und besser verstehen zu können. Dies könnte in Form von persönlichen Profilen für jeden Nutzer erreicht werden, auf welchem sie einen Überblick über das Erreichte, sowie den noch offenen Themen erhalten.

8.4. ALADIN als Wissensvermittlungsplattform

Um ALADINs Vielfalt zu erweitern, wäre es sinnvoll, zusätzlich zu den Übungen auch Lernmaterialien zur Verfügung zu stellen. Dadurch könnten Studierende nicht nur ihr bereits vorhandenes Wissen festigen, sondern sich auch neues Wissen aneignen. Eine Kombination aus theoretischem Wissen über ein bestimmtes Thema und anschließendem praktischem Üben würde den Studierenden helfen, Lehrmaterialien aus dem Studium besser zu verstehen und zu verinnerlichen. Diese könnten kontinuirlich aktualisiert werden, sodass ALADIN den wandelnden Anforderungen gerecht wird.