TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Entwicklung eines Machine-Learning Modells zur Auswertung von Überlebensdaten und Bereitstellung als Web-Service

Bachelor-Thesis

von

Stefan Tissen

08. September 2020

Technische Universität Kaiserslautern,
Department of Computer Science,
67653 Kaiserslautern,
Germany

Examiner:
Prof. Dr. Klaus Schneider
Dr. Sascha Feth

EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit versichere ich, dass die vorliegende Arbeit selbstständig verfasst und keine weiteren als die angegebenen Quellen und Hilfsmittel verwendet wurden.

Kaiserslautern, den 08.09.2020

_____

Stefan Tissen

ZUSAMMENFASSUNG

Die Bachelorarbeit beinhaltet eine verbesserte und angepasste Version des Zuverlässigkeitsassistenten von Jurojin. Bei Jurojin handelt es sich um eine Statistiksoftware zur Planung und Auswertung von Betriebsfestigkeitsversuchen, welche vom Fraunhofer-Institut für Techno- und Wirtschaftsmathematik (ITWM) entwickelt und gepflegt wird. Hierbei soll der Nutzer bei der Wahl des passenden Algorithmus unterstützt werden. Dies wird durch einen Experten, welcher eine passende Bewertung der Algorithmen in die Software implementiert, umgesetzt. Dieser Prozess wird in der aktuellen Umsetzung, mühseliger sowie fehleranfälliger, da die Anforderungen regelmäßig erweitert werden. Daher liegt es nahe dies mithilfe von maschinellem Lernen umzusetzen. Damit nicht nur Jurojin von dieser Funktionalität profitieren kann, wird diese als Webservice umgesetzt. In dieser Bachelorarbeit wird die passende Datenstruktur der Datenbank beschrieben und erklärt, wie diese verwendet werden. Hierzu werden passende Algorithmen vorgestellt. Anschließend wird die Implementierung als Web-Service, welches von Jurojin verwendet wird, präsentiert.

ABSTRACT

The bachelor thesis is motivated by the improvement of the reliability wizard included in Jurojin. Jurojin is a statistical program for planning and evaluating samples, which is developed and maintained by the “Fraunhofer-Institut für Techno- und Wirtschaftsmathematik” (ITWM). It is intended to assist the user in choosing the appropriate algorithm. This is implemented by an expert who implements a suitable evaluation of the algorithms in the software. But this process is also complicated and error-prone in the current implementation, as the requirements are regularly expanded. Therefore, it is obvious to implement it with the help of machine learning. So that not only Jurojin can benefit from this functionality, it is implemented as a web service. This bachelor thesis describes the appropriate data structure of the database and explains how to use it. Suitable machine learning models are presented. The implementation is then presented as a web service used by Jurojin.

INHALTSVERZEICHNIS

1 EINLEITUNG

Das Fraunhofer Institut für Techno- und Wirtschaftsmathematik (ITWM) gehört zu den größten mathematischen Forschungsinstituten weltweit. Es entwickelt die Mathematik als Schlüsseltechnologie weiter, gibt innovative Anstöße und setzt diese mittels Modellierung, Simulation sowie Optimierung gemeinsam mit Industriepartnern praktisch um [1]. Einer dieser Lösungen ist unter anderem die Software Jurojin, welche für die Planung und Auswertung von Betriebsfestigkeitsversuchen entwickelt und gepflegt wird. Hierbei orientiert sich die Methodik und Programmgestaltung an Projekten und praktischen Problemstellungen aus der PKW und Nutzfahrzeugindustrie, um Kosten beim Zuverlässigkeitsnachweis einzusparen. Jurojin bietet einen breiten Leistungsumfang an, unter anderem Planung von Lebensdauerversuchen oder Auswertung der Daten mit ausgewählten Methoden [2]. Bei der Auswertung der Daten mit ausgewählten Methoden hilft der sogenannte „Zuverlässigkeitsassistent“ den Nutzer bei der Wahl des passenden Algorithmus. Da die jeweiligen Daten unterschiedlich interpretiert werden können und für die korrekte Bewertung der Methoden eine Expertise notwendig ist, ist die aktuelle „klassische“ Umsetzung häufig fehleranfällig und schwierig zu warten. Außerdem sind unterschiedliche Bewertungen durch subjektive Einschätzungen möglich. Sobald eine Anpassung des Zuverlässigkeitsassistenten umgesetzt wurde, muss der Kunde sich jedoch bis zu der Auslieferung der neuen Version von Jurojin gedulden, was ebenfalls den Optimierungsprozess des Zuverlässigkeitsassistenten verlangsamt. Hier bietet sich eine Lösung mithilfe von maschinellem Lernen als Webservice an. Somit können Daten von Experten generiert und gesammelt werden, welche anschließend analysiert und mithilfe eines passenden Algorithmus in einem Entscheidungsbaum umgewandelt werden. Der daraus resultierende Entscheidungsbaum kann nun die Expertise übernehmen. Um das Problem der langwidrigen Auslieferung zu lösen, wird dies als Webservice umgesetzt. Da es als Webservice umgesetzt wird, kann nicht nur Jurojin von der Funktionalität profitieren.

1.1 AKTUELLE UMSETZUNG

Die aktuelle Lösung des Bewertungssystems sucht zum aktuellen Datensatz des Kunden den besten Algorithmus. Optional ist ebenfalls die Angabe eines Versuchsplans und Interpretationsdaten der Gruppeneinträge möglich. Die Bewertung wurde in eine If-Anweisungen-Struktur umgesetzt, indem Richtwerte abgefragt wurden und nach diesen entschieden wurde, welche Bewertung am ehesten zutrifft. Sollte der Kunde eine unpassende Bewertung für einen bestimmten Datensatz finden, kann dieser dies dem Hersteller mitteilen. Der Statistiker am ITWM führt manuell eine Bewertung durch und die Programmierer suchen eine neue If-Anweisungen-Struktur, mit Beachtung der Unit-Tests, die diesem neuen Fall gerecht wird. Anschließend erhält der Kunde in der nächsten Auslieferung der Software die neue DLL mit dem neuen Bewertungssystem.

1.2 Probleme der Aktuellen Umsetzung

Die aktuelle Umsetzung birgt einige Probleme mit sich. Es ist für den menschlichen Experten müheselig, die Bewertung von statistischen Methoden vorzunehmen. Außerdem ist es für den Programmierer aufwendig eine passende If-Anweisungen-Struktur zu finden und gleichzeitig einen Überblick der immer mehr werdenden Testdaten zu haben. Wurde eine passende If-Anweisungen-Struktur gefunden, muss diese auf alle Kundendaten abbildbar sein, da dies nicht immer möglich ist muss der Programmierer entscheiden an welchen Stellen Abstriche vorgenommen werden. Wurde die jeweilige Anpassung umgesetzt, muss der Kunde sich jedoch bis zu der Auslieferung der neuen Version von Jurojin gedulden, was ebenfalls den Optimierungsprozess des Zuverlässigkeitsassistenten verlangsamt. Sollte der Kunde nun mit einer Bewertung nicht einverstanden sein, kann er sich beim Jurojin-Support melden. So kommen neue Kundendaten als Testdaten und der Prozess wiederholt sich. Mit zunehmenden Kundendaten erschwert sich dieser Prozess für den menschlichen Experten jedoch und es müssen immer mehr Abstriche in Kauf genommen werden.

1.3 Umsetzung mit maschinellem Lernen

Maschinelles lernen kann diesen langwidrigen Prozess vereinfachen und optimieren. Es gibt zahlreiche Algorithmen die mithilfe von Trainingsdaten Modelle trainieren können. Hierfür muss für die Datenbank eine passende Datenstruktur definiert werden. Die Datenbank muss mit Kundendaten gefüllt werden und ein passender Algorithmus trainiert mithilfe dieser Daten ein Modell. Einige Algorithmen lassen zu, das trainierte Modell auszuwerten, somit können Schlussfolgerungen gezogen werden, welche beim Optimierungsprozess behilflich sind. Am Ende ist es nur noch notwendig die Datenbank mit neuen Kundendaten zu ergänzen und ein neues Modell trainieren zu lassen. Aufgrund der variablen Länge, der in Jurojin enthaltenden Stichprobe ist, es nicht möglich diese ohne Weiteres zum Trainieren zu verwenden. Außerdem läuft die Beurteilung des resultierenden Modells in diesem Anwendungsfall aufgrund der initial niedrigen Datenmenge und dem Interpretationsfreiraum der Bewertungen spezieller aus. Hier muss ebenfalls eine Lösung gefunden werden.

1.4 Vision des fertigen Produkts

Der Kunde soll im besten Fall beim Zuverlässigkeitsassistenten keine Veränderung der Menüführung bemerken. Das Verhalten seitens des Nutzers bleibt somit gleich und die dargestellten Bewertungen bleiben unverändert. Der Zuverlässigkeitsassistent soll die Bewertungen der Algorithmen durch eine Anfrage von einem Web-Service erhalten. Der Web-Service verwendet ein, durch maschinelles Lernen trainiertes, Modell für die Klassifikation der passenden Bewertung. Die Verwendung des Web-Services soll durch eine Jurojin-Lab-Lizenz geschützt werden. Sollte der Benutzer nicht im Besitz dieser Lizenz sein

wird das ursprüngliche Verfahren verwendet. Außerdem bekommt der Nutzer die Möglichkeit alle relevanten Daten im Falle einer unbefriedigenden Bewertung, angepasst zu exportieren um schnell ein serverseitiges Update erhalten, dass dieses Verhalten korrigiert.

1.5 Erzielte Ergebnisse

Maschinelles Lernen kann nur mithilfe von Trainingsdaten erfolgen, die passende Strategie sammelt die Daten ein. Hierfür wurde eine Datenstruktur definiert, die auch bei zukünftigen Anpassungen des maschinellen Lernens verwendet werden kann. Jurojin erhielt dazu ein Exportieren von benutzerdefinierten Bewertungen. Des Weiteren wurde sich, passend zur Problemstellung, ein Algorithmus zum Trainieren eines Modells mithilfe von maschinellem Lernen ausgewählt. Das Problem der variablen Länge wurde durch das Verwenden von beschreibenden Merkmalen gelöst. Eine passende Darstellung und spezielle Validierungsmethoden beurteilen das Modell. Die Lösung wurde als Web-Service implementiert und Jurojin mit der Verwendung des Dienstes erweitert.

2 STAND DER TECHNIK

2.1 JUROJIN

Das Statistikprogramm Jurojin, wird vom Fraunhofer-Institut ITWM für die Planung und Auswertung von Betriebsfestigkeitsversuchen entwickelt und gepflegt. Als Grundlage für die Programmgestaltung dienen praktische Problemstellungen aus der PKW- und Nutzfahrzeugindustrie. Im Leistungsumfang enthalten sind Planung von Lebensdauerversuchen, Auswertung der Daten mit ausgewählten Methoden, grafische Datenaufbereitung, Berücksichtigung kleiner Stichprobenumfänge, etc. Ziel dieser Software ist es, Unternehmen die hohen Kosten für Bauteile und Prüfdauer beim Zuverlässigkeitsnachweis zu reduzieren. Dies kommt vor allem für Unternehmen zugute die nur wenige Prototypen zur Verfügung haben [2].

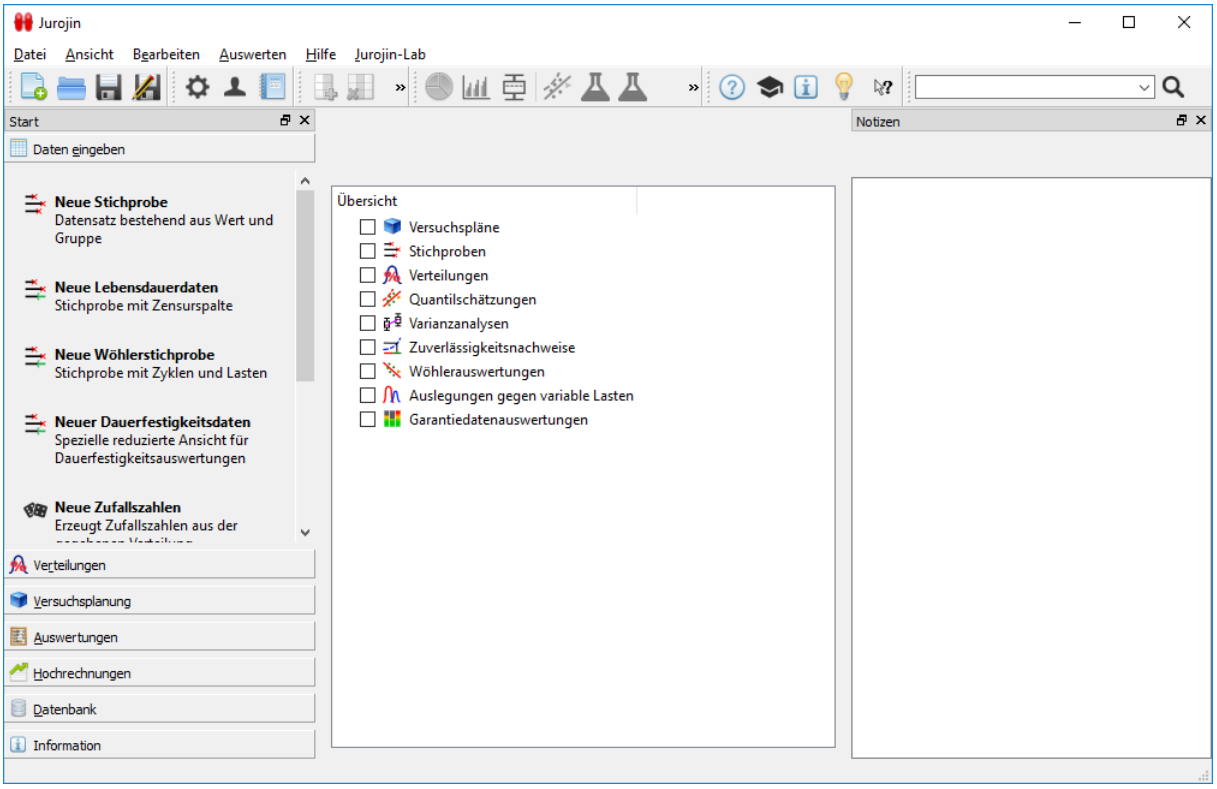

Abbildung 1 - Workbench - Untermenü „Daten eingeben“ geöffnet

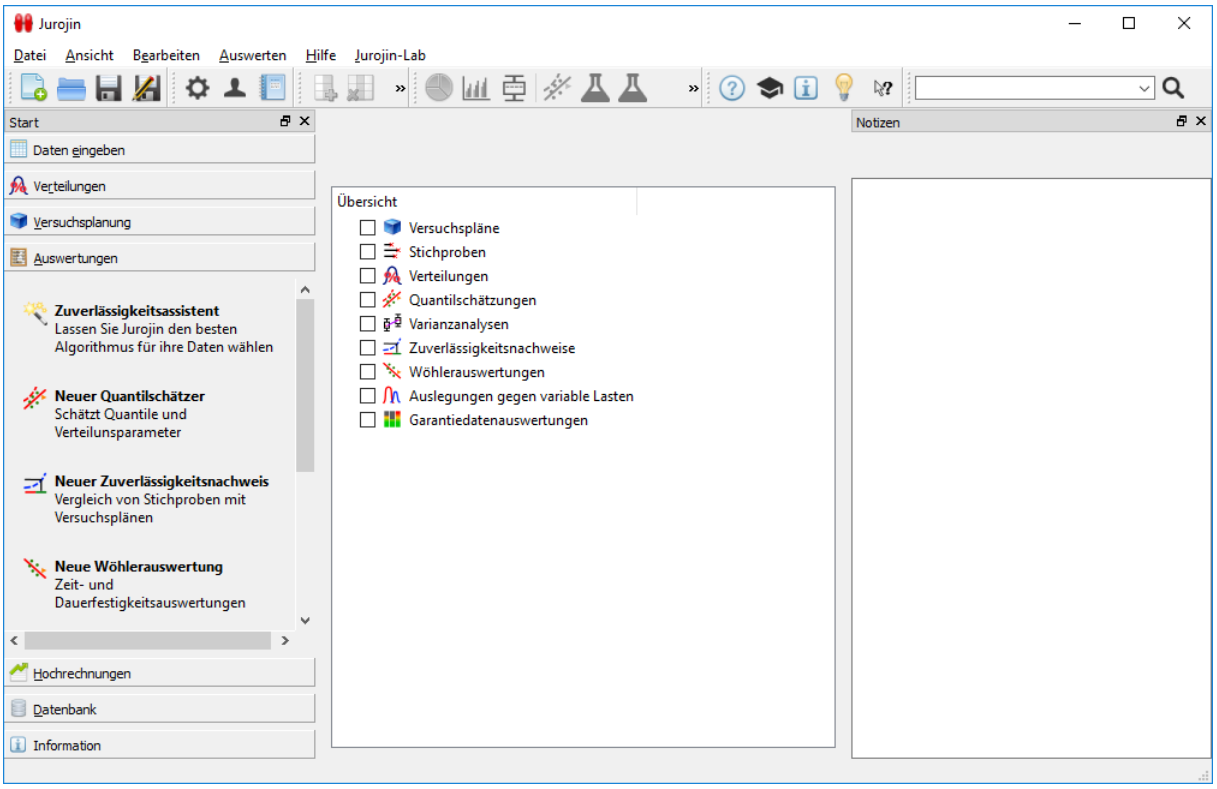

Abbildung 2 – Jurojin – Workbench – Untermenü „Auswertung“ geöffnet

2.1.1 Neuer Zuverlässigkeitsnachweis

Die Freigabeberechnung wird auf Stichproben angewendet, die größtenteils oder komplett aus Durchläufern bestehen, da in diesen Fällen das Modul Quantil- und Verteilungsschätzung nicht anwendbar ist. Für die Durchführung dieses Moduls muss ein Versuchsplan angelegt sein, dessen Werte (z.B. Stichprobenumfang) sich mit der Stichprobe decken [3].

2.1.2 Neuer Quantilschätzer

Der Quantil- und Verteilungsschätzer passt eine Lebensdauerverteilung, bei genügend Ausfällen, (z.B. Weibull oder Lognormal) an den Datensatz an (inkl. Konfidenzlinien) [4].

2.1.3 Gruppenweise Auswertung

Quantil- und Verteilungsschätzer mit Berücksichtigung der Gruppierung der Daten.

2.1.4 Poly-Weibull

Wenn ein Bauteil auf verschiedene Arten oder Positionen ausfallen kann, ist die Gesamtlebensdauer gerade das Minimum dieser einzelnen Risiken. Jeder Ausfall an einer Position ist automatisch einen Durchläufer für alle übrigen Positionen. In dieser Situation kann Jurojin dies automatisch berücksichtigen und eine Poly-Weibull-Verteilung ermitteln [5].

2.1.5 3-Parameter-Weibull

Im Gegensatz zu den gewöhnlichen Weibullverteilungen modellieren 3-Parameter-Weibullverteilungen aus den Daten eine ausfallfreie Zeit. Zur Anpassung der drei Parameter an einem Datensatz, wird die Modified-Maximum-Likelihood-Methode verwendet [6].

2.1.6 Zuverlässigkeitsassistent

Jurojin bietet mehrere Algorithmen zur Auswertung von Daten an. Je nachdem welche Daten bereitgestellt wurden und wie diese interpretiert werden, eignen sich nicht alle Algorithmen zur für den Zuverlässigkeitsnachweis. Da der Nutzer das bestmögliche Ergebnis anstrebt wird ebenfalls zwischen der Eignung des Algorithmus entschieden. Hier wird der Nutzer durch den Zuverlässigkeitsassistenten unterstützt. Optional kann für eine passendere Bewertung ein dazugehöriger Versuchsplan herangezogen werden. Bei Stichproben mit Gruppeneinträgen bietet der Zuverlässigkeitsassistent, wie in Abbildung 3 zu sehen ist, einen zusätzlichen Bereich über die Interpretation der Gruppeneinträge an.

- **Gruppeneinträge nicht beachten**: Die Gruppeneinteilung kann zunächst ignoriert werden, bzw. ist von informativer Natur
- **Gruppen als Teilpopulationen interpretieren**: z.B. Produktionsort, Zulieferer, usw. Solche Situationen führen dazu, dass die Auswertungen gruppenweise erfolgen und typischerweise eher unabhängig voneinander sind
- **Gruppen repräsentieren verschiedene Ausfallursachen**: Ist ein Bauteil nach Ursache A ausgefallen, so hätte es auch nach Ursache B ausfallen können, wäre A nicht "schneller" gewesen. In der Auswertung für Ausfallursache A muss also berücksichtigt werden, dass alle Bauteile, die nicht mit anderer Ausfallursache ein Durchläufer für A sind.

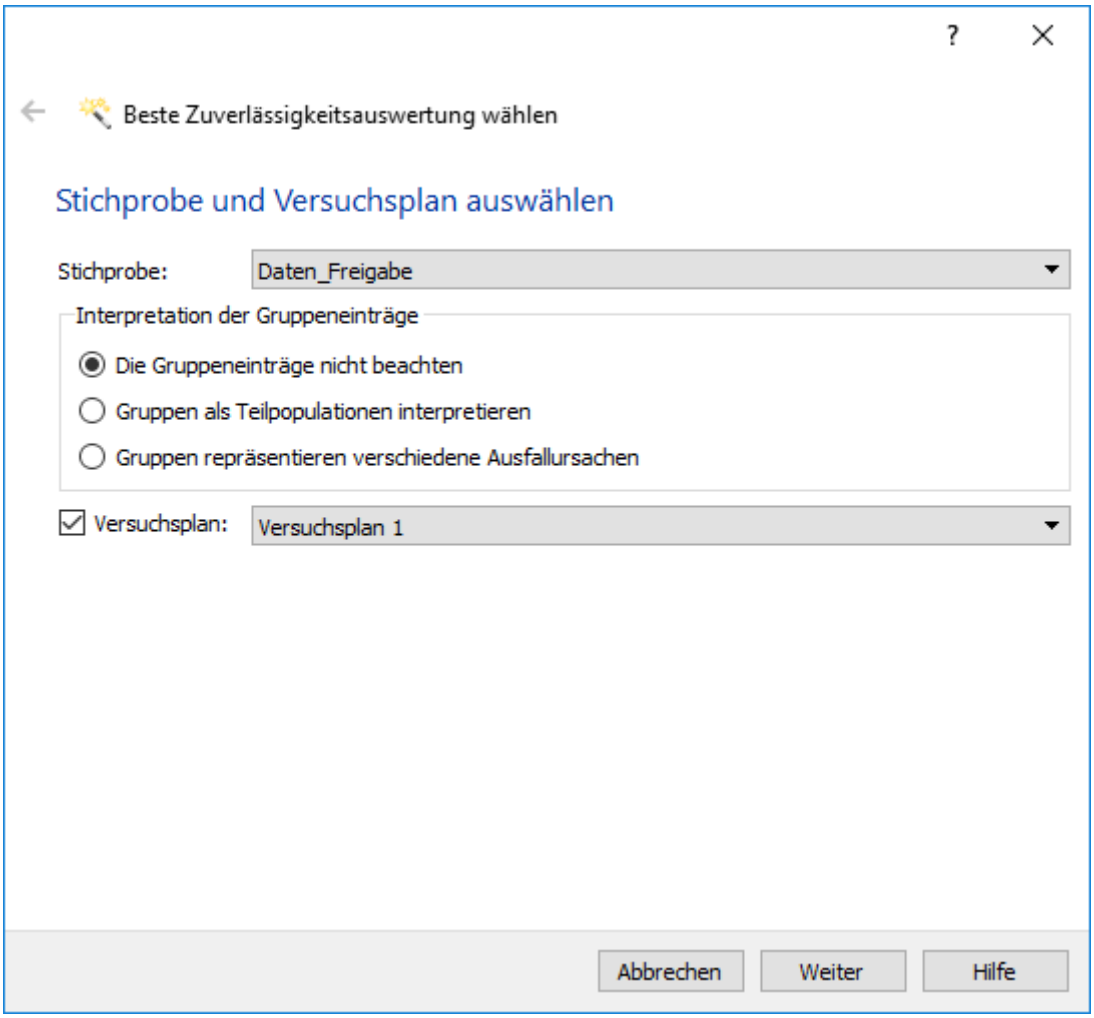

Abbildung 3 – Jurojin – Zuverlässigkeitsassistent – Seite 1

Mithilfe von den oben genannten Daten kann der Zuverlässigkeitsassistent eine Bewertung der Algorithmen vornehmen. Wie in Abbildung 4 zu sehen ist, wird die Bewertung durch Sterne symbolisiert.

- **Voller Stern**: Empfohlener Algorithmus
- **Halber Stern**: Algorithmus anwendbar, jedoch gibt es Zweifel an der numerischen Stabilität
- **Leerer Stern**: Algorithmus kann/sollte nicht angewendet werden

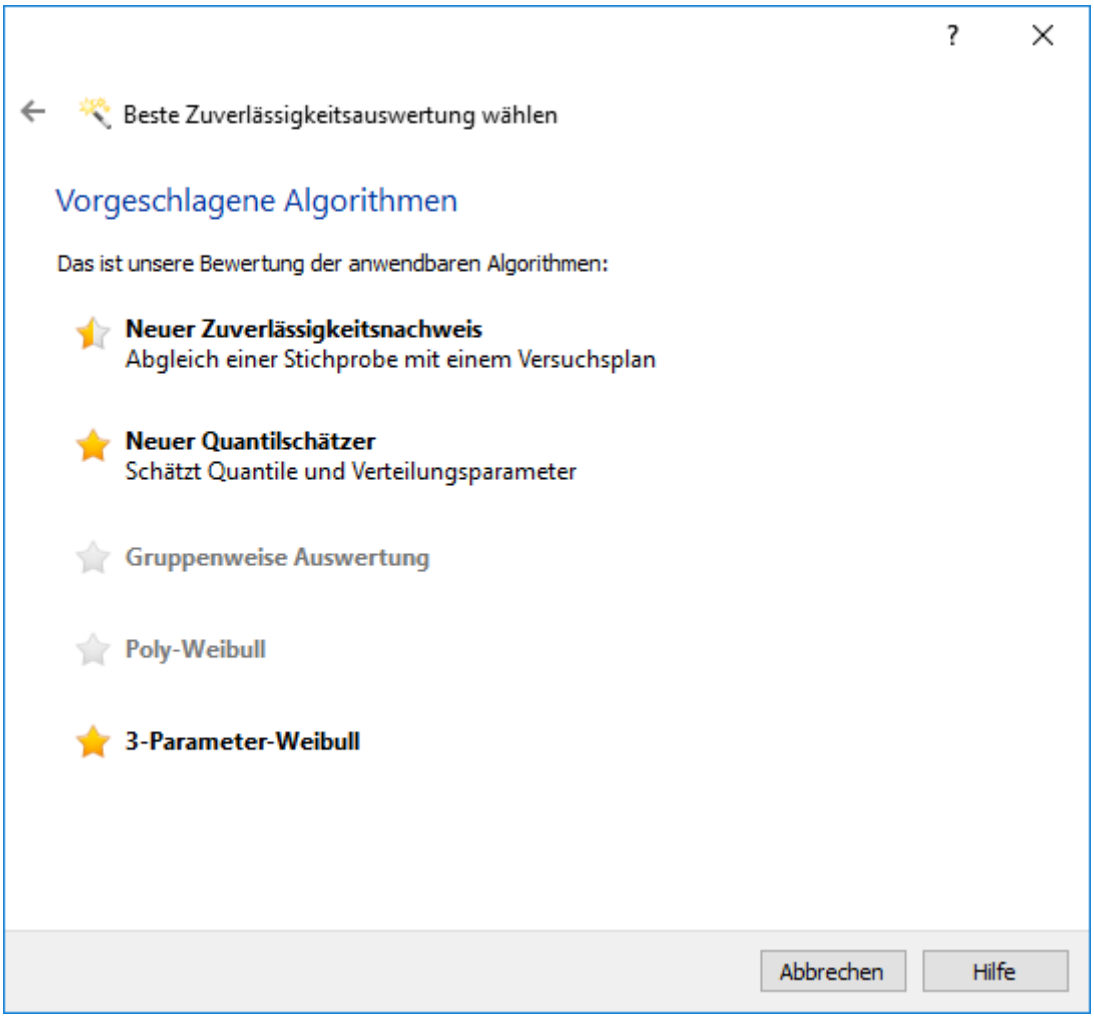

Abbildung 4 – Jurojin – Zuverlässigkeitsassistent – Seite 2

Schließlich kann sich der Nutzer für einen Algorithmus entscheiden und diesen für die Zuverlässigkeitsauswertung nutzen. Die Erstellung der jeweiligen Module erfolgt hierbei automatisch [5].

2.2 DATENBANK

Um die Eignung der Algorithmen korrekt zu bewerten, ist es notwendig die Stichprobe durch einen menschlichen Experten zu analysieren. Sollte die Stichprobe Gruppendaten enthalten, ist ebenfalls die richtige Interpretation von Anwender zu benennen. Der Versuchsplan kann optional bei der korrekten Bewertung behilflich sein Diese Informationen wurden zuvor vom Nutzer in anderen Modulen in Jurojin angelegt. Im Folgenden wird näher auf die Stichprobe und dem Versuchsplan eingegangen. Diese enthalten Daten die gesammelt und für das maschinelle Lernen verwendet werden.

2.2.1 Stichprobe

Jurojin stellt die Stichprobe als Tabelle dar. Je nach Klassifikation der Stichprobe enthalten die Spalten Wert/Lebensdauer, Ausfall/Durchläufer und Last/Gruppe. Die Anzahl der Zeilen kann über den Stichprobenumfang verändert werden. Der Nutzer kann die Daten von Hand, via Kopieren und Einfügen oder durch Importieren einer CSV-Datei eintragen. Jurojin unterscheidet zwischen folgenden Klassifikationen.

- **Stichproben**: Werte mit einer optionalen Gruppen- bzw. Chargenzugehörigkeit
- **Lebensdauerdaten**: Stichprobe zzgl. Information über Ausfall oder Durchläufer
- **Wöhler-Stichproben**: Lebensdauer, Lastniveau, sowie Information über Ausfall oder Durchläufer
- **Dauerfestigkeitsdaten**: Lastniveau und Information über Ausfall oder Durchläufer

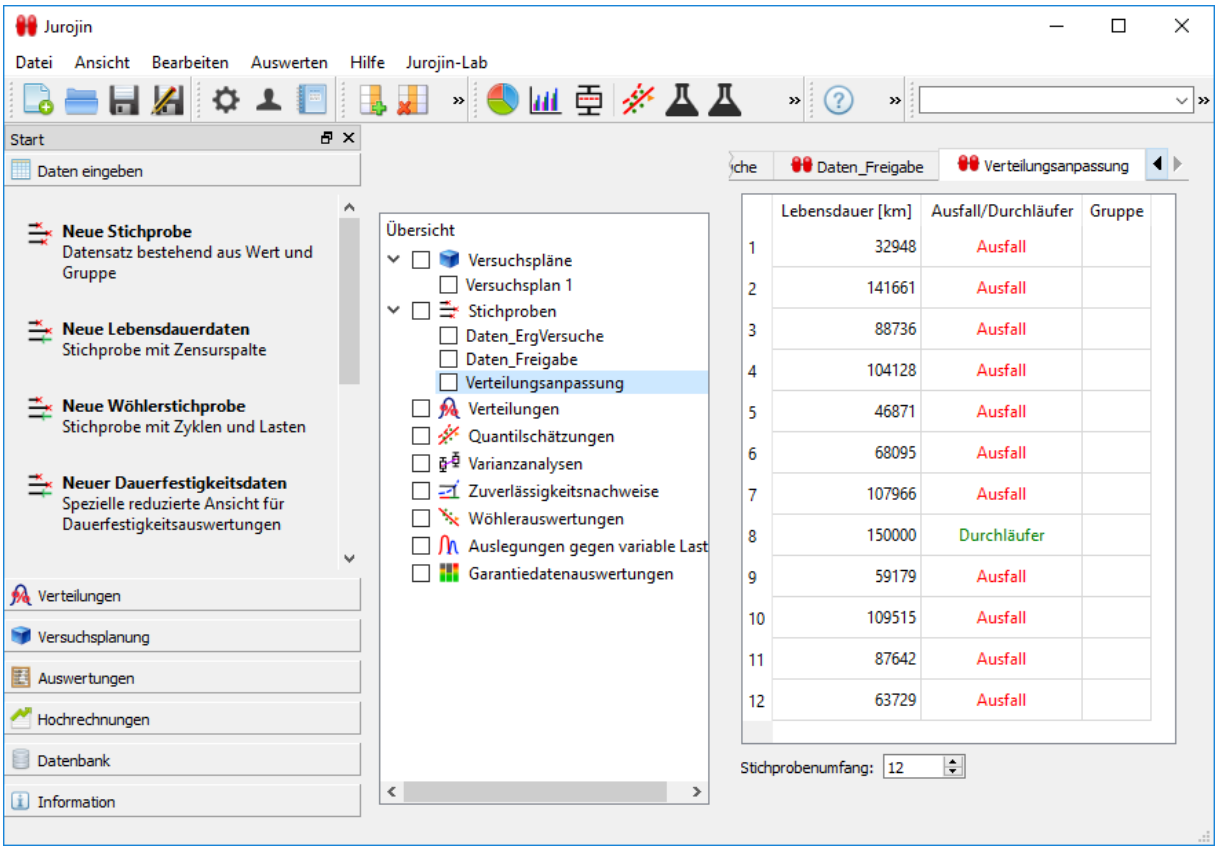

Abbildung 5 – Jurojin – Workbench (Stichprobe auf der rechten Seite)

2.2.2 Versuchsplan

Ein Versuchsplan in Jurojin ist graphisch in vier Bereiche eingeteilt: Forderung, Aufwand, Verteilungsmodell und Lastüberholung. Diese Bereiche können vom Nutzer mit den gewünschten Parametern gefüllt werden (z.B.: Es soll 99% Zuverlässigkeit nachgewiesen werden). Jurojin kann hier den Benutzer bei der Wahl einiger Werte unterstützen. Auf Knopfdruck kann dann z.B. der notwendige Stichprobenumfang für den

Zuverlässigkeitsnachweis bestimmt werden. In Abbildung 6 ist das Modul-Fenster, mit allen relevanten Feldern, zu sehen.

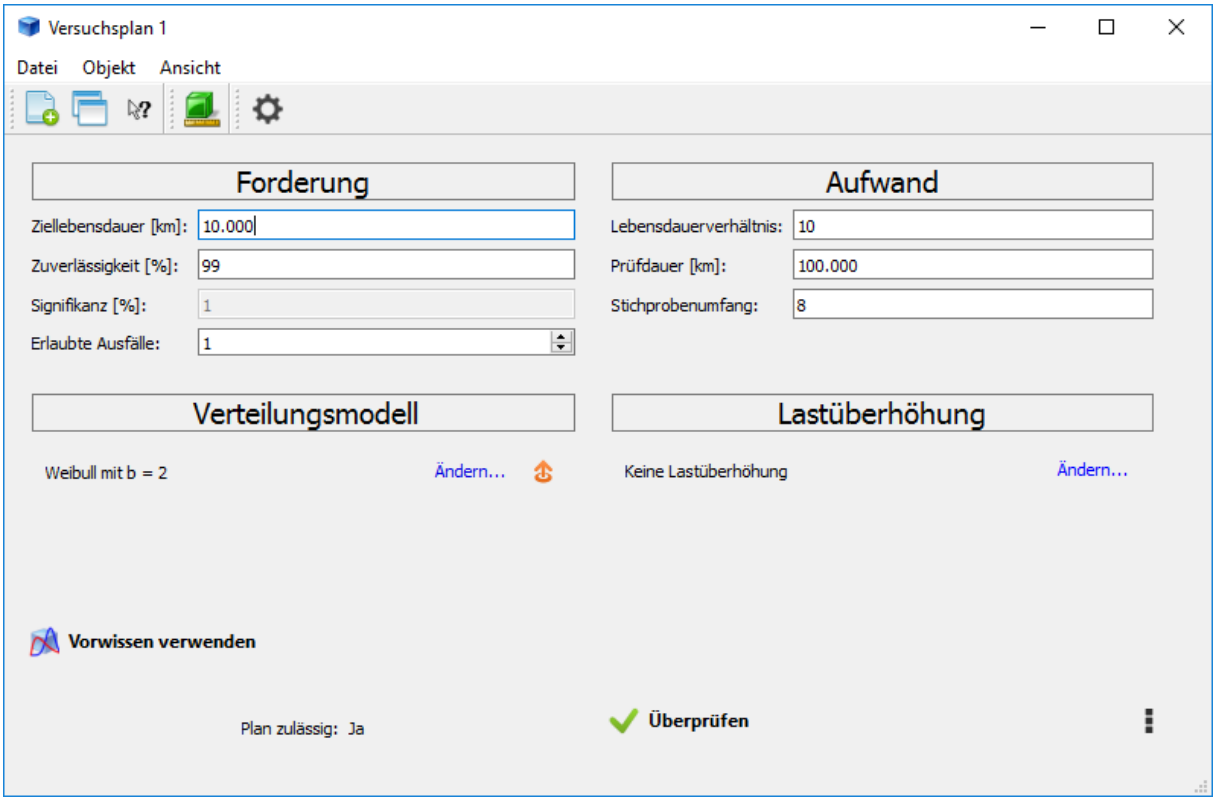

Abbildung 6 – Jurojin – Versuchsplan

2.3 Maschinelles Lernen

Maschinelles Lernen bezeichnet einen Bereich der Informatik, in dem mit Hilfe von komplexen Algorithmen eigenständige Lösungen entwickelt werden. Durch die Berechnung von Wahrscheinlichkeiten sowie das Sammeln relevanter Daten, werden Probleme gelöst und Verallgemeinerungen vorgenommen. Hier gibt es eine Großzahl bereits bewährter Algorithmen mit unterschiedlichen Vor- und Nachteilen zur Erstellung eines Modells. Es ist zu beachten, dass die Prognose eines solchen Modelles nicht immer akkurat ist. Somit stellt die Zuverlässigkeit des Modells eine entscheidende Eigenschaft dar. Im Folgenden wird näher in die Grundlagen und Begrifflichkeiten eingegangen.

2.3.1 Anwendung

Maschinelles Lernen ist vor allem in den letzten Jahren sehr populär geworden. Ob bei Bilderkennung, Spracherkennung, autonome Transportsysteme oder in der Aktienkursvorhersage. Die Anwendungsgebiete sind breit gefächert. Grundsätzlich lässt sich maschinelles Lernen nahezu überall anwenden, wo Daten gesammelt werden können.

2.3.2 Grundlagen

Um näher in das maschinelle Lernen einzugehen, ist das Verständnis einiger Grundlagen notwendig. Vor allem für die Wahl eines geeigneten Algorithmus ist die Unterscheidung der jeweiligen Datentypen notwendig, da nicht alle Algorithmen auf alle Datentypen anwendbar sind. Aber auch die Skalierung der Werte spielt eine entscheidende Rolle. Ebenso ist zu unterscheiden mit welcher Vorhersagemethode unser Algorithmus Prognosen vornehmen soll.

2.3.2.1 Skalenniveaus

Die Daten, die das maschinelle Lernen verwendet, besitzen unterschiedliche Skalenniveaus. Je nachdem wie hoch der Informationsgehalt der Daten ist, sind unterschiedliche Skalierungen und arithmetische Operationen möglich. Das Modell muss mit den passenden Typen trainieren, um ein fehlerfreies und effizientes Training zu gewährleisten.

Nominalskala

Die Nominalskala ist das Skalenniveau mit dem geringsten Informationsgehalt. Hier lässt sich ausschließlich schlussfolgern, ob zwei Einheiten sich gleichen oder nicht. Ein Beispiel hierfür

wäre die Haarfarbe von Personen. Es lässt sich zwar aussagen ob eine Person blond oder brünett ist jedoch lässt sich keine Ordnung erkennen, da blond nicht besser oder großer brünett ist. Auch andere mathematische Operationen wie Addition oder Multiplikation ergeben hier keinen Sinn.

Ordinalskala

Bei diesem Typ lässt sich zusätzlich zur Nominalskala eine Ordnung erkennen. Ein beliebtes Beispiel hierfür sind Schulnoten. Es lässt sich überprüfen welche Schulnote ein Schüler erreicht hat und zusätzlich lässt sich überprüfen welche Note besser ist und welche schlechter. Aber auch hier ergeben andere mathematische Operationen wie Addition oder Multiplikation keinen Sinn.

Intervallskala

Der dritte Typ erweitert die Ordinalskala mit der Möglichkeit Additionen und Subtraktionen auszuführen. Wie der Name Intervall schon sagt, handelt es sich hierbei meist um Längen und Abstände. Der Zeitpunkt des Kaufs eines Produktes ist hierfür ein gutes Beispiel. Es lässt sich aussagen ob das Produkt in einem gewissen Zeitpunkt gekauft wurde, ob es vor oder nach einem Zeitpunkt gekauft wurde und es lässt sich errechnen wie viele Tage zwischen 2 Zeitpunkten liegen. Jedoch lassen sich keine Verhältnisse bei den Zeitpunkten erkennen, da Beispielsweise die Verdopplung eines Zeitpunktes keinen Sinn.

Verhältnisskala

Die Verhältnisskala ist das Skalenniveau mit dem größten Informationsgehalt. Erweiternd zur Intervallskala lassen sich hier Multiplikationen und Divisionen sinnvoll durchführen. Ein Beispiel wäre ein Konto mit Geldeinheiten. Es lässt sich problemlos bestimmen ob ein gewisser Geldbetrag vorliegt oder nicht, es lässt sich Aussagen ob ein Betrag größer oder kleiner ist als ein anderer Betrag ist, die Beträge lassen sich addieren und subtrahieren und eine Verdopplung oder Halbierung des Betrages lässt sich ebenfalls sinnvoll interpretieren. Der Unterschied zur Intervallskala liegt hier also bei der Eigenschaft des absoluten Nullpunkts.

Kardinalskala

Da viele Algorithmen nicht zwischen Intervall- und Verhältnisskala unterscheiden, werden diese häufig als Kardinalskala zusammengefasst.

2.3.2.2 Vorhersagemethoden

Bei der Wahl eines passenden Algorithmus für maschinelles lernen ist es notwendig zu unterscheiden welchen Typ die prognostizierte Variable hat. Man unterscheidet hier zwischen Klassifikation und Regression.

Regression

Die Regression wird dann verwendet, sobald bei der Vorhersage ein kontinuierlicher Wert erwartet wird. Hierbei wird grundsätzlich versucht eine Funktion zu finden, welche die eingegebenen Eigenschaften verrechnet, um eine passende Vorhersage zu bestimmen. Es ist zu beachten, dass die Regression numerische Daten benötigt.

Beispielsweise wird bei der Schätzung vom Körpergewicht durch andere Körpermerkmale ein numerischer Wert erwartet.

Klassifikation

Unter der Klassifikation versteht man grundsätzlich die Vorhersage von Gruppenzugehörigkeiten. Hierbei werden unterschiedliche Klassen definiert. Der Algorithmus versucht bei der Vorhersage die richtige Klasse zu prognostizieren. Es lassen sich jedoch auch kontinuierliche Vorhersagen in Klassen unterteilen, hier muss aber eine korrekte Aufteilung der Intervalle erfolgen, was in Kosten der Genauigkeit kommt und sich somit die alleinige Nutzung der Regression empfiehlt.

Beispielsweise werden bei der Fahrzeugerkennung zwischen den Klassen Pkw, Lkw, Motorrad, Fahrrad, etc. unterschieden.

2.3.3 Lernmethoden

Die Art des Lernens funktioniert je nach Algorithmus unterschiedlich. Je nachdem welche Daten zur Verfügung stehen sind andere Algorithmen anwendbar. Grundsätzlich unterscheidet man zwischen überwachten und unüberwachten Lernen.

2.3.3.1 Überwachtes lernen

Bei dieser Lernmethode enthalten die Trainingsdaten sowohl die Eingabedaten als auch die Ausgabedaten. Der Algorithmus versucht, mithilfe dieser Daten, Assoziationen zu erkennen. Da bei den Trainingsdaten die Ausgabe bekannt ist, lassen sich mangelhafte Assoziationen präventiv erkennen. Im Rahmen dieser Bachelorarbeit ist nur diese Lernmethode von Interesse.

Einer der beliebtesten Beispiele für überwachtes lernen ist die Handschrifterkennung. Hier sind sowohl Daten für die jeweiligen Zeichen enthalten als auch die dazu passende Klassifikation.

2.3.3.2 Unüberwachtes lernen

Im Gegensatz zum überwachten Lernen enthalten beim unüberwachten Training die Trainingsdaten nur die Eingabedaten. Hierbei versucht der Algorithmus die Daten in unterschiedliche Gruppen einzuteilen. Dies wird durch das Erkennen von Eingabemustern ermöglicht.

Ein Beispiel für Unüberwachtes Lernen ist der k-Means-Algorithmus. Hier wird versucht Daten in k Gruppen zu unterteilen. Da die Trainingsdaten keine Ausgabedaten besitzen, muss der Algorithmus die Gruppierung nur mithilfe der Eingabedaten vornehmen.

2.3.4 Testen

Zum Testen des trainierten Modells ist es notwendig die Daten in Trainings- und Testdaten aufzuteilen. Während die Trainingsdaten zum Trainieren des Modells verwendet werden, werden die Testdaten zum Testen verwendet. Dies erfolgt beispielsweise durch das Zufallsprinzip. Es ist zu beachten das die Testdaten unabhängig von den Trainingsdaten sind und das trainierte Modell diese nicht zum Lernen verwendet. Mithilfe der Testdaten kann das trainierte Modell getestet werden und somit bestimmt werden welche Tests richtig und welchen falsch prognostiziert werden. Somit lassen sich Schlussfolgerungen über die Zuverlässigkeit eines solchen Modells treffen.

Abbildung 7 - Aufteilung der Daten in Trainings- und Testdaten

3 UMSETZUNG

3.1 JUROJIN

Jurojin benötigt zwei Erweiterungen. Zum einen muss Jurojin dem Nutzer eine Möglichkeit zum Exportieren von Trainingsdaten geben. Zum anderen benötigt die Software eine Implementierung, um mit dem Web-Service kommunizieren zu können.

3.1.1 Export von Daten

Um Daten für das Training des Modells, welches maschinelles Lernen verwendet, sammeln zu können, muss Jurojin den Nutzer beim Exportieren der Daten unterstützen. Dies soll im Zuverlässigkeitsassistenten durch eine Erweiterung der Menüführung geschehen. Sollte der Nutzer mit der Bewertung des Zuverlässigkeitsassistenten unzufrieden sein, muss eine Option zur Anpassung der Bewertung mit anschießendem Export angeboten werden. Dabei sollte die Implementierung konsistent zur bisherigen Menüführung sein, um den Nutzer eine intuitive Lösung zu bieten. Abbildung 8 und Abbildung 9 zeigen die gewählte Lösung. Hier kann der Nutzer, falls unzufrieden mit der Bewertung, das blaue Label anklicken und anschließend per Klick die Bewertung anpassen. Danach muss nur noch ein Verzeichnis zum Exportieren gewählt werden.

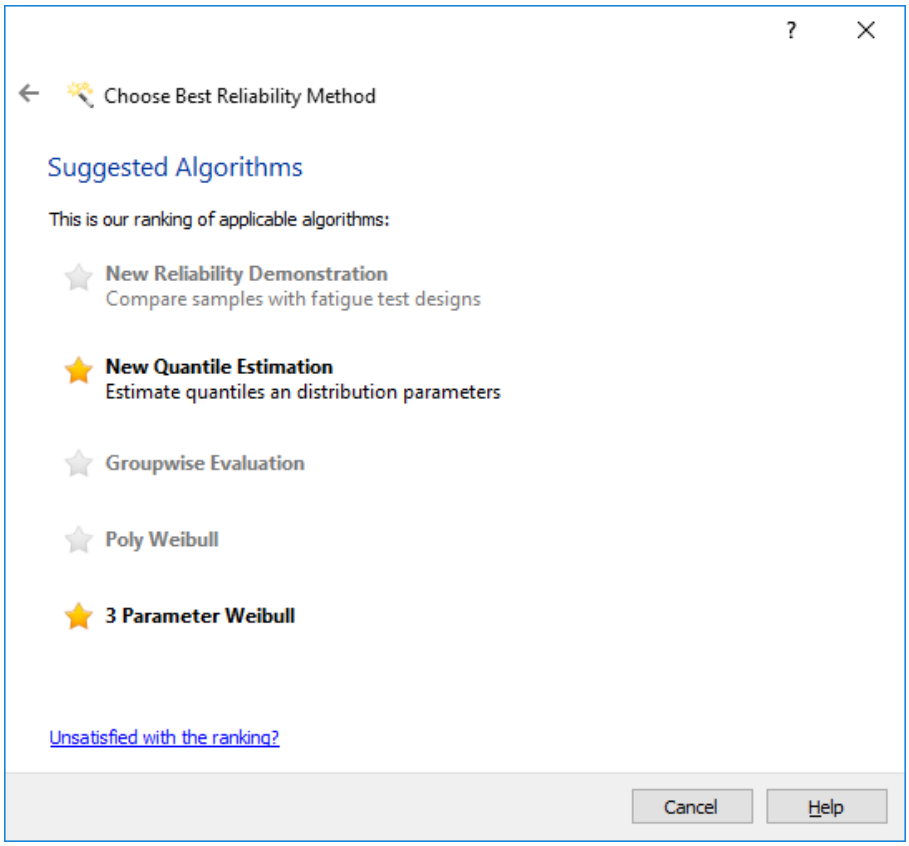

Abbildung 8 - Zuverlässigkeitsassistent mit Label zur Anpassung

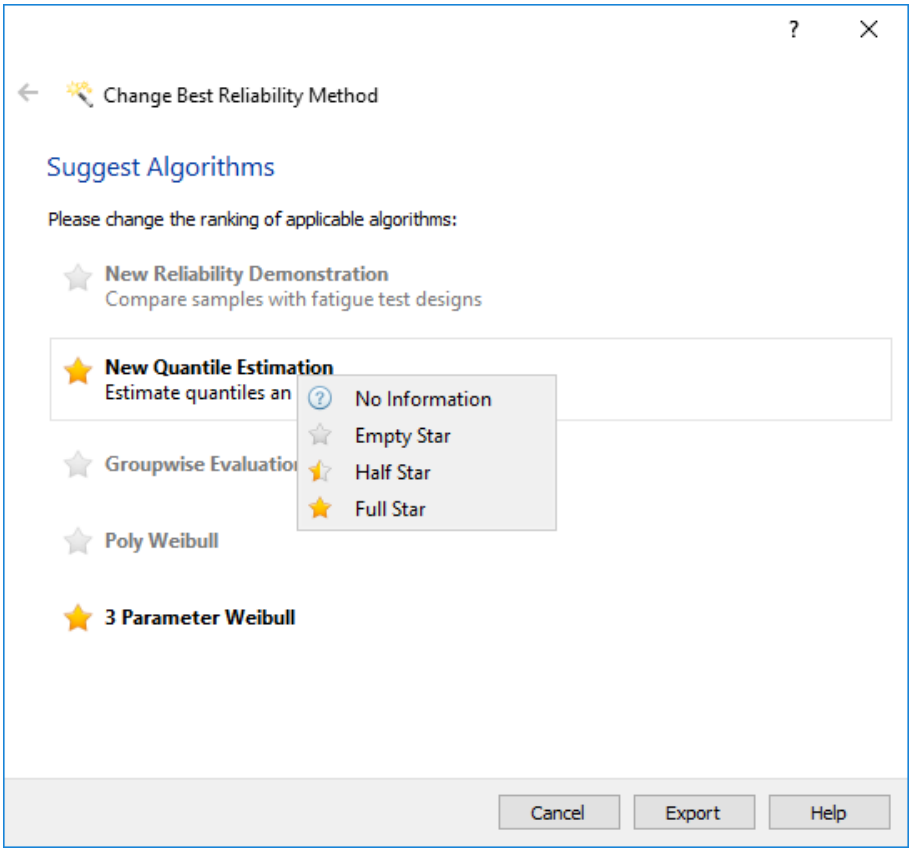

Abbildung 9 - Zuverlässigkeitsassistent - Menüführung Anpassung der Bewertung mit Exportmöglichkeit

3.1.2 Anbindung Web-Service

Da der Zuverlässigkeitsassistent keine Änderungen an der Menüführung erhält, bleibt das Verhalten des Nutzers unverändert. Jedoch sollte der Web-Service nur für Nutzer mit einer gültigen Jurojin-Lab-Lizenz zur Verfügung stehen. Hierfür muss von Jurojin unterschieden werden ob eine solche Lizenz zur Verfügung steht. Sollte der Nutzer eine Jurojin-Lab-Lizenz besitzen, wird der Web-Service zur Bewertung verwendet. Andernfalls wird die Ursprüngliche Offline-Variante verwendet.

3.2 DATENBANK

Um einen Entscheidungsbaum, mithilfe von maschinellen lernen, trainieren zu können werden Trainingsdaten benötigt. Da stellen sich mehrere Probleme. Unter anderem muss entschieden werden welche Daten benötigt bzw. relevant sind. Außerdem muss sich für ein passendes Format der Daten entschieden werden, unter welche diese gespeichert werden. Ein weiteres Problem ist die variable Länge der Stichprobe. Beim maschinellen Lernen ist eine konstante Länge der abhängigen Variablen zunächst erforderlich, hierfür muss eine Lösung gefunden werden. Es muss aber auch entschieden werden, wie die Daten seitens Jurojin bereitgestellt werden.

3.2.1 Strategien und Möglichkeiten

Es bieten sich mehrere Optionen für eine geeignete Datenstruktur. Ein Trainingsdatum besteht aus Anwendersicht aus Stichprobe, optionalen Versuchsplan, usw. und einer dreistufigen Bewertung. Jedoch ist zu beachten, dass die Stichprobe eine variable Länge besitzt. Hier muss entschieden werden wie mit der variablen Länge nun umgegangen werden muss. Des Weiteren stellt sich die Frage welche Daten nun gesammelt werden sollen. Außerdem muss ein passendes Datenformat gewählt werden.

3.2.1.1 Umgang der variablen Länge der Stichproben

Da maschinelles Lernen eine konstante Anzahl an abhängigen Variablen bzw. Merkmalen benötigt, ist es nicht möglich Stichproben mit variabler Länge zum Trainieren zu verwenden. Beim Versuchsplan ist dies kein Problem, da diese eine fixe Anzahl an Merkmalen besitzt. Zum Lösen des Problems gibt es mehrere Ansätze. Eine Lösung wäre es eine Fixe Größe der Stichprobe zu definieren. Jedoch zieht dies weitere Probleme mit sich. Zum einen muss eine Strategie zum Vergrößern der zu kleinen Stichproben und eine Strategie zum Verkleinern der zu großen Stichproben gefunden werden. Aus Sicht der Statistik könnte es jedoch ausreichen, die Stichprobe stellvertretend durch einige deskriptive Merkmale zu vertreten. Da das durch den Entscheidungsbaum trainierte Modell sehr transparent ist, lässt sich durch

Testen verschiedenster Merkmale herausfinden, welche nun die Stichprobe am besten vertreten. Somit neigen Merkmale, die sich näher an der Wurzel befinden dazu, sich besser zu eignen als Merkmale, die sich weit weg von der Wurzel befinden. Folgende Merkmale haben sich durch dieses Vorgehen bewährt.

- **Größe der Stichprobe:** Dies entspricht der Anzahl an Einträgen in der Stichprobe.
- **Verhältnis der Zensurdaten:** Dieses Merkmal beschreibt das Verhältnis, zu dem die Zensurdaten aufgeteilt wurden. Hier müssen die Durchläufer- und Ausfalldaten verrechnet werden.
- **Gruppen mit ausschließlich Durchläufern:** Einige Methoden benötigen von der Stichprobe die Auskunft, ob sich Gruppen finden, deren Zensurdaten ausschließlich aus Durchläufern bestehen.
- **Anzahl der Gruppen:** Hier ist es lediglich notwendig die Anzahl der unterschiedlichen Gruppen zu bestimmen.
- **Gruppenverhältnis:** Dieses Merkmal ist die Abwägung der Abdeckung der Gruppendaten. Eine Stichprobe mit einer großen Anzahl an Einträgen in einer Gruppe und einer kleinen Anzahl an Einträgen in der anderen Gruppe gibt eine Entscheidende Aussage über die Eignung einiger Methoden. Folgende Formel bestimmt diesen Wert: $\prod_{k=n}^{n} a_n \cdot \frac{b}{c}$, wobei a_n für die Anzahl der Einträge in der jeweiligen Gruppe, b für die Anzahl der unterschiedlichen Gruppen und c für die Stichprobengröße steht.
- **Klassifikation der Stichprobe:** Jurojin unterscheidet zwischen verschiedenen Klassifikationen (siehe 2.2.1).

3.2.1.2 Rohdaten als Vorstufe der Trainingsdaten

Grundsätzlich gibt es die Möglichkeit die gesamte Stichprobe zu sichern oder die Daten auf deskriptive Werte zu beschränken. Da maschinelles Lernen, in diesem Fall, nur die deskriptive Werte verwenden kann und keine beliebige Anzahl an Merkmalen möglich ist, erscheint es sinnvoll nur diese zu sammeln. Jedoch besteht dadurch das Problem, dass zukünftige Anpassungen am Algorithmus durch den Informationsverlust nicht möglich sind. Des Weiteren muss Jurojin sich darum kümmern die deskriptiven Werte zu bestimmen. Es ergibt mehr Sinn alle relevanten Daten zu sichern und diese vor dem Trainieren in deskriptive Werte umzuwandeln. Somit entstehen kein Datenverlust und alle Vorteile des Web-Services bleiben erhalten.

3.2.1.3 Datenformat

Es gibt verschiedene Datenformate, die geeignet wären, um die Daten zu sichern. Da Jurojin bereits für Stichproben eine Funktionalität zum Exportieren im CSV-Format anbietet ist es naheliegend dieses Dateiformat fortzuführen. CSV-Dateien lassen sich auf Tabellen abbilden,

welche als Input für den Algorithmus benötigt werden. Des Weiteren ist dieses Dateiformat leicht lesbar und bearbeitbar. Jurojin verwendet bei CSV-Dateien als Trennzeichen ein Semikolon, dies wird ebenfalls beibehalten. Mit Formaten, wie JSON oder XML wäre der ganze Prozess ebenfalls möglich gewesen. Zu berücksichtigen ist jedoch, dass Jurojin die Werte je nach Spracheinstellung in unterschiedlichen Zahlenformate abspeichert.

3.2.2 Überführung der Daten

Aus Gründen der Übersicht wird ein Datensatz in vier CSV-Dateien unterteilt. Neben der Stichprobe und dem Versuchsplan wird eine weitere Datei mit den Eigenschaften und eine Datei mit den Bewertungen erzeugt. Die CSV-Dateien haben folgenden Inhalt.

3.2.2.1 Stichprobe

1. Spalte: Wert/Lebensdauer (double)
2. Spalte: Ausfall/Durchläufer (int)
3. Spalte: Last/Gruppe (string)

3.2.2.2 Versuchsplan

1. Spalte: Familie des Verteilungsmodells (string)
2. Spalte: Form des Verteilungsmodells (double)
3. Spalte: Stichprobenumfang im Versuchsplan (int)
4. Spalte: Zuverlässigkeit der Forderung (double)
5. Spalte: Signifikanz der Forderung (double)
6. Spalte: Prüfdauer des Aufwands (double)
7. Spalte: Lebensdauerverhältnis des Aufwands (double)

3.2.2.3 Einstellungen

1. Spalte: Interpretation der Gruppendaten (string)
2. Spalte: Klassifikation der Stichprobe (string)
3. Spalte: Verwendete Spracheinstellung (string)

3.2.2.4 Bewertung

1. Spalte: Bewertung des Zuverlässigkeitsnachweises (int)
2. Spalte: Bewertung des Quantilschätzers (int)
3. Spalte: Bewertung der Gruppenweiser Auswertung (int)
4. Spalte: Bewertung der Auswertung durch Poly-Weibull (int)
5. Spalte: Bewertung der Auswertung durch 3-Parameter-Weibull (int)

3.2.3 Datenbeschaffung

Der bisherige Zuverlässigkeitsassistent wurde um die Möglichkeit erweitert, die oben genannten Daten zu exportieren. Somit kann der Kunde durch Zusenden der Daten die Datenbank erweitern. Da bei Beginn der Bachelorarbeit nur eine geringe Anzahl an Kundendaten zu Verfügung stehen, wurden intern weitere Daten generiert. Außerdem wurden Daten aus den bereits bestehenden Unittests und einiger bereits gesammelter Kundendaten entnommen. Bei demselben Datensatz kann es vorkommen, dass Experten unterschiedliche Bewertungen angeben. Da die Bewertungen der Algorithmen von vielen Faktoren abhängen, ist es manchmal nicht möglich eindeutig zu bestimmen, ob ein Algorithmus z.B. nur einen halben oder vollen Stern zugewiesen bekommen soll. Hier geben die initial generierten Daten eine Orientierung der zukünftigen Bewertungen vor.

3.3 Maschinelles Lernen

Bei der Wahl eines passenden Algorithmus, welche maschinelles Lernen verwendet, müssen einige Voraussetzungen erfüllt werden, auf denen im weiteren Verlauf näher eingegangen wird. Des Weiteren müssen, aufgrund der initial niedrigen Datensammlung, Strategien zum Aufteilen der Trainings- und Testdaten gefunden werden. Außerdem müssen die Ergebnisse den Umständen entsprechend korrekt validiert werden. Im Folgenden wird auf einige Algorithmen eingegangen, für welches das Buch „Maschinelles Lernen“ von Ethem Alpaydin [7] sehr aufschlussreich war.

3.3.1 Algorithmen

Es haben sich bereits eine Vielzahl an Algorithmen für maschinelles Lernen bewährt. Diese haben unterschiedliche Vor- und Nachteile und unterscheiden sich teilweise komplett in ihrem Vorgehen. Nicht alle Algorithmen eignen sich für alle Anwendungsfälle. Wenn der gewählte Algorithmus für die Daten nicht geeignet ist, können die Ergebnisse möglicherweise nicht sinnvoll interpretiert werden. Außerdem lassen sich nicht alle Algorithmen für Klassifikation und Regression verwenden und bieten somit ausschließlich

eine Vorhersagemethode an. Im Folgenden wird auf einige der bereits bewährten Algorithmen näher eingegangen.

3.3.1.1 Ungeeignete Algorithmen

Da in unserem Fall ein Algorithmus zur Klassifikation benötigt wird, werden Algorithmen, die ausschließlich zur Regression geeignet sind vernachlässigt. Hierzu gehört zum Beispiel die Lineare Regression. Des Weiteren ist es notwendig, dass die Klassifikation von 3 Klassen möglich ist. Algorithmen wie die Logistische Regression oder die Support-Vector-Machine ermöglichen beispielsweise nur die Klassifikation von 2 Klassen und fallen somit ebenfalls weg. Außerdem muss das Ergebnis nachvollziehbar sein, um passende deskriptive Werte für die Stichprobe zu finden. Somit entfallen die neuronalen Netze, da prognostizierten Ergebnisse hinsichtlich Ihrer Merkmale nicht mehr nachvollziehbar sind. Da die aktuelle Implementierung der Methodenbewertung einer Baumstruktur ähnelt, ergibt das Trainieren einer Lösung in Form eines Baumes in diesem Fall am meisten Sinn.

3.3.1.2 Entscheidungsbaum

Das Trainieren eines Entscheidungsbaums mithilfe von maschinellem Lernen ist aufgrund seiner breit gefächerten Anwendbarkeit sehr beliebt. Der Algorithmus lässt sich sowohl zur Regression als auch zur Klassifikation anwenden. Da, nach Regeln zum Aufteilen eines Baumes gesucht wird, lassen sich ordinale Merkmale verwenden. Eine große Anzahl an Merkmalen stellt ebenfalls kein Problem dar. Der trainierte Entscheidungsbaum ermöglicht eine Visualisierung, somit lassen sich beispielsweise Schlussfolgerungen über die Aussagekraft der jeweiligen Merkmale bestimmen. Das sorgt für Nachvollziehbarkeit der Ergebnisse und macht den Algorithmus somit transparenter.

Im Folgenden wird ein Beispiel, der die generelle Vorgehensweise von diesem Algorithmus zeigt, veranschaulicht. Hierbei wurden Daten gesammelt bei denen Eltern entschieden haben, ob Sie Ihre Kinder zum Spielen rausschicken sollen, unter den Merkmalen Wetter, Temperatur und Windstärke.

Wetter	Temperatur	Windstärke	Raus gehen
Regnerisch	Mild	Stark	Nein
Sonnig	Heiß	Schwach	Ja
Bewölkt	Mild	Schwach	Ja
Regnerisch	Kalt	Normal	Nein
Sonnig	Mild	Schwach	Ja
Bewölkt	Kalt	Normal	Nein
Sonnig	Heiß	Stark	Nein

Sonnig	Heiß	Normal	Ja
Regnerisch	Heiß	Schwach	Nein

Tabelle 1 - Daten - Raus gehen

Die dargestellte Tabelle zeigt Trainingsdaten auf, hierbei wird in der letzten Spalte entschieden ob man seine Kinder nach draußen zum Spielen schicken sollte. Für das Erstellen eines Entscheidungsbaum müssen zuerst die Merkmale Wetter, Temperatur und Windstärke aufgeteilt werden. Dies erfolgt durch die Aufteilung der unterschiedlichen Werte in Untertabellen (numerische Werte werden in Intervalle unterteilt). Dies wird in den Untertabellen für die restlichen Merkmale so lange wiederholt, bis die Ergebnisspalte in allen Untertabellen keine unterschiedlichen Werte besitzt. Dabei spielt die Wahl des nächst zu teilendem Merkmal eine entscheidende Rolle, da durch die richtige Reihenfolge weniger Untertabellen erstellt werden müssen und somit ein optimaler Entscheidungsbaum näher angestrebt wird.

Wetter	Temperatur	Windstärke	Raus gehen
Regnerisch	Mild	Stark	Nein
Regnerisch	Kalt	Normal	Nein
Regnerisch	Heiß	Schwach	Nein

Wetter	Temperatur	Windstärke	Raus gehen
Sonnig	Heiß	Schwach	Ja
Sonnig	Mild	Schwach	Ja
Sonnig	Heiß	Stark	Nein
Sonnig	Heiß	Normal	Ja

Wetter	Temperatur	Windstärke	Raus gehen
Bewölkt	Mild	Schwach	Ja
Bewölkt	Kalt	Normal	Nein

Tabelle 2 - Beispiel Aufteilung Untertabellen (Wetter)

Dies erfolgt durch die Wahl des Merkmals mit der geringsten Streuung. Hierfür gibt es mehrere Verfahren, ein Verfahren ist mittels Gini-Index. Dies misst den Grad oder die Wahrscheinlichkeit, dass eine bestimmte Variable falsch klassifiziert wird, wenn sie zufällig ausgewählt wird. Die Messung erfolgt mithilfe folgender Formel, wobei p_i für die Wahrscheinlichkeit steht, dass ein Objekt in eine bestimmte Klasse eingestuft wird. Es gibt weitere Verfahren, jedoch bieten diese für das Erstellen eines Entscheidungsbaumes keine signifikanten Unterschiede.

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Am Ende werden die Daten der Untertabellen verwendet, um einen Entscheidungsbaum zu generieren. Eine wichtige Eigenschaft des erstellten Baumes ist, dass die Merkmale die näher zur Wurzel sind, wie oben bereits beschrieben, eine höhere Relevanz besitzen. Der generierte Baum kann nun für die Prognose ausgewertet werden.

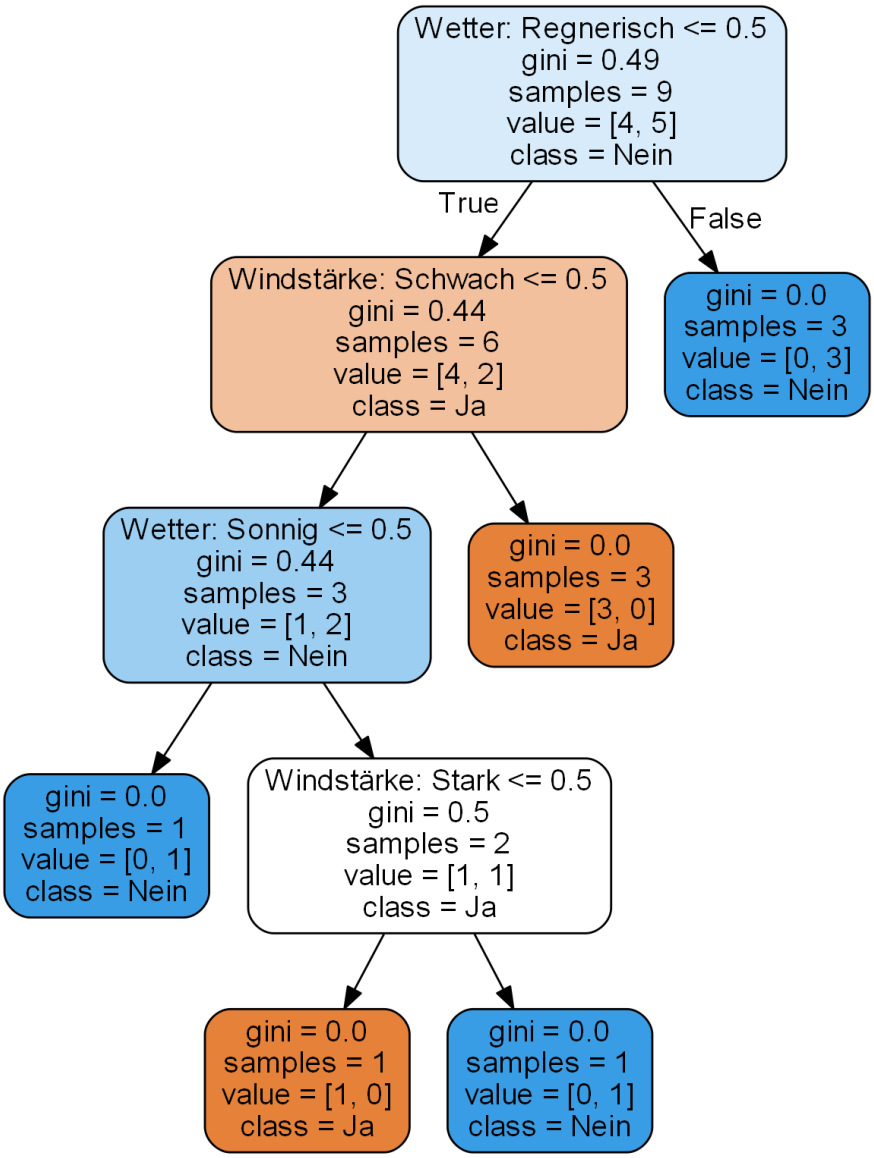

Abbildung 10 – Anwendungsbeispiel Entscheidungsbaum mit Aufteilung durch Gini-Index

3.3.1.3 Random Forrest

Der Algorithmus Random Forest ist eine Erweiterung des Entscheidungsbaumes mit dem Ziel die Zuverlässigkeit mithilfe von Randomisierung zu erhöhen. Die Lernphase ist hier teilweise analog zum Entscheidungsbaum. Ein Unterschied ist, dass neue Trainingsdaten erstellt werden. Dies geschieht durch das zufällige Auswählen von Dateneinträgen aus den alten Trainingsdaten. Es werden so lange Datensätze rausgesucht, bis die Trainingsdaten dieselbe

Anzahl an Einträge hat wie die alten Trainingsdaten. Hierbei ist zu beachten, dass dieselben Dateneinträge mehrfach ausgewählt werden können und einige Dateneinträge gar nicht erst zum Training ausgewählt werden. Das Weitern wird nur eine beliebige Anzahl der Merkmale verwendet, jedoch nie alle Merkmale. Der daraus resultierende Baum entscheidet sich erheblich vom Baum des ursprünglichen Algorithmus und ist weit davon entfernt optimal zu sein. Dies führt dazu, dass sich nicht mehr im selben Maß wie beim klassischen Entscheidungsbaum erkennen lässt, welche Aussagekraft die jeweiligen Merkmale besitzen. Zu guter Letzt wird klar, warum dieser Algorithmus Random Forrest genannt wird, denn es werden mehrere dieser Bäume erzeugt. Somit entsteht ein „Wald“ an „Bäumen“ die zufällig generiert wurden. Sollte man nun einen Datensatz verwenden, um ein Ergebnis zu prognostizieren, werden alle Bäume ausgewertet und sich für die Klassifikation mit den meisten Ergebnissen entschieden.

3.3.2 Auswahl des passenden Algorithmus

Wie bereits beschrieben ist ein Algorithmus, welches einen Baum mithilfe von maschinellem Lernen trainiert, am Sinnvollsten. Die Algorithmen Entscheidungsbaum und Random Forest sind hier am naheliegendsten. Da der Random Forest eine Erweiterung des Entscheidungsbaumes ist und zuverlässigere Ergebnisse liefen soll, liegt es nahe diesen zu wählen. Jedoch ist zu beachten, dass dies in unserem Fall nicht unbedingt zutrifft. Es stimmt, dass die Ergebnisse besser sind, jedoch benötigt der Algorithmus auch wesentlich mehr Trainingsdaten, um dies zu erreichen. Da erst in Zukunft eine große Menge an Kundendaten gesammelt werden muss, besitzen wir am Anfang nur eine überschaubare Anzahl an Daten. Somit ist der Entscheidungsbaum eher geeignet. Des Weiteren lässt sich aus den generierten Bäumen des Random Forest nicht in selben Maß nachvollziehen welche Merkmale für das Ergebnis die größte Gewichtung haben. Dies ist zum einen für das Erstellen von Merkmalen aus der Stichprobe von Bedeutung, zum anderen für die Anpassung der bestehenden Implementation notwendig. Somit fällt die Wahl auf den Entscheidungsbaum.

3.3.3 Datentransformation

Die Daten, die für das maschinelle Lernen genutzt werden, können nicht immer ohne weitere Datenanpassung verwendet werden. Hier ist es je nach Algorithmus notwendig die „Rohen“ Merkmale passend zu Skalieren oder anders abzubilden. Im Folgenden betrachten wir die Datenanpassung näher im Falle des Trainings mithilfe eines Entscheidungsbaumes.

3.3.3.1 Skalierung

Eine Großzahl an Algorithmen benötigt eine Skalierung der Merkmale, um diese zum Trainieren verwenden zu können. Der Entscheidungsbaum benötigt keine Skalierung. Das ist

darin begründet, da beim Erstellen eines solchen Baumes nach passenden Regeln gesucht wird und somit die Notwendigkeit der Skalierung entfällt.

3.3.3.2 Konvertierung von kategorialer Daten

Kategoriale Daten müssen für die meisten Algorithmen in numerische Daten umgewandelt werden. Enthalten die Trainingsdaten beispielsweise das Merkmal Farbe, kann man einfach jeder Farbe einen Wert zuteilen. Hier bekommt Rot beispielsweise den Wert 0, Gelb den Wert 1 und grün 2. Somit steht Rot Beispielsweise für „Schlecht“ und grün für „Gut“ (Siehe Tabelle 3 und Tabelle 4). Für Kategoriale Daten, welche keine ordinale Beziehung besitzen, ist jedoch ein anderes Vorgehen notwendig. In diesem Fall ist es notwendig das Merkmal Farbe in weitere Merkmale aufzuteilen. Somit stehen Rot, Gelb und Grün jeweils für ein Merkmal, wobei der Wert Beispielsweise 0 für trifft, nicht zu und 1 für trifft zu steht (Siehe Tabelle 3 und Tabelle 5).

Farbe
Rot
Grün
Gelb
Rot

Tabelle 3 - Daten - Farbe

Farbe
0
2
1
0

Tabelle 4 - Daten - Farbe - Umgewandelt als integer

Rot	Grün	Gelb
1	0	0
0	1	0
0	0	1
1	0	0

Tabelle 5 - Daten Farbe – In mehrere Merkmale aufgeteilt

3.3.4 Validierung

Da aus der Stichprobe Merkmale entnommen werden müssen, ist es notwendig beurteilen zu können aus welchen Merkmalen ein zuverlässigeres Ergebnis resultiert. Hierzu ist es Notwendig die Zuverlässigkeit des Algorithmus zu ermitteln. Es gibt bereits Verfahren, um dies zu bestimmen, jedoch reicht dies nicht unbedingt aus. Da am Anfang nur eine kleine Menge an Trainingsdaten zur Verfügung steht, benötigen wir Verfahren, die ohne eine große Aufteilung an Testdaten auskommt. Außerdem sind im Falle des Zuverlässigkeitsassistenten Bewertungen, die um einen halben Stern daneben liegen nicht unbedingt falsch, da hier ein Interpretationsfreiraum vorliegt. Dies muss bei den Testverfahren und bei der Auswertung der Ergebnisse berücksichtigt werden.

3.3.4.1 Konfusionsmatrix

Da wir einen gewissen Interpretationsfreiraum besitzen und somit falsch prognostizierte Ergebnisse nicht unbedingt falsch sind, wenn diese um einen halben Stern daneben liegen, benötigen wir eine Darstellung der Auswertung, welche dies berücksichtigen kann. Hierfür eignet sich die Konfusionsmatrix.

		Tatsächliche Klasse	
		Positiv	**Negativ**
Ermittelte Klasse	**Positiv**	**TP**	FP
	Negativ	FN	**TN**

Abbildung 11 - Konfusionsmatrix

Die Zeilen repräsentieren bei der oben dargestellten Abbildung die prognostizierten Klassen und die Spalten die Klassen aus den Testdaten. Dadurch lässt sich in jeder Zelle bestimmen, welches Ergebnis prognostiziert und welches erwartet wurde. TP steht hier true-positive, TN für true-negative, FN für false-negative und FP für false-positive. Generell bedeutet das, dass optimalerweise große Werte in der Diagonalen zu erreichen ist. Eine Konfusionsmatrix mit ausschließlich Werten in der Diagonalen und Nullen außerhalb der Diagonalen ist somit ein perfektes Ergebnis.

		Tatsächliche Klasse		
		Leerer Stern	**Halber Stern**	**Voller Stern**
Ermittelte Klasse	**Leerer Stern**	**20**	0	1
	Halber Stern	1	**13**	4
	Voller Stern	0	2	**17**

Abbildung 12 - Konfusionsmatrix - Beispiel

Abbildung 12 zeigt nun ein Beispiel für eine Konfusionsmatrix, die als mögliche Klassen einen leeren, halben und vollen Stern darstellt. In Falle von diesem Beispiel, erlauben wir jedoch einen Interpretationsfreiraum von einem halben Stern. Somit sind Ergebnisse, die um einen halben Stern daneben liegen nicht unbedingt falsch. Jedoch sind Prognosen, die um einen ganzen Stern daneben liegen immer falsch. Der grün dargestellte Bereich zeigt somit Richtige Prognosen auf, während der gelbe Bereich Prognosen darstellt, die um einen halben Stern daneben liegen und der rote Bereich zeigt Schätzungen auf, die um einen ganzen Stern verschätzt wurden.

3.3.4.2 Kreuzvalidierung

Bei der Kreuzvalidierung werden k Durchläufe vorgenommen, wobei k nicht größer als die Anzahl der Daten sein darf. Es wird versucht die Daten in k möglichst gleich große Teilmengen aufzuteilen und diese als Testdaten zu verwenden. Bei jedem Durchlauf wird nun ein neues Modell mit den jeweiligen Trainingsdaten trainiert und mit den jeweiligen Testdaten getestet. Am Ende wird der Mittelwert der Ergebnisse bestimmt. Abbildung 13 veranschaulicht dieses Prozedere.

		Datenaufteilungen				
		1	2	3	4	5
Durchläufe	1	Test	Training	Training	Training	Training
	2	Training	Test	Training	Training	Training
	3	Training	Training	Test	Training	Training
	4	Training	Training	Training	Test	Training
	5	Training	Training	Training	Training	Test

Abbildung 13 - Kreuzvalidierung mit 5 Datenaufteilungen

Bei großen Datenmengen ist es unproblematisch einen Teil der Daten (z.B. 25%) zum Testen des trainierten Modells zu verwenden. Bei der Kreuzvalidierung ist es jedoch möglich den Anteil der Testdaten problemlos zu verkleinern. Bei der Leave-One-Out-Kreuzvalidierung werden die Testdaten so gering wie möglich gehalten. Hier entspricht k der Anzahl der vorhandenen Daten. Somit wird pro Durchlauf nur ein Datum für das Testen verwendet. Da wie oben beschrieben nur eine geringe Anzahl an Daten zur Verfügung stehen und somit das Nutzen von einem großen Anteil an Testdaten, für das trainierte Modell ein wesentlich ungenaues Ergebnis verursacht, liefert dieses Verfahren ein repräsentativeres Ergebnis. Es ist zu beachten, dass die Anzahl der Daten der Anzahl der zu trainierenden Modelle entspricht und dies vor allem mit wachsenden Daten einen großen Rechenaufwand mit sich zieht. Dies lässt sich jedoch mit der Verkleinerung von k kompensieren.

4 IMPLEMENTIERUNG

Dieses Kapitel beschäftigt sich mit der Realisierung der Softwarelösung als Web-Service. Ebenfalls wurde Jurojin derart erweitert, dass vom Benutzer Trainingsdaten exportiert werden können und der Software eine Kommunikation mit den Web-Service ermöglicht wird. Da wie in der Einleitung bereits erläutert, der Nutzer so wenig wie möglich von der Änderung erfahren soll, erhält die Menüführung in Jurojin kaum Änderungen.

4.1 JUROJIN

Jurojin verwendet eine Vielzahl an Bibliotheken. In diesem Fall sind *Qt* [8], *AWS SDK für C++* [9], *libcurl* [10] und *JSON for modern C++* [11] relevant. Diese wurden verwendet, um die notwendigen Anpassungen vorzunehmen.

4.1.1 Exportierung benutzerdefinierter Daten

Der Export der benutzerdefinierten Daten wurde genau wie in 3.1.1 (Export von Daten) beschrieben implementiert. Nachdem die Anpassung der Bewertungen vorgenommen wurde kann durch einen Klick auf Export ein Verzeichnis ausgewählt um die Daten wie in 3.2.2 (Überführung der Daten) beschrieben, als CSV-Dateien exportiert werden.

4.1.2 Verwendung des Web-Services

Der Web-Service wird nur bei einer gültigen Jurojin-Lab-Lizenz verwendet. Jurojin wandelt alle relevanten Daten in eine JSON-Anfrage (Abbildung 23) um und sendet diese an den Service. Sobald die JSON-Antwort (Abbildung 24) empfangen wurde, wird diese eingelesen und fließt in den Zuverlässigkeitsassistenten ein. Die einzige Anpassung an der Benutzeroberfläche ist ein Ladebalken, welches bis zum Eintreffen der Antwort angezeigt wird (Abbildung 14).

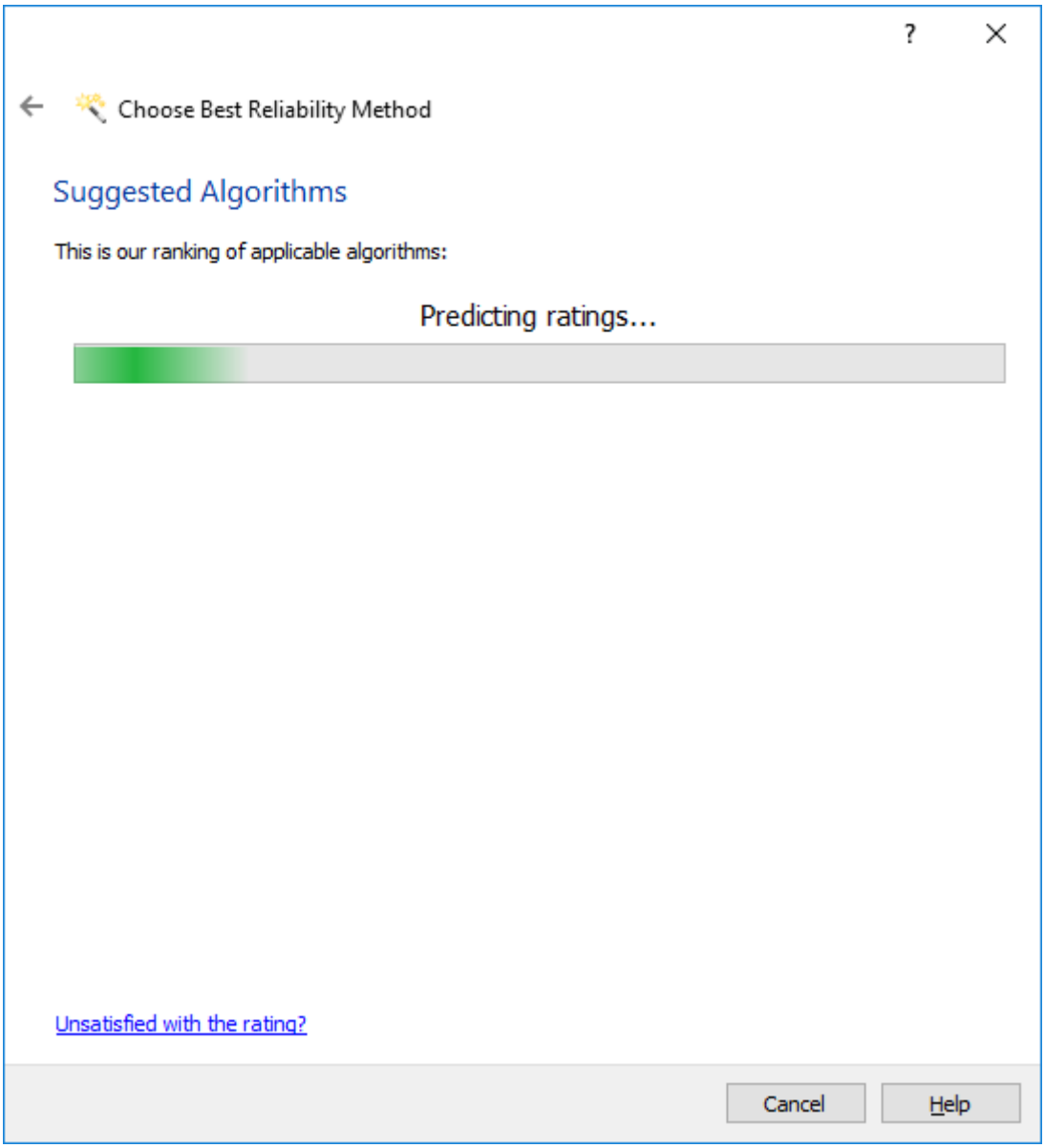

Abbildung 14 - Zuverlässigkeitsassistent Ladebalken

4.2 Web-Service

Für die Implementierung des Web-Services wurde Python gewählt. Python bietet zahlreiche Tools zum Manipulieren von Daten an und ist bei Anwendern von maschinellem Lernen sehr beliebt. Hier ist zu beachten, dass sowohl das Einlesen und die Umwandlung der Daten, das Trainieren des Entscheidungsbaumes, die Analyse des resultierenden Baumes und die Prognose des Entscheidungsbaumes mit Python umgesetzt wurden. Des Weiteren wird die daraus resultierende Softwarelösung in Amazon Web Services eingebunden und muss dementsprechend angepasst werden. Im Folgenden wird näher in die Implementierung dieser Komponenten eingegangen.

4.2.1 Maschinelles Lernen

Die gesammelten Daten müssen vor dem maschinellen Lernen vorbereitet werden, damit der Algorithmus diese verwenden kann. Hierfür wurde pandas verwendet, eine Programmbibliothek die als Werkzeug zur Verwaltung von Daten und deren Analyse dient. Dies wird hauptsächlich für das Erstellen und Bearbeiten von Tabellen genutzt. Des Weiteren müssen für die Stichproben, welche eine variable Länge besitzen Merkmale entzogen werden, um diese Stellvertretend dem maschinellen Lernen zu übergeben. Im Folgenden werden entscheidende Code-Abschnitte erläutert und gegebenenfalls veranschaulicht. Außerdem wird erläutert, wie das maschinelle Lernen in Python angewandt wurde.

4.2.1.1 Einlesen der Daten

Die gesammelten Daten wurden als CSV-Dateien abgespeichert. Hierfür wurde eine Funktion *convertData* angelegt, welche mithilfe von pandas alle CSV-Dateien durchläuft, einliest, ordnungsgemäß interpretiert und die Daten einheitlich als Tabelle abspeichert. Hierbei beachtet die Funktion die jeweilige Formatierung, die Jurojin beim Exportieren verwendet und beachtet ebenfalls die Spracheinstellungen der unterschiedlichen Zahlenformate. Außerdem werden einige Einträge angepasst indem beispielsweise leere Werte mit 0 oder wie im Falle der Familie im Versuchsplan mit „None“ ersetzt werden. Am Ende wird die Tabelle „bereinigt“ indem Spalten, die nicht mehr benötigt werden, weil diese beispielsweise die Spracheinstellungen enthalten, entfernt. Außerdem ist zu beachten, dass es unterschiedliche Stichprobenklassifikationen gibt, die sich jedoch nur durch Wegfallen von Informationen und unterschiedlicher Bezeichnungen der Informationen unterscheiden. Somit kann hierfür ein allgemeines Format verwendet werden. Die Funktion *createStandartSample* (Abbildung 15) wandelt eine Beliebige Stichprobe in eine standardisierte Stichprobe um. Diese kann somit verwendet werden, um Merkmale aus dieser zu entnehmen.

```
1.  def createStandardSample(sample_raw, sampleClassification):
2.      lifeTimeValues = pd.DataFrame()
3.      groupValues = pd.DataFrame()
4.      censorValues = pd.DataFrame()
5.
6.      if 'SampleType' == sampleClassification:
7.          lifeTimeValues = sample_raw.iloc[:,0]
8.          groupValues = sample_raw.iloc[:,1]
9.          censorValues = np.nan
10.     elif 'SurvivalType' == sampleClassification:
11.         lifeTimeValues = sample_raw.iloc[:,0]
12.         groupValues = sample_raw.iloc[:,2]
13.         censorValues = sample_raw.iloc[:,1]
14.     elif 'WoehlerType' == sampleClassification:
15.         lifeTimeValues = sample_raw.iloc[:,1]
16.         groupValues = np.nan
17.         censorValues = sample_raw.iloc[:,2]
18.     elif 'EnduranceLimitType' == sampleClassification:
19.         lifeTimeValues = np.nan
20.         groupValues = np.nan
21.         censorValues = sample_raw.iloc[:,1]
22.
23.     standardSample = pd.DataFrame(columns=['lifeTimeValues', 'censorValues','groupVa
    lues'])
24.     standardSample['lifeTimeValues'] = lifeTimeValues
25.     standardSample['censorValues'] = censorValues
26.     standardSample['groupValues'] = groupValues
27.     return standardSample
```

Abbildung 15 - Funktion createStandardSample

4.2.1.2 Bestimmung der Merkmale (Stichprobe)

lifeTimeValues	censorValues	groupValues
79728	1	Januar
92385	0	Januar
90860	0	Januar
60102	1	Februar
95511	0	Februar

Tabelle 6 - Standardisierte Stichprobe

Die oben dargestellte Tabelle zeigt ein Beispiel für eine Standarisierte Stichprobe, da in Jurojin die Stichproben unterschiedlich klassifiziert werden. Aus dieser müssen deskriptive Merkmale entnommen werden, um mit Ihrer variablen Länge umzugehen. Die Klassifikation kann hierbei übernommen werden. Die Funktion *convertData* verwendet ebenfalls Funktionen, die der Stichprobe einige Merkmale entnimmt. Die Funktion

calcSuspendedRatio (Abbildung 16) berechnet das Verhältnis der Zensurdaten. Da die Zensurdaten nur aus Ausfällen (1) oder Durchläufern (0) bestehen müssen nur die Durchläufer gezählt und durch die Länge der Stichprobe berechnet werden.

```
1.  def calcSuspendedRatio(censorValues):
2.      valueCounts = censorValues.value_counts()
3.      if(censorValues.nunique() < 2):
4.          if 1 in censorValues.unique():
5.              return 0
6.          else:
7.              return 1
8.      valueCounts = censorValues.value_counts()
9.      suspendedRatio = float(valueCounts[0]) / float((valueCounts[0] + valueCounts[1])
    )
10.     return suspendedRatio
```

Abbildung 16 - Funktion calcSuspendedRatio

Die Funktion *calcGroupAllocation* berechnet einen Wert, der Auskunft darüber gibt wie gleichmäßig die Gruppen verteilt wurden. Hierbei steht der Wert 1 für eine gleichmäßige Verteilung und Werte nahe 0 für eine starke Ungleichheit. Sollten keine Gruppendaten vorhanden sein wird der Wert auf 1 festgelegt.

```
1.  def calcGroupAllocation(groupValues, numberOfUniqueGroups, sampleSize):
2.      groupAllocation = 1
3.      for groupCount in groupValues.value_counts():
4.          groupAllocation *= numberOfUniqueGroups * groupCount / sampleSize
5.      return groupAllocation
```

Abbildung 17 - Funktion calcGroupAllocation

Ein weiteres Merkmal wird durch die Funktion *checkOnlySuspendedGroups* (Abbildung 18) bestimmt. Hier werden alle Zensurdaten durch die Gruppendaten aufgeteilt und überprüft ob sich Gruppen finden lässt, die nur aus Durchläufern bestehen. Sollten keine Gruppendaten vorhanden sein wird *False* festgelegt.

```
1.  def checkOnlySuspendedGroups(standardSample):
2.      groupValues = standardSample['groupValues']
3.      censorValues = standardSample['censorValues']
4.      groupCounts = groupValues.value_counts()
5.      hasOnlySuspendedGroups = False
6.      for groupName in groupValues.unique():
7.          if isinstance(groupName, float):
8.              if math.isnan(groupName):
9.                  if censorValues.isna().sum() > 0:
10.                     break
11.                 if groupValues.all() and 0 in groupValues:
12.                     hasOnlySuspendedGroups = True
13.                 break
14.         groupNameOccurrences = groupValues.value_counts().loc[groupName]
15.         groupNameAndSuspendedOccurrences = standardSample.loc[(censorValues == 0) &
    (groupValues == groupName), 'censorValues'].value_counts()
16.         if len(groupNameAndSuspendedOccurrences) == 0:
17.             continue
18.         groupNameAndSuspendedOccurrences = groupNameAndSuspendedOccurrences.iloc[0]

19.         if (groupNameOccurrences == groupNameAndSuspendedOccurrences):
20.             hasOnlySuspendedGroups = True
21.             break
22.     return hasOnlySuspendedGroups
```

Abbildung 18 - Funktion checkOnlySuspendedGroups

Schließlich wird noch die Stichprobengröße als Merkmal verwendet.

4.2.2 Training

Für maschinelles Lernen wurde die Bibliothek *scikit-learn* verwendet. Diese bietet zahlreiche Werkzeuge für das maschinelle Lernen und ist kompatibel mit den Tabellen-Datenstrukturen von pandas. Um Entscheidungsbäumen, die zur Klassifikation geeignet sind, zu trainieren bietet *scikit-learn* den *DecisionTreeClassifier* [12] an. Da der Zuverlässigkeitsassistent dem Nutzer für fünf verschiedene Methoden jeweils eine Bewertung gibt, ist es notwendig fünf verschiedene Modelle zu trainieren. Da, wie bereits im Unterabschnitt Skalierung im Kapitel Maschinelles Lernen erläutert wurde, eine Skalierung der Trainingsdaten für den Entscheidungsbaum nicht notwendig ist, benötigt man nur noch eine Umwandlung der kategorialen Daten. Bei diesen Daten handelt es sich um die Klassifikation der Stichprobe, der Verteilungsfamilie und der Gruppeninterpretation. Hier bietet *scikit-learn* den *OneHotEncoder* an. Dieser verwendet das im Unterabschnitt 3.3.3.2 (Kreuzvalidierung) beschriebe Verfahren. Der unten abgebildete Code demonstriert, wie dieser erstellt wird.

```
1.  def createOneHotEncoder():
2.      groupPolicy = ['IgnoreGroupsPolicy', 'TreatAsSubPopulationsPolicy', 'TreatAsComp
    etingRiskPolicy']
3.      family = ['W', 'N', 'LN', 'LN10', 'None']
4.      sampleClassification = ['SampleType', 'SurvivalType', 'WoehlerType', 'EnduranceL
    imitType']
5.      return OneHotEncoder(categories=[groupPolicy, family, sampleClassification])
```

Abbildung 19 – Funktion createOneHotEncoder

Nachdem der *OneHotEncoder* angewandt wurde, können die Modelle mit dem *DecisionTreeClassifier* trainiert werden.

4.2.3 Amazon Web Services

Die Implementierung als Web-Service wurde mit Amazon Web Services gelöst. Es bietet eine Vielzahl an zuverlässigen, skalierbaren und Cloud Computing-Services an [13]. Im Rahmen dieser Bachelor-Arbeit wurden die Services Amazon API Gateway, Amazon Lambda und Amazon SageMaker verwendet. Hierfür stellt Amazon API Gateway die HTTPS-Schnittstelle bereit, bei denen ein Benutzer eine Anfrage senden kann und Amazon Lambda leitet Anfragen an Amazon SageMaker weiter, welcher das Training und die Vorhersagen übernimmt. Wurde eine Vorhersage getroffen sendet Amazon SageMaker die Vorhersage an Amazon Lambda weiter und dieser übergibt es Amazon API Gateway, welcher die Antwort an den Benutzer weiterleitet. Abbildung 20 veranschaulicht das eben Beschriebene Vorgehen.

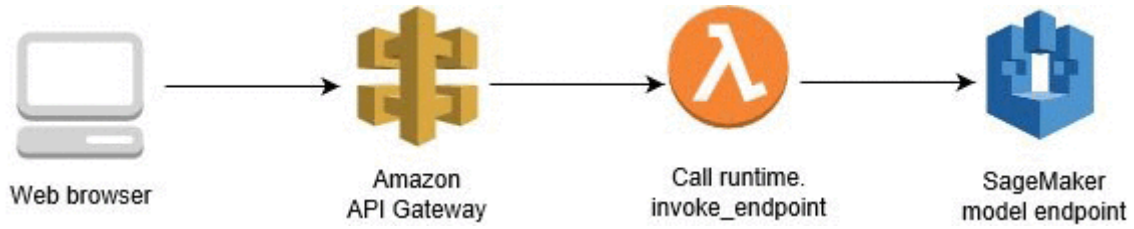

Abbildung 20 - Schema Vorgehen Webservice

4.2.3.1 Amazon API Gateway

Amazon API Gateway bietet dem Entwickler eine Vielzahl an Schnittstellen an, die beliebig konfigurierbar sind. In Falle dieser Bachelor Arbeit wurde eine HTTPS-Schnittstelle angelegt. Diese wurde so konfiguriert, dass eine Anfrage an Amazon Lambda weitergeleitet wird.

4.2.3.2 *Amazon Lambda*

Amazon Lambda ist eine serverlose Lösung, mit der sich verschiedene Skripte und Programme ausführen lassen. Es wurde verwendet, um die Kommunikation zwischen Amazon API Gateway und Amazon SageMaker mithilfe von Python zu ermöglichen. Sobald Amazon Lambda vom Amazon API Gateway eine Anfrage erhält, wird ein überschaubares Python-Skript ausgeführt, welches die Anfrage an Amazon API Gateway weiterleitet und dessen Antwort verarbeitet. Es ist zu beachten, dass das Python-Skript nur JSON-Anfragen weiterleitet und die Antwort vom Amazon SageMaker entsprechend als JSON-Objekt erstellt und an Amazon API Gateway weiterleitet.

```
1.  import os
2.  import io
3.  import boto3
4.  import json
5.  import csv
6.
7.  # grab environment variables
8.  ENDPOINT_NAME = os.environ['ENDPOINT_NAME']
9.  runtime= boto3.client('runtime.sagemaker')
10.
11. def lambda_handler(event, context):
12.     print("Received event: " + json.dumps(event, indent=2))
13.
14.     data = json.loads(json.dumps(event))
15.     payload = data['data']
16.
17.     response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
18.                                        ContentType='application/json',
19.                                        Body=json.dumps(payload))
20.
21.     result = response['Body'].read().decode()
22.     results = []
23.     for line in result.splitlines():
24.         results.append(line)
25.
26.     return {
27.         'ComponentCount' : results[0],
28.         'GroupwiseEvaluation' : results[1],
29.         'ThreeParamWeibull' : results[2],
30.         'PolyWeibull' : results[3],
31.         'QuantileEstimation' : results[4]
32.     }
```

Abbildung 21 - Amazon Lambda Python-Skript

4.2.3.3 *Amazon SageMaker*

Amazon SageMaker ist ein Dienst, welches alle benötigten Werkzeuge bereitstellt und somit ermöglicht, Modelle mithilfe von maschinellem Lernen zu trainieren und zu verwenden. Die Implementierung beinhaltet neben dem Trainieren der Modelle auch eine Validierung dieser Modelle. Hierbei wird nach dem Training eine Auflistung aller Fehlgeschlagenen Daten der

Kreuzvalidierung ausgegeben. Wie in Abbildung 22 zu sehen ist, wird ebenfalls eine Konfusionsmatrix zu jedem Modell erstellt und dargestellt. Somit lassen sich die neuen Modelle mit den Alten vergleichen und die falschen Schätzungen nachvollziehen. Die Implementierung erwartet eine JSON-Anfrage und antwortet ebenfalls mittels JSON. In Abbildung 21 ist zu sehen, dass Amazon SageMaker mittels *invoke_endpoint* angesprochen wird. Abbildung 23 zeigt eine mögliche Anfrage und Abbildung 24 eine mögliche Antwort.

```
ComponentCount:

[Half Star missed] Wrong prediction at row: 27 Description:
Zuverlaessigkeitsnachweis bei kleinen Stichproben (N=3) - 3
Expected: ['Empty Star'] Predicted: ['Half Star']
[Half Star missed] Wrong prediction at row: 37 Description:
Zuverlaessigkeitsnachweis bei kleinen Stichproben (N=5) - 2
Expected: ['Half Star'] Predicted: ['Full Star']
[Half Star missed] Wrong prediction at row: 42 Description:
QuantileEstimation_twoEarlySuspended
Expected: ['Half Star'] Predicted: ['Full Star']
[Half Star missed] Wrong prediction at row: 48 Description:
Zuverlaessigkeitsnachweis bei kleinen Stichproben (N=3) - 2
Expected: ['Half Star'] Predicted: ['Full Star']
[Half Star missed] Wrong prediction at row: 58 Description:
Zuverlaessigkeitsnachweis bei kleinen Stichproben (N=5) - 1
Expected: ['Full Star'] Predicted: ['Half Star']

Precision: 0.91935483871

Confusion-matrix:
48   1    0
0    2    3
0    1    7
```

Abbildung 22 - Beispiel Ausgabe SageMaker

```json
{
  "data": {
    "sample": {
      "lifeTimeValues": [
        12,
        2
      ],
      "censorValues": [
        1,
        0
      ],
      "groupValues": [
        "A",
        "B"
      ]
    },
    "doe": {
      "family": "W",
      "shape": 2,
      "doESampleSize": 27,
      "reliability": 99,
      "significance": 1,
      "designLife": 1,
      "lifeSpanRatio": 5
    },
    "settings": {
      "groupPolicy": "IgnoreGroupsPolicy",
      "sampleClassification": "SampleType"
    }
  }
}
```

Abbildung 23 - JSON-Anfrage

```json
{
  "ComponentCount": "Half Star",
  "GroupwiseEvaluation": "Empty Star",
  "ThreeParamWeibull": "Full Star",
  "PolyWeibull": "Empty Star",
  "QuantileEstimation": "Full Star"
}
```

Abbildung 24 - JSON-Antwort

4.2.4 Experimentelle Ergebnisse

Im Folgenden werden die experimentellen Ergebnisse der trainierten Modelle vorgestellt. Aufgrund der geringen Anzahl der Trainingsdaten sind diese jedoch nur bedingt aussagekräftig. Die unten dargestellten Abbildungen veranschaulichen die jeweiligen Konfusionsmatrizen mit der Genauigkeit einer Schätzung in Prozent. Da die gelb Markierten Felder im Interpretationsfreiraum liegen, wird die Genauigkeit einer Schätzung in einem Bereich von x bis y berechnet. Hierbei steht x für die richtige Schätzung bei den grünen Feldern und y bei den grünen und gelben Feldern. Das Testen fand mithilfe der Kreuzvalidierung statt.

Neuer Zuverlässigkeitsnachweis 90-100%		Tatsächliche Klasse		
		Leerer Stern	Halber Stern	Voller Stern
Ermittelte Klasse	Leerer Stern	**48**	1	0
	Halber Stern	0	**2**	3
	Voller Stern	0	2	**6**

Abbildung 25 - Konfusionsmatrix Neuer Zuverlässigkeitsnachweis

Gruppenweise Auswertung 89-98%		Tatsächliche Klasse		
		Leerer Stern	Halber Stern	Voller Stern
Ermittelte Klasse	Leerer Stern	**50**	0	1
	Halber Stern	0	**1**	3
	Voller Stern	0	3	**4**

Abbildung 26 - Konfusionsmatrix Gruppenweise Auswertung

3-Parameter-Weibull 87-95%		Tatsächliche Klasse		
		Leerer Stern	Halber Stern	Voller Stern
Ermittelte Klasse	Leerer Stern	**44**	2	2
	Halber Stern	2	**2**	0
	Voller Stern	1	1	**8**

Abbildung 27 - Konfusionsmatrix 3-Parameter-Weibull

Poly-Weibull 85-94%		Tatsächliche Klasse		
		Leerer Stern	Halber Stern	Voller Stern
Ermittelte Klasse	Leerer Stern	**49**	1	1
	Halber Stern	2	**1**	1
	Voller Stern	3	1	**3**

Abbildung 28 - Konfusionsmatrix Poly-Weibull

Neuer Quantilschätzer 90-100%		Tatsächliche Klasse		
		Leerer Stern	Halber Stern	Voller Stern
Ermittelte Klasse	Leerer Stern	**40**	3	0
	Halber Stern	1	**3**	0
	Voller Stern	0	2	**13**

Abbildung 29 - Konfusionsmatrix Neuer Quantilschätzer

Die unterschiedlichen Methoden wurden nicht gleich gut bewertet. Die Eignung für einen neuen Zuverlässigkeitsassistenten und einen neuen Quantilschätzer haben die besten Ergebnisse. Hier gab es im Gegensatz zu den anderen keine Fehlprognosen, die über einen ganzen Stern verfehlten. 90% der Vorhersagen lagen hier bei den grünen Feldern und 100% bei den grünen und gelben Feldern. Des Weiteren ist zu beachten, dass der größte Teil der Trainingsdaten Bewertungen mit leeren Sternen beinhaltet. Jedoch ist das Ergebnis für die geringe Anzahl an Trainingsdaten zufriedenstellend. Da in Zukunft die Datenbank mit Kundendaten erweitert wird, sind bessere Ergebnis zu erwarten.

5 ZUSAMMENFASSUNG UND AUSBLICK

Mithilfe von maschinellem Lernen konnten die Auswertungen von Überlebensdaten für den Zuverlässigkeitsassistenten in Jurojin verbessert werden. Hierfür wurde Jurojin mit dem Export der relevanten Daten erweitert und eine Datenstruktur gewählt, die für einen Entwickler leicht nachvollziehbar und bearbeitbar ist. Der Entscheidungsbaum wurde als ein geeignetes Verfahren für maschinellen Lernen gewählt und für das Problem mit der variablen Länge der Stichprobe wurden deskriptiven Merkmale verwendet. Da keine Trainingsdaten vorliegen wurde die erste Menge an Trainingsdaten aus Unit-Test-Fällen entnommen, generiert und von Kunden zugesendete Daten verwendet. Für die trainierten Modelle musste ein Validierungsverfahren gewählt werden, welches selbst für eine gering Anzahl an Trainingsdaten ein Aussagekräftiges Ergebnis liefern kann. Am Ende wurde das maschinelle Lernen als Webservice umgesetzt und Jurojin entsprechend angepasst.

Trotz der geringen Anzahl an Trainingsdaten sind die trainierten Modelle zufriedenstellend. Mit zunehmenden Daten werden die Modelle immer genauere Schätzungen durchführen können. Zudem kann der Entscheidungsbaum jederzeit ohne große Mühen mit dem Random Forrest ausgetauscht werden, was bei mehr Trainingsdaten Sinn ergeben kann.

A LITERATURVERZEICHNIS

[1] „Fraunhofer ITWM,“ [Online]. Available: https://www.itwm.fraunhofer.de/. [Zugriff am April 2020].

[2] Fraunhofer, „JUROJIN – Statistische Auswertung von Betriebsfestigkeitsversuchen,“ Fraunhofer ITWM, [Online]. Available: https://www.itwm.fraunhofer.de/de/abteilungen/mf/produkte-und-leistungen/jurojin.html. [Zugriff am April 2020].

[3] Fraunhofer, „Jurojin - Dokumentation - Zuverlässigkeitsnachweis“.

[4] Fraunhofer, „Jurojin - Dokumentation - Quantil- und Verteilungsschätzung“.

[5] Fraunhofer, „Jurojin - Dokumentation - Beste Zuverlässigkeitsauswertung wählen“.

[6] Fraunhofer, „Jurojin - Dokumentation - 3-Parameter-Weibullverteilung“.

[7] E. Alpaydin, Maschinelles Lernen, Oldenbourg, 2008.

[8] „The Qt Company,“ [Online]. Available: https://www.qt.io/. [Zugriff am 07 2020].

[9] A. AWS, „Amazon,“ [Online]. Available: https://aws.amazon.com/de/sdk-for-cpp/. [Zugriff am 07 2020].

[10] „haxx.se,“ [Online]. Available: https://curl.haxx.se/libcurl/c/libcurl.html. [Zugriff am 07 2020].

[11] N. Lohmann, „JSON for modern C++,“ [Online]. Available: https://github.com/nlohmann/json. [Zugriff am 07 202].

[12] „scikit-learn - Decision Trees,“ 2020. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#classification.

[13] „Amazon Web Services,“ 2020. [Online]. Available: https://aws.amazon.com/de/.

[14] „machinelearningmastery,“ [Online]. Available: https://machinelearningmastery.com/how-to-save-and-load-models-and-data-preparation-in-scikit-learn-for-later-use/. [Zugriff am April 2020].

[15] Fraunhofer, „Jurojin - Dokumentation - Poly-Weibull-Verteilung“.

[16] „Amazon API Gateway,“ [Online]. Available: https://aws.amazon.com/de/api-gateway/. [Zugriff am 2020].

[17] „Call an Amazon SageMaker model endpoint using Amazon API Gateway and AWS Lambda,“ 2020. [Online]. Available: https://aws.amazon.com/de/blogs/machine-learning/call-an-amazon-sagemaker-model-endpoint-using-amazon-api-gateway-and-aws-lambda/.

[18] „scikit-learn homepage,“ 2020. [Online]. Available: https://scikit-learn.org/stable/.

[19] „Amazon Lambda,“ [Online]. Available: https://aws.amazon.com/de/lambda/. [Zugriff am 2020].

B ABBILDUNGSVERZEICHNIS