INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Lerngruppenbildung anhand von Kompetenzschätzung

mittels Bayesscher Netze

Robert Pospisil

Masterarbeit

Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, Niels Heller
Abgabe am	16.01.2019

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 16.01.2019 Robert Pospisil

Abstract

Group work is used in teaching in almost the entire education system. While allocations for group work in the primary and secondary education sector can still be carried out by the teachers themselves, this is in tertiary education such as at universities, due to significantly larger course attendances and partly less personal contact, often meaningful only with software-based solutions.

In this work, a novel generic software was developed that can create different learning groups based on different factors and criteria. This includes heterogeneous groups, homogeneous groups and peer review assignments. In general, some of these criteria are not directly measurable, or directly usable data are not available, therefore must be estimated. This work focuses on students' competence as a criteria for group formation.

For the group formation, only data within the project management platform of the chair for programming and modeling languages of the Ludwig-Maximilian-University of Munich is available, in which the software has been integrated. Due to this fact, a competence estimator was developed based on the submissions on exercise tasks. On the one hand, this uses the „knowledge space theory" to structure the students' knowledge and, on the other, a bayesian network for the actual assessment.

At the beginning of the work needed basic concepts are explained. Subsequently, the data used are presented and, on this basis, the competence estimator is constructed such as the functioning of the group education module is explained. Afterwards, the implementation of the individual components is discussed. Finally, the competence estimator is evaluated and a conclusion is drawn.

Zusammenfassung

Gruppenarbeit wird in der Lehre im nahezu gesamten Bildungssystem eingesetzt. Während die Zuteilungen für Gruppenarbeiten im primären und sekundären Bildungsbereich noch von den Lehrenden selbst durchgeführt werden können, ist dies im tertiären Bereich wie Universitäten, aufgrund von deutlich größerer Anzahl an Teilnehmern in Kursen und teilweise geringerem persönlichen Kontakt, häufig nur noch mit softwarebasierten Lösungen sinnvoll möglich.

In dieser Arbeit wurde eine generische Software entwickelt, die unterschiedlichste Lerngruppen anhand diverser Faktoren und Kriterien erstellen kann. Dies beinhaltet sowohl heterogene Gruppen als auch homogene Gruppen, sowie Zuteilungen für Peer Reviews. In der Regel sind einige dieser Kriterien nicht direkt messbar, beziehungsweise direkt verwertbare Daten sind nicht verfügbar, was zur Folge hat, dass diese abgeschätzt werden müssen. Diese Arbeit konzentriert sich auf die Kompetenz der Studenten als Kriterium für die Gruppenbildung.

Für die Gruppeneinteilung stehen nur Daten aus der Projektverwaltungsplattform des Lehrstuhls für Programmier- und Modellierungssprachen der Ludwig-Maximilians-Universität München zur Verfügung, in welche die Software integriert wurde. Aufgrund dessen wurde ein Kompetenzschätzer auf Basis von Abgaben zu Übungsaufgaben entwickelt. Dieser verwendet zum einen die „Knowledge Space Theorie“, um das Wissen der Studenten zu strukturieren und zum anderen ein Bayessches Netz zur eigentlichen Einschätzung.

Zu Beginn der Arbeit werden benötigte Grundlagen erläutert, mit anschließender Vorstellung der verwendeten Daten. Darauf aufbauend wird der Kompetenzschätzer konstruiert sowie die Funktionsweise des Gruppenbildungsmoduls erläutert. Darauffolgend wird auf die Implementierung der einzelnen Komponenten eingegangen. Abschließend wird der Kompetenzschätzer evaluiert und ein Fazit gezogen.

Danksagung

Zuallererst danke ich Herrn Prof. Dr. François Bry für die Gelegenheit meine Masterarbeit an der Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen zu schreiben. Zudem möchte ich mich bei ihm für sein kontinuierliches Feedback während dem Entstehen dieser Arbeit danken. Meinem Betreuer Niels Heller danke ich, dass er mir dieses Thema vorgeschlagen und mir die dafür notwendigen Daten zur Verfügung gestellt hat. Außerdem bin ich ihm für seine durchgehende Unterstützung bei der Entwicklung und sein Feedback beim Schreiben dieser Arbeit dankbar. Während unserer Gespräche entstanden viele nützliche Ideen, die in diese Arbeit eingeflossen sind. Abschließend möchte ich meiner Familie für die Unterstützung während meines Studiums danken sowie insbesondere meiner Mutter für das Korrekturlesen dieser Arbeit.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Die Problematik effiziente Gruppen zu bilden betrifft die meisten Bereiche des Lebens in denen Menschen sich zu Gruppen zusammenfinden, sei es im Beruf für das erfolgreiche Abschließen von Projekten, in der Freizeit für einen Verein oder im Bildungssegment für eine effektive Lehre. Verschiedene Gruppenkonstellationen können diverse Vor- und Nachteile gegenüber anderen Konstellationen haben und somit das Erreichen des übergeordneten Zieles entweder erleichtern oder erschweren. Als einfaches Beispiel können Aufstellungen in einem Mannschaftssport wie Fußball betrachtet werden. Basierend auf dem Gegner können unterschiedlichste Aufstellungen sinnvoll sein. Zudem gibt es Aufstellungen, die vermutlich selten bis nie zum Erfolg führen; exemplarisch ein Spiel mit 11 Stürmern zu bestreiten.

Insbesondere für die computergestützte Bildung von Gruppen sind spezielle Kriterien nötig, mit denen die Mitglieder untereinander verglichen und eingeordnet werden können. Häufig ist es wichtig zu ermitteln, welche Fähigkeiten jemand besitzt. Um während eines Projektes beispielsweise eine bestimmte Aufgabe zu erfüllen sind breitgefächerte Kenntnisse nötig. Insbesondere bei größeren Projekten können viele verschiedene Kenntnisse oder Fähigkeiten für einen erfolgreichen Abschluss Voraussetzung sein, die nicht alle Mitglieder besitzen. Für die Entwicklung einer Anwendung werden gegebenenfalls Programmierer, Designer, Projektleiter usw. benötigt.

Neben der Frage, ob und welche Fähigkeiten eine Person besitzt, ist es auch wichtig zu ermitteln, wie fähig eine Person in einem bestimmten Bereich ist; oder wie fähig sie vermutlich ist, sofern eine exakte Einstufung nicht möglich ist. Somit wird für die einzelnen Personen eine Leistungseinschätzung benötigt. Mit dieser Einschätzung können dann etwa Gruppen, deren Teilnehmer die benötigten Fähigkeiten besitzen, gebildet werden und somit für diese passende Ziele definiert werden. Hiermit kann eine Unter- oder Überforderung von Gruppen vermieden werden.

Neben einer Einteilung, basierend auf den Fähigkeiten der einzelnen Mitglieder einer Gruppe, sind auch weitere Faktoren für erfolgreiche Gruppenprojekte von Bedeutung. Hierzu zählt unter anderem auch die Motivation und Arbeitsbereitschaft der Personen; aber auch soziale Aspekte wie die vorherrschende Stimmung während eines Projektes. Diese können sowohl positive wie auch negative Auswirkungen auf das Resultat haben und beeinflussen sich bilateral. Jedoch sind diese Faktoren häufig schwer abschätzbar, insbesondere sofern es sich um „unbekannte“ Personen handelt, mit denen zuvor kein persönlicher Kontakt bestand.

Je nachdem welches Ziel eine Gruppe oder mehrere Gruppen erreichen soll, können

verschiedene Gruppierungsstrategien benötigt werden. Für manche kann es von Vorteil sein viele ähnliche Individuen zu gruppieren; eine homogene Gruppe zu bilden. Während es für andere Aufgaben relevant sein kann in jeder Gruppe eine gleiche Verteilung der benötigten Kriterien sicher zu stellen; eine heterogene Gruppe zu bilden. Teilweise kann auch eine Mischung aus beidem nützlich sein.

Für einen reibungslosen Lehr- und Lernbetrieb ist es mitunter nötig die Studenten in verschiedene Gruppen einzuteilen. Dies kann einerseits für Gruppenarbeiten wie Programmierpraktika nötig sein, andererseits auch für den Übungsbetrieb. Es kann zweckmäßig sein, Studenten mit ähnlichem Wissen homogen zu gruppieren, um sie gezielt zu unterrichten. Dahingegen kann es sich als nützlich erweisen, Studenten mit unterschiedlichen Fähigkeiten in heterogene Gruppen einzuteilen, mit dem Ziel, dass diese von einander lernen können und sich gegenseitig ergänzen.

In manchen Bildungseinrichtungen, wie in Schulen, können die Lehrenden die Lernenden mitunter recht gut einschätzen, da sie zum einen viel Zeit miteinander verbringen und zum anderen die Gruppengröße dies noch zulässt. Im Gegensatz dazu werden Vorlesungen an Universitäten teilweise vor mehreren hundert Studenten gehalten, wodurch der persönliche Kontakt der Beteiligten offensichtlich geringer ausfällt. Um dennoch „unbekannte“ Studenten anhand von gewissen Kriterien einzuteilen, ist es erforderlich mithilfe von vorliegenden Daten die Studenten einzuschätzen. Zudem ist es aufgrund der Anzahl der Studenten erstrebenswert, dass dies automatisch erfolgt.

Diese Arbeit beschreibt die automatische Lerngruppenbildung unter Berücksichtigung diverser Faktoren. Auf Grundlage der zur Verfügung stehenden Daten wird ein Kompetenzschätzer aufgebaut. Die Daten stammen von der lehrstuhleigenen Projektverwaltungswebsite, in die die Lerngruppenbildung integriert werden soll. Dadurch soll die Möglichkeit für unterschiedliche Arten der Gruppenbildungen geschaffen werden. Dies beinhaltet die Bildung von homogenen als auch heterogenen Gruppen sowie die Zuteilung für Peer Review Verfahren.

Das zweite Kapitel gibt einen Überblick über benötigte Grundlagen. Dazu zählen die für den Kompetenzschätzer benutzten Konzepte „Bayessches Netz“ sowie „Knowledge Space Theory“. Zudem werden einige Ansätze anderer Studien in diesem Gebiet betrachtet und deren Vorgehensweise und Ergebnisse dargelegt. Anschließend wird zu Beginn des dritten Kapitels der von der Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen zur Verfügung gestellte Datensatz erläutert. Dieser beinhaltet Daten zu von Studenten bearbeiteten Programmieraufgaben.

Darauf aufbauend erfolgt im dritten Kapitel die Erklärung für die Konstruktion des Kompetenzschätzers sowie der Gruppenbildung. Im vierten Kapitel wird auf die technische Implementierung eingegangen. Unter anderem erfolgt die Betrachtung für das vom Kompetenzschätzer genutzte Framework Weka, welches eine Implementierung für Bayessche Netze bietet.

Darauffolgend wird im fünften Kapitel eine Evaluierung des Kompetenzschätzers durchgeführt. Hierbei werden zunächst wichtige Konzepte und Metriken für die Evaluierung erläutert, sowie diverse Einsatzmöglichkeiten des Kompetenzschätzers betrachtet und miteinander verglichen. Abschließend folgt im letzten Kapitel eine Zusammenfassung der Ergebnisse und ein Fazit.

KAPITEL 2

Einordnung in den Forschungskontext

Nachfolgend werden einige im weiteren Verlauf der Arbeit genutzte Konzepte kurz eingeführt. Begonnen wird dabei mit der Einschätzung von Individuen, die für die Einteilung in Gruppen nötig ist. Hierbei wird auf verschiedene Ansätze sowohl für die Gruppenbildung, als auch für die Einschätzung eingegangen. Anschließend werden die mathematischen Grundlagen zum verwendeten Schätzer, Bayessches Netz, erläutert. Die Struktur des in Kapitel 3 definierten Bayesschen Netzes, das der für diese Arbeit verwendeten Anwendung zugrunde liegt, wird durch die Knowledge Space Theory motiviert, die im letzten Abschnitt dieses Kapitels betrachtet wird.

2.1 Kompetenzschätzung

Abhängig von den Zielen der Gruppenarbeit bieten sich unterschiedliche Kriterien zur Gruppenbildung an. So kann es unter anderem für Übungsgruppen gewünscht sein, dass alle Teilnehmer einer Gruppe einen ähnlichen Wissensstand aufweisen. Während es für Gruppenarbeiten unter Umständen besser sein kann möglichst inhomogene Gruppen zu bilden, so dass in jeder Gruppe eine gewisse Erfahrung vorhanden ist, damit die erfahreneren Gruppenmitglieder den unerfahreneren helfen können. Dies kann den Betreuungsaufwand reduzieren und für fairere Grundvoraussetzungen sorgen. Ausgehend von den untersuchten Studien wurden die in diesen vorgefundenen Merkmale für Lerngruppenbildung in zwei Übergruppen aufgeteilt. Merkmale der ersten Gruppe sind gegebenenfalls variabler als die der zweiten Gruppe.

Persönliche Leistung Zu dieser Gruppe zählen zum Beispiel Qualifikationen in gewissen Gebieten wie Abschlüsse oder Klausurergebnisse. Des Weiteren sind auch Kriterien wie die Motivation beim Bearbeiten von Aufgaben, die Bearbeitungsgeschwindigkeit und ähnliches von Interesse. Allgemein stellt sich die Frage, wie fähig ein Student in unterschiedlichen Disziplinen ist. Einige dieser Daten werden eventuell bei der Einschreibung in ein Studium oder in einen Kurs angegeben, andere müssen eher beobachtet und geschätzt werden.

Persönliche Eigenschaften In diese Gruppe zählen zum einen körperliche Attribute wie Alter, Geschlecht, Herkunft/Ethnie oder familiärer Hintergrund; zum anderen auch das vorhandene Zeitbudget, das zum Beispiel durch einen ausgeübten Beruf oder die

Anzahl an Vorlesungen beeinflusst wird. Während manche Eigenschaften bekannt sein können, wie zum Beispiel Alter oder das Geschlecht, müssen andere gezielt durch Befragungen ermittelt werden.

Nachfolgend werden einige der Studien kurz vorgestellt. Zunächst liegt der Fokus auf der Verwendung der unterschiedlichen Merkmale. Im nächsten Abschnitt werden auch Studienergebnisse bezüglich der Gruppierung gezeigt.

Carrell et al. [2] nutzen in einer Studie, in Zusammenarbeit mit der U.S. Air Force Academy, zur Einschätzung der Studenten Daten aus beiden Kategorien. Neben verschiedenen Durchschnittsnoten wie dem GPA[1] oder auch den SAT Scores[2] werden zudem persönliche Eigenschaften wie die oben aufgeführten Alter, Geschlecht und so weiter berücksichtigt.

Khasanah und Harwati[10] verwendeten ebenfalls Daten aus beiden Kategorien und verglichen in ihrer Studie Einteilungen, welche mit Bayesschen Netzen und Entscheidungsbäumen getätigt wurden. Dabei schnitten Bayessche Netze besser ab, sie lieferten genauere Resultate. Außerdem untersuchten sie noch die einzelnen Kriterien auf ihre Aussagekraft in den jeweiligen Schätzern. Die Durchschnittsnote GPA sowie die Anwesenheit im ersten Semester wurden hierbei am höchsten eingestuft.

Kumar et al. verwendeten unter anderem Noten aus vorhergehenden Semestern und versuchten damit die Abschlussnoten vorherzusagen. Hierfür setzten sie verschiedene Entscheidungsbaumalgorithmen ein. ID3, C4.5 und CART erreichten dabei eine Genauigkeit von 45-56 % wobei der CART Algorithmus die höchste Genauigkeit erreichte. Ahadi et al. [1] verglichen in ihrer Studie unterschiedliche Klassifizierer, sie betrachteten dabei Bayessche, Entscheidungsbäume sowie regelbasierte Klassifizierer. Die Genauigkeit lag je nach gewähltem Algorithmus im Bereich von 75 - $<$85 %. Am genauesten war dabei der Random Forest Algorithmus. Evaluiert an einem anderen Semester erreichte dieser jedoch nur noch 71 %. Die Klassifizierung beruhte erneut auf einer Mischung der beiden Kategorien.

Von größerem Interesse ist auch die Vorhersage von Abbrechern, sei es nur für bestimmte Veranstaltungen oder aber für das gesamte Studium. Dies wurde bereits von diversen Autoren untersucht.

Für die Einschätzung der Studenten sollen zunächst lediglich Daten aus der genutzten Plattform verwendet werden. Dies hatte zur direkten Folge, dass keine persönlichen Eigenschaften zur Gruppenbildung zur Verfügung stehen. Darüber hinaus fehlen auch Leistungseinschätzungen wie Klausurergebnisse und anderweitige Qualifikationen. Somit wurde für die Kompetenzeinschätzung der Studenten ein Klassifikator gebildet, welcher Ergebnisse aus Übungsaufgaben nutzt. Dieser basiert auf einem Bayesschen Netz, dessen Grundlagen in Kapitel 2.3 erläutert werden. Zuzüglich wäre aus den vorhandenen Daten auch noch eine Motivationseinschätzung denkbar, welche unter anderem Login-/Abgabezeiten und Ähnliches nutzten könnte.

2.2 Gruppenbildung

Neben der Einschätzung der Studenten ist auch die spätere Gruppeneinteilungsstrategie von Bedeutung. Hierfür gibt es drei Möglichkeiten:

Homogene Gruppen Studenten haben möglichst ähnliche Werte in den betrachteten Kriterien.

Heterogene Gruppen Studenten haben möglichst unähnliche Werte in den betrachteten Kriterien.

[1] Grade Point Average
[2] Scholastic Assessment Test

Gemischte Gruppen Es gibt sowohl homogene als auch heterogene Gruppen.

Carrell et al. [2] versuchten bei ihrer Studie, in Zusammenarbeit mit der U.S. Air Force Academy, das Abschneiden des untersten Drittels der Studenten, bezogen auf deren GPA, durch eine spezielle Gruppenbildung zu verbessern. Dafür mischten sie diese mit Studenten aus dem oberen Drittel und bildeten Gruppen mit einer Größe von etwa 30 Studenten. Das mittlere Drittel wurde dagegen in homogene Gruppen eingeteilt. Die Erwartungshaltung ging dahin, dass sich die schlechteren Studenten somit verbessern, die beiden anderen Gruppen hingegen sollten im Ergebnis unverändert bleiben. Überraschenderweise verzeichneten sie dabei jedoch einen negativen Effekt. Die Teilnehmer des schlechteren Drittels der Versuchsgruppen schnitten schlechter ab als die in den zufälligen Kontrollgruppen. Das bessere Drittel wurde nicht beeinflusst, während die mittelmäßigen Studenten sich verbessern konnten. Als mögliche Ursache geben die Autoren der Studie an, dass es in den einzelnen Gruppen weitere Untergruppen gab, die sich auf Grundlage von verschiedenen Faktoren wie Sympathie gebildet hatten. Auffällig war eine homogene Verteilung der Studenten bezüglich ihrer eingeschätzten Leistung. Somit fand die erhoffte Interaktion der besseren mit den schlechteren Studenten unter Umständen nicht statt.

Ounnas et al. [11] verglichen elf verschiedene Gruppenbildungssysteme. Diese nutzten unterschiedliche Algorithmen wie Multi-Agenten Systeme oder auch genetische Algorithmen. Sie bemängelten dabei, dass die meisten Systeme nur mit einer fixen Anzahl an Parametern arbeiten konnten, wodurch diese unflexibel wurden. Einige der Systeme nutzen zudem eine opportunistische Gruppenbildung mit einem Fokus auf selbstauswählenden Gruppen. Dadurch sind sie für eine effiziente Lerngruppenbildung generell nicht geeignet. Ein weiteres oft angetroffenes Problem sind verbleibende Studenten, die von den Systemen nicht zugewiesen werden können.

Für diese Arbeit wurde auf den Einsatz von spezifischen Gruppenbildungssystemen verzichtet, stattdessen werden den Einschätzungen Werte zugewiesen und ein einzelner Gesamtwert gebildet. Hierdurch vereinfacht sich die Gruppenbildung zu einem erweiterten Sortierproblem. Dies ermöglicht zukünftig, dass leichter Änderungen an der Gruppenbildung durchgeführt werden können. Da die Bildung des Wertes beliebig ist, kann dieser Teil auch für andere Gruppenbildungen genutzt werden.

2.3 Bayessches Netz

Bei einem Bayesschen Netz handelt sich um ein probabilistisch grafisches Modell benannt nach Thomas Bayes. Ein Bayessches Netz wird repräsentiert durch einen gerichteten azyklischen Graphen, kurz DAG[3]. Dabei entsprechen die Knoten den Zufallsvariablen und die Kanten den Abhängigkeiten der Variablen. Als Elternknoten eines Knoten v werden diejenigen Knoten bezeichnet von welchen er abhängt. Allen Knoten des Graphen ist eine bedingte Wahrscheinlichkeitsverteilung der von ihm charakterisierten Variable gegeben, die die Variable den Elternknoten und ihrer Ereignisse zuordnet. Besitzt ein Knoten keine Eltern, so handelt es sich bei der zugeordneten Wahrscheinlichkeitsverteilung um eine unbedingte Verteilung.

Zusammengefasst lässt sich ein Bayessches Netz G = (V, K) unter anderem wie von Ertel [4, S. 184] definieren:

- V entspricht einer Menge von Variablen und K einer Menge von gerichteten Kanten zwischen diesen Variablen.
- Jede Variable hat endlich viele mögliche Werte.

[3] directed acyclic graph

- Die Variablen stellen zusammen mit den Kanten einen DAG dar, welcher keine Zyklen A→...→ A enthalten darf.
- Für jede Variable A ist die Tabelle der bedingten Wahrscheinlichkeiten $P(A|Eltern(A))$, kurz CPT,[4] angegeben.

Anhand eines bekannten Beispiels von J. Pearl wird nachfolgend sowohl die Konstruktion eines Bayesschen Netzes sowie das Schließen in diesem erläutert. Bob hat in seinem Haus eine Einbruchsalarmanlage installiert. Ferner haben ihm seine beiden Nachbarn, John und Mary, versprochen ihn zu informieren, sofern der Alarm ausgelöst wurde. John ist dabei zuverlässiger als Mary, ruft Bob aber auch manchmal an, obwohl der Alarm nicht losgegangen war. Abgesehen von einem Einbruch kann der Alarm auch durch ein schwaches Erdbeben ausgelöst werden. Dabei besitzen die unterschiedlichen Ereignisse die in Tabelle 2.1 angegebenen Wahrscheinlichkeiten, vgl. Russell [14, S. 437ff.]. Die Wahrscheinlichkeit für einen Einbruch beträgt 0.001, die für ein Erdbeben 0.002.

Einbruch	Erdbeben	P(Alarm)
w	w	0.95
w	f	0.94
f	w	0.29
f	f	0.001

Alarm	P(John)
w	0.90
f	0.05

Alarm	P(Mary)
w	0.70
f	0.01

Tabelle 2.1: Verteilung für das Beispiel

Für die Konstruktion des Netzes bietet es sich zunächst an sämtliche Variablen zu notieren. Dies sind die Ereignisse *Einbruch* und *Erdbeben*, die von Bob beobachtbaren Ereignisse (John und Mary), sowie der *Alarm*. Zur Bestimmung der Kanten im Netz muss überprüft werden, welche Variablen voneinander abhängig sind. Sofern Variable B von A abhängig ist, wird eine Kante von A zu B eingezeichnet A→B.

Ein weiterer wichtiger Begriff in diesem Zusammenhang ist die bedingte Unabhängigkeit zweier Knoten bezüglich eines dritten. Angenommen B ist abhängig von A, und C von B; dies ergibt das Netz A→B→C. Darüber hinaus könnte C auch von A abhängig sein. Ob eine Kante zwischen A und C nötig ist, kann dabei folgendermaßen überprüft werden. Gilt $P(A,C|B) = P(A|B) \cdot P(C|B)$, wird keine Kante von A zu C eingetragen. Ist der Wert der Variable B bekannt, so sind die Variablen A und C unter der Bedingung B unabhängig, siehe Ertel [4, S. 174].

Die Netzstruktur ist abhängig von der Reihenfolge der Variablen. In diesem Beispiel werden die kausalen Zusammenhänge wiedergegeben. Wird eine andere Reihenfolge gewählt entsteht ein anderes Netz. Häufig ist die Konstruktion von und das Arbeiten mit nicht kausalen Netzen komplizierter als mit kausalen Netzen, siehe [4, S. 182]. Das resultierende Netz ist in Abbildung 2.1(a) zu finden.

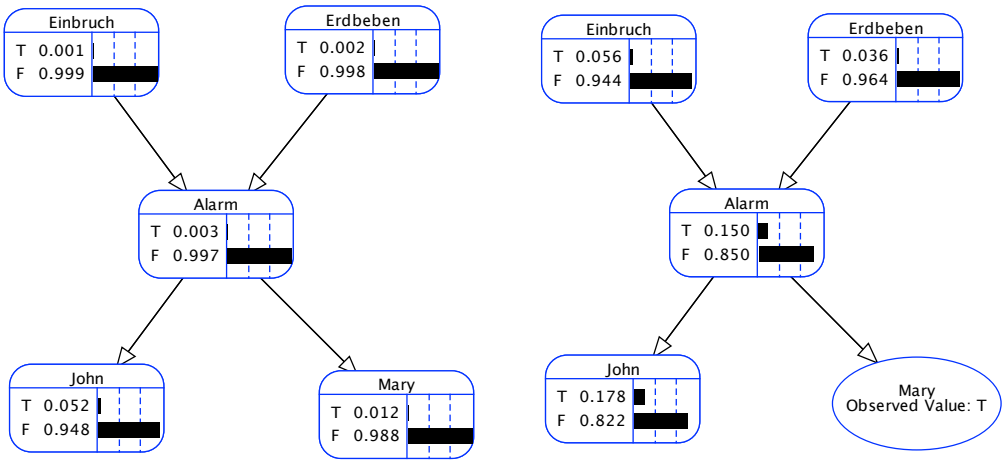
Zu Beginn werden die Knoten bzw. Variablen in das Netz eingetragen, die keine Eltern besitzen. Dies sind in diesem Fall Einbruch und Erdbeben. Nach jedem Eintragen von Knoten werden diese auf Unabhängigkeit geprüft. In diesem Fall sind Erdbeben und Einbruch unabhängig. Nachfolgend wird der Knoten eingefügt, der direkt von Einbruch und Erdbeben abhängt; Alarm. Da dieser abhängig ist, werden Kanten zwischen Einbruch und Alarm sowie zwischen Erdbeben und Alarm eingefügt.

Als Letztes bleiben noch John und Mary übrig. Da diese den Alarm beobachten, sind sie von diesem abhängig. Sowohl John als auch Mary sind bedingt unabhängig zu Einbruch und Erdbeben bezüglich Alarm. Sie reagieren nicht auf einen Einbruch oder ein Erdbeben selbst,

[4] conditional probability table

sondern lediglich auf den Alarm. Des Weiteren sind John und Mary bedingt unabhängig bezüglich Alarm. Somit wird zwischen den beiden auch keine Kante eingetragen.

Abschließend müssen noch die CPTs für die einzelnen Knoten ermittelt werden. Dies kann zum einen manuell erfolgen oder alternativ auf Basis der vorhandenen Daten automatisch geschätzt werden, was sich speziell bei mehreren Variablen anbietet. Neben dem Lernen für die CPTs ist es auch möglich die Struktur eines Netzes automatisch aus Daten lernen zu lassen. Aufgrund von inadäquater Daten oder einer zu geringen Datenmenge können die daraus resultierenden Netze jedoch unbrauchbar sein. Die Schwierigkeiten beim Strukturlernen von Bayesschen Netzen sind beispielsweise unter [4, S. 234ff] beschrieben.



(a) Bayessches Netz des Beispiels

(b) Netz mit Beobachtung Mary

Abbildung 2.1: Netz des Beispiels mit und ohne Beobachtung

Im Anschluss an die Konstruktion des Netzes kann dieses nun zum Schließen genutzt werden. Zu Beginn kann man dem Netz für alle Variablen die Wahrscheinlichkeiten ihrer Ereignisse entnehmen, siehe Abbildung 2.1(a). Die Wahrscheinlichkeit für einen Alarm beträgt folglich 0.003. Beobachtet man ein Ereignis oder möchte die Wahrscheinlichkeiten unter der Bedingung eines oder mehrerer Ereignisse eruieren, so können Evidenzen im Netz vermerkt werden. Dies führt dazu, dass die neuen Erkenntnisse im Netz kommuniziert werden. Hierfür gibt es verschiedene Vorgehensweisen. Ein Algorithmus ist zum Beispiel im 4. Kapitel von Pearl [12, S. 143ff] gegeben. Demzufolge werden die Wahrscheinlichkeiten für die einzelnen Variablen und ihrer Ereignisse angepasst. Hierdurch ist anschließend eine Vorhersage für die einzelnen Variablen unter Berücksichtigung der vermerkten Beobachtung möglich. Ein Beispiel für die Beobachtung „Mary ruft an“ ist in Abbildung 2.1(b) gegeben. Die Wahrscheinlichkeit für einen Alarm hat sich durch die Beobachtung „Mary ruft an“ auf 0.150 erhöht.

Eine beispielhafte Anwendung von Bayesschen Netzen findet sich in der Darstellung von Expertenwissen [7, S. 198]. Die Arbeitsweise der Experten beschreibt das generelle Verhalten solcher Netze verständlich, siehe Abbildung 2.2. Experten treffen auf Grundlage ihrer Erfahrungen Entscheidungen und erwarten gewisse Resultate. Diese stellen sich im späteren Verlauf als richtig oder falsch heraus. Diese Ergebnisse sind die Lerngrundlage für zukünftige Entscheidungen. Häufig treffen Experten Entscheidungen auf Basis von unvoll-

ständigen, unsicheren Daten. Zur Unterstützung der Experten bieten sich beispielsweise Bayessche Netze an um Entscheidungen nicht nur zu treffen sondern auch zu begründen.

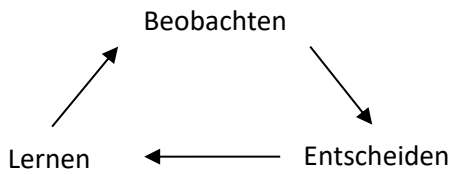


Abbildung 2.2: Arbeitsweise von Experten nach Jensen [8, S. 1]

2.4 Knowledge Space Theory

Die Knowledge Space Theory (KST) beschreibt eine umfassende Theorie der Wissensrepräsentation und -bewertung. Sie wurde von Doignon, Falmagne und ihren Mitarbeitern in den 1990er Jahren entwickelt. Die Theorie geht davon aus, dass sich Wissen durch Aufgaben abprüfen lässt. Verschiedene Aufgaben erfordern verschiedenes Wissen von Studenten. Das Wissen ist hierarchisch geordnet. Durch das Lösen einer Aufgabe weist der Student nicht nur das Wissen über den Bereich nach, sondern auch über alle Teilbereiche des Bereiches. Ausgehend von Rechenaufgaben braucht der Student zum Beispiel Wissen über Multiplikation und Division. Diese könnten noch weiter unterteilt werden in Prozentrechnung, einstellige und mehrstellige Multiplikation und so weiter. Nachfolgend sind 4 Beispielaufgaben a-d angegeben, siehe [16, S. 492]

a.) 4 x 7 = ? b.) 1/4 x 1/7 = ? c.) 0.4 x 7 = ? d.) 40 % von 7 = ?

Der Wissensstand (Knowledge State) des Studenten ist dabei die Menge der richtig beantworteten Fragen. Hierbei gilt es zu beachten, dass weder Zeitdruck, emotionaler Aufruhr, Flüchtigkeitsfehler noch zufälliges Beantworten und ähnliches berücksichtigt werden. In der Regel ist der Wissensstand nicht direkt messbar, siehe Doignon [3, S. 3]. Nicht alle Teilmengen entsprechen dabei möglichen Zuständen. Kann der Student Aufgabe d lösen, so kann davon ausgegangen werden, dass er neben Prozentrechnung zudem exemplarisch einstellige Multiplikation beherrscht. Demzufolge sollte er zumindest auch die Aufgabe a lösen können.

Als Wissensstruktur (Knowledge Structure) wird ein Paar (Q,K) bezeichnet. Hierbei entspricht Q der Domäne bestehend aus allen Fragen, K einer Anzahl an Teilmengen mit möglichen Zuständen, mindestens aber der {} sowie Q, siehe [3, S. 18]. Eine Beispielstruktur ist in Abbildung 2.3 für die obigen Aufgaben a-d angegeben. Ein von Kambouri durchgeführtes Experiment zeigte, dass die Anzahl an möglichen Zuständen deutlich geringer als das theoretische Maximum von 2^n ist. 5 Experten konstruierten dabei eine Struktur für 50 gymnasiale Rechenaufgaben, die Ergebnisse reichten dabei von etwa 900 bis 8000 Zuständen, siehe Kambouri [9, S. 134].

Eine Wissensstruktur wird Wissensraum (Knowledge Space) genannt, wenn sie unter Vereinigung geschlossen ist. Sofern zwei Elemente Mitglied der Struktur sind, so muss auch ihre Vereinigung enthalten sein. Die in Abbildung 2.3 dargestellte Struktur weist einige weitere Besonderheiten auf. Sie beginnt mit der leeren Menge und wird von links nach rechts kontinuierlich um ein Element erweitert, bis letztendlich die Gesamtmenge erreicht

ist. Ein Pfad gilt als Abstufung (gradation) wenn sich die Anzahl an Zuständen immer exakt um eins erhöht. Sofern jeder Wissenszustand in mindestens einer Abstufung enthalten ist, handelt es sich dabei um einen gut abgestuften (well-graded) Wissensraum; so wie bei dem vorhandenen Beispiel, vgl. Villano [16, S. 491ff.].

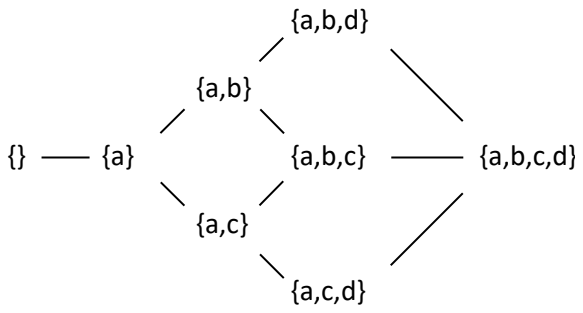


Abbildung 2.3: Beispielstruktur für Aufgaben a-d

KAPITEL 3

Methode

Im nachfolgenden Kapitel werden die Umsetzungen der beiden Anforderungen Lerngruppenbildung bzw. generell Gruppenbildung, sowie die Kompetenzschätzung erläutert. Zudem sollen die beiden Bestandteile in die Lehrstuhl eigene Projektverwaltungswebsite integriert werden. Hierauf wird explizit in Kapitel 4 eingegangen. Eingangs wird kurz der zur Verfügung stehende Datensatz vorgestellt. Dieser stammt aus einer Informatikveranstaltung. Anschließend wird die Konstruktion des Kompetenzschätzers erklärt. Dieser wurde ausgehend von den Daten konstruiert. Abschließend wird die Methodik zur Gruppenbildung vorgestellt.

Dabei werden drei Begriffe bezüglich der Bewertung einer Codeabgabe verwendet. Diese bilden aufeinander aufbauende Stufen, die der Student für eine korrekte Abgabe erreichen muss. Siehe hierfür das Kapitel 2.4 über Knowledge Space Theory.

form Die *Form* einer Abgabe ist korrekt, wenn die Abgabe maschinell auswertbar ist. Hierfür muss zum einen ein auswertbarer Dateityp verwendet werden, zum anderen die Benennung der Dateien den vorgegebenen Richtlinien[1] entsprechen.

syntax Eine Abgabe ist *syntaktisch* korrekt, sofern sie in der jeweiligen Sprache syntaktisch korrekt ist und somit kompilierbar ist.

semantic Eine Abgabe ist *semantisch* korrekt, wenn die zugehörigen Tests erfüllt werden. Hierfür gibt es für jede Aufgabe auch eine Testdatei, welche die Abgabe prüft.

3.1 Verwendete Daten

Die zugrunde gelegten Daten stammen aus einer Wiederholungsveranstaltung des Lehrstuhls für Programmier- und Modellierungssprachen der LMU aus dem Wintersemester 2017/18. Die Veranstaltung diente zur Wiederholung und Vorbereitung auf die Zweitklausur einer Vorlesung zur Einführung in die funktionale Programmierung des vorherigen Semesters. Die Teilnahme an der Veranstaltung sowie an den Übungen erfolgte auf freiwilliger Basis. In den Übungen der Veranstaltungen sollten die Studenten meist mittels kleineren Programmieraufgaben den Vorlesungsstoff wiederholen. Dabei konnten die Studenten ihre

[1] Kennzeichnung der Testdateien; korrekte Nutzung von Modulen in den jeweiligen Programmiersprachen

Abgaben auf der Plattform „Backstage“ hochladen. Über diese wurde die gesamte Veranstaltung verwaltet. Für jede Programmieraufgabe wurden zusätzlich noch Testdateien angegeben, mit denen die Studenten ihren Haskell Code testen konnten. Mittels dieser Testdateien wurden die Abgaben der Studenten ausgewertet. In Abbildung 3.1 ist das Abgabeverhalten der Studenten während der Veranstaltung angegeben. In jeder Übung gab es für gewöhnlich mehr als eine Aufgabe.

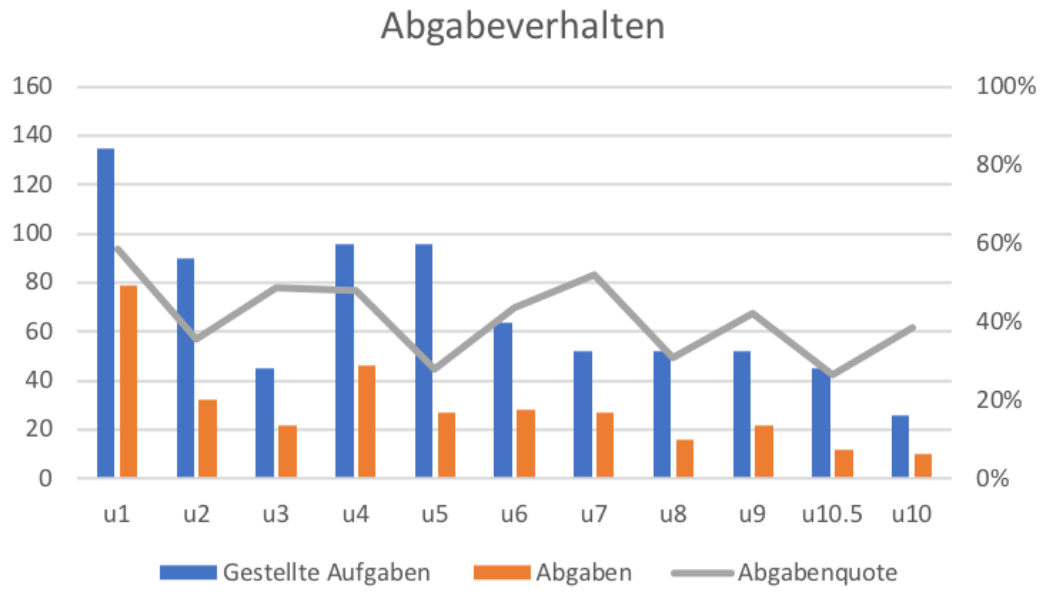


Abbildung 3.1: Abgabeverhalten Tutorium Programmierung und Modellierung 2017/18

Die blauen Balken stellen dabei die in Summe gestellten Aufgaben dar. Die orangen Balken repräsentieren die Abgaben und die graue Linie reflektiert, wie viel Prozent der gestellten Aufgaben abgegeben wurden. Sofern ein Student an der Veranstaltung nicht weiter teilnehmen wollte, wurde dies berücksichtigt und daraus resultierend auch keine Aufgaben für diesen in nachfolgenden Übungen erstellt. Dies erklärt warum die Anzahl an Aufgaben über die Veranstaltung hinweg abnimmt. Zu Beginn nahmen etwa 40 Studenten an der Veranstaltung teil, gegen Ende lediglich circa 15. Insgesamt wurden 753 Aufgaben gestellt, von diesen wurden 321 abgegeben. Das entspricht einem Verhältnis von etwa 43 %. Die Balken beziehen sich bei allen Diagrammen auf die linke y-Achse, die Prozent Linie auf die rechte Achse.

In Abbildung 3.2 ist die Auswertbarkeit der einzelnen Abgaben je Übung abgebildet. Die einzelnen Abgaben konnten nur ausgewertet werden, sofern es sich bei den Aufgaben um Programmieraufgaben handelte. Andere Aufgaben sind in der Abbildung gelb markiert. Sofern es sich um eine fehlerhafte Abgabe handelt, zum Beispiel falscher Dateityp oder fehlerhafte Benennung der Dateien, ist dies in rot gekennzeichnet. In grün sind die verwertbaren Abgaben markiert. Die Gesamtsumme der Kategorien entspricht dabei den Abgaben aus der Abbildung 3.1. In grau ist der Prozentsatz auswertbarer Abgaben zu verwertbaren Abgaben eingetragen. Von den 321 Abgaben erfolgten 252 zu auswertbaren Aufgaben. Insgesamt gab es dabei 25 fehlerhafte Abgaben. Somit konnten 90% der Abgaben ausgewertet werden. Bis auf die letzten beiden Übungen gab es immer nicht auswertbare Abgaben.

In Abbildung 3.3 sind die Ergebnisse der Auswertung der Abgaben abgebildet. In

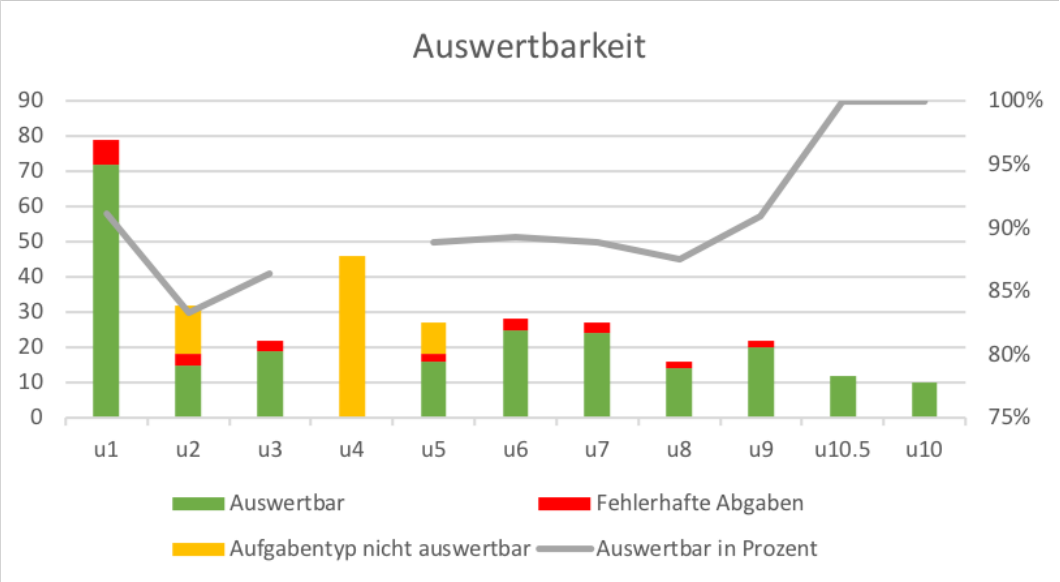


Abbildung 3.2: Auswertbarkeit Tutorium Programmierung und Modellierung 2017/18

blau sind dabei die bereinigten Abgaben eingetragen. Dies entspricht allen Abgaben zu auswertbaren Aufgaben. Diese wurden in der Abbildung 3.2 in grün und rot eingetragen. Die weiteren drei Balken geben an, welche Stufe die Abgabe erreicht hat. Dies ist in orange, lila und gelb angeben. Wie bereits beschrieben bauen diese aufeinander auf. Die orangen Balken, *Form* korrekt, entsprechen dabei den grünen Balken aus Abbildung 3.2 und beschreiben die Abgaben, die ausgewertet werden konnten. Die lila Balken zeigen die Abgaben, die *syntaktisch* korrekt sind. Die gelben stellen die *semantisch* korrekten Abgaben dar. Auffällig ist, dass bis auf Übung 2 alle Abgaben, die *syntaktisch* korrekt sind ebenfalls *semantisch* korrekt sind. Zu beachten ist, dass bei diesem Datensatz immer alle Tests erfolgreich waren oder alle Tests fehlgeschlagen sind. Die graue Linie gibt hier an, wie viel Prozent der bereinigten Abgaben *semantisch* korrekt sind. 28% der bereinigten Abgaben waren dabei *syntaktisch* korrekt und 26% *semantisch*.

In Tabelle 3.1 sind nochmals alle Statistiken über alle Übungen absolut und teilweise prozentual angeben. Abgaben in Prozent bezieht sich dabei auf die Spalte „Aufgaben“ und die anderen Spalten auf die Spalte „Bereinigt“.

	Aufgaben	Abgaben	Bereinigt	Fehlerhaft	Form	Syntax	Semantik
Absolut	753	321	252	25	227	70	66
Prozent		43%		10%	90%	28%	26%

Tabelle 3.1: Statistik über alle Übungen

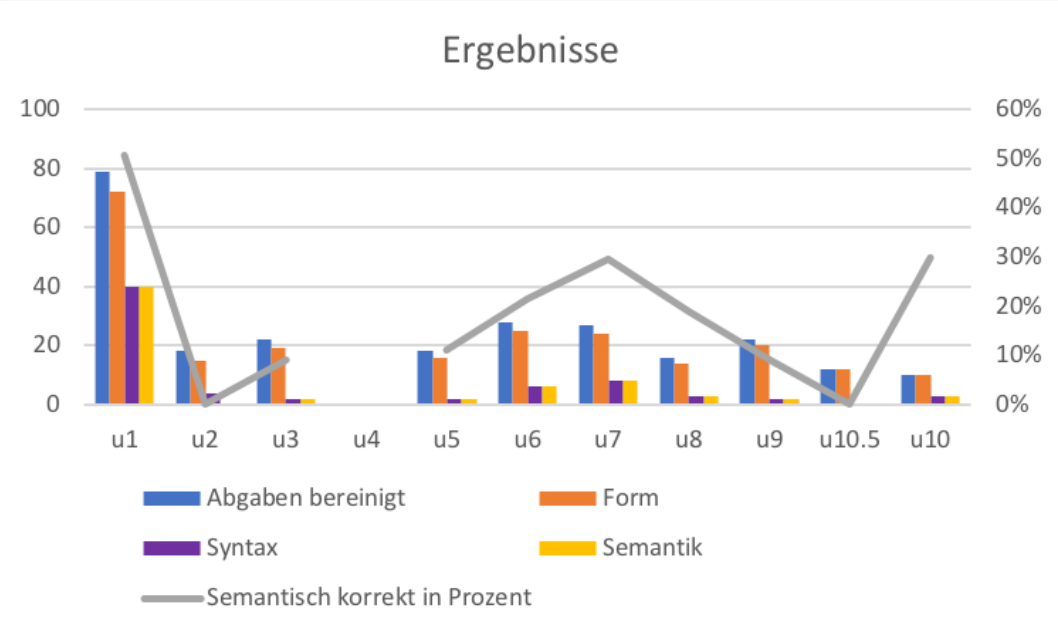


Abbildung 3.3: Ergebnisse Tutorium Programmierung und Modellierung 2017/18

3.2 Kompetenzschätzer

Der Kompetenzschätzer soll anhand der in Kapitel 3.1 vorgestellten Daten eine Kompetenzeinschätzung ermöglichen. Ferner sollte er in Zukunft um zusätzliche Faktoren erweiterbar sein.

Aufgrund dessen, dass immer alle Tests entweder erfolgreich oder aber nicht erfolgreich waren, kann eine Abgabe semantisch korrekt oder nicht semantisch korrekt sein. Somit wird ein Klassifikator gesucht, der zukünftige Abgaben der Studenten in eine der beiden Kategorien einteilen kann.

Unter Berücksichtigung der beiden Kategorien aus Kapitel 2.1 persönliche Leistung und persönliche Eigenschaften sollte darauf geachtet werden, dass ein Klassifikator auch mit Faktoren zurechtkommt, die nur geschätzt werden können. Einerseits können diese dabei ungenau sein, somit ist der Wahrheitsgehalt unsicher, andererseits könnten diese gegebenenfalls auch ganz fehlen. Beachtet man dies nicht, so kann es zu Problemen kommen wie unter anderem dieses Beispiel mit folgenden Aussagen zeigt:

1. Hat ein Student die Vorlesung Einführung in die Programmierung gehört so kann er Java

2. Alice hat die Vorlesung Einführung in die Programmierung nicht gehört.

Aus 1. und 2. lässt sich nicht schließen ob Alice Java kann, da nicht zwangsläufig gilt „Hat ein Student die Vorlesung Einführung in die Programmierung **nicht** gehört so kann er **nicht** Java“. Somit kann aus den vorliegenden Daten keine Aussage getroffen werden. Daneben könnte die erste Aussage auch ungenau sein, es gibt einige wenige Ausnahmen.

Aufgrund der Anforderungen für den Kompetenzschätzer und die soeben angeführten Limitationen wurde ein Bayessches Netz als Grundlage gewählt. Zur Repräsentation des Wissens wird die Knowledge Space Theory genutzt. Mit ihrer Hilfe können auch zukünftig

kompliziertere Wissensstrukturen kompakt in das Netz integriert werden. Konnte ein Student bereits einen syntaktisch korrekten Code abgeben, so ist damit zu rechnen, dass er auch zukünftig zumindest Code abgibt, dessen *Form* korrekt ist.

Ausgehend von den Daten kann eine Abgabe in eine von vier Kategorien eingeteilt werden: die Abgabe ist nicht *Form* korrekt, die Abgabe ist *Form* korrekt, die Abgabe ist *syntaktisch* korrekt und die Abgabe ist *semantisch* korrekt. Im Sinne der in Kapitel 2.4 vorgestellten Knowledge Space Theory entspricht das der Struktur in Abbildung 3.4.

$$\{\} - \{\text{form}\} - \{\text{form, syntax}\} - \{\text{form, syntax, semantic}\}$$

Abbildung 3.4: Wissensstruktur für Übungsaufgaben

Dies in ein Bayessches Netz übertragen führt zu dem in Abbildung 3.5 dargestelltem Ausgangsnetz.

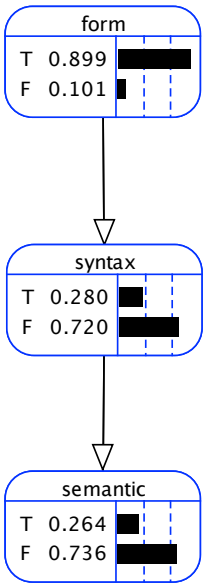


Abbildung 3.5: Ausgangsnetz

Mithilfe dieses Netzes ist es möglich vorherzusagen wie wahrscheinlich eine Abgabe *semantisch* korrekt ist unter der Bedingung, dass sie *Form* korrekt oder sogar *syntaktisch* korrekt ist. Liegt bspw. eine Evidenz für *Form* Korrektheit vor, so beträgt die Wahrscheinlichkeit für *syntaktische* Korrektheit 31 Prozent und die für *semantische* nun 29 Prozent. Liegt eine Evidenz für *syntaktische* Korrektheit vor, so ändert sich die Wahrscheinlichkeit für *semantische* Korrektheit auf 94 Prozent. Aufgrund der Konstruktion des Netzes mit der Knowledge Space Theory muss für eine Evidenz von *syntax* auch eine Evidenz von *form* vorliegen. Dies wird im Bayesschen Netz gut abgebildet. Zum einen verändert sich die Wahrscheinlichkeit

von *form* unter der Bedingung *syntax* zu 99 Prozent, nicht auf 100 Prozent aufgrund der Unsicherheit. Zum anderen ist *semantisch* korrekt und *Form* korrekt bedingt unabhängig bezüglich *syntaktisch* korrekt; es existiert keine Kante zwischen *semantisch* korrekt und *Form* korrekt. Folglich hätte eine Evidenz für nicht *form* korrekt (beziehungsweise keine Evidenz oder *form* korrekt) und *syntaktisch* korrekt das gleiche Resultat, da eine Evidenz für *syntaktisch* korrekt eine Evidenz für die *Form* überstimmt.

Das eigentliche Ziel der Kompetenzschätzung ist aber eine Vorhersage für eine neue Übung zu treffen, nicht für eine Übung, die bearbeitet wurde und teilweise ausgewertet wurde, siehe oben. Somit muss das Netz erweitert werden. Es bietet sich in diesem Fall an, den Verlauf der von einzelnen Studenten erreichten Stufen zu betrachten. Es soll ermittelt werden, wie erfolgreich sie bei vorherigen Übungen waren eine der drei Stufen zu erreichen. Hierfür werden nun drei weitere Beobachtungen als Knoten in das Netz eingefügt. Diese geben jeweils die „History“ der Stufen an. Diese kann dabei negativ, neutral oder positiv sein. Intern wird dies über eine Zahl repräsentiert. Erreicht der Student dabei eine Stufe so erhält er einen Pluspunkt. Scheitert er bei einer Aufgabe an einer Stufe resultiert dies in einem Minuspunkt innerhalb der dazugehörigen History. Hierdurch wird grob modelliert wie häufig diese Stufe erreicht wurde. Dabei wird das Ergebnis wie in Formel 3.1 berechnet.

$$\text{history}_{n+1} = \text{newValue} + \text{historyFactor} * \text{history}_n \tag{3.1}$$

Dadurch ist sichergestellt, dass neue Ereignisse stärker gewichtet werden als ältere, diese jedoch nicht komplett vernachlässigt werden. Der History Faktor beträgt dabei 0.5. Abhängig wie stark neue Werte bevorzugt werden sollen, kann dieser größer oder kleiner ausfallen. Zu Beginn werden alle Werte auf 0 gesetzt. Erreichte ein Student in zwei vorherigen Übungen zunächst nicht das Ziel und anschließend gelang ihm dies, führt das zu einem Faktor von $h_{ft} = 0.5$. Zum Vergleich ist der Wert für das umgedrehte Ereignis $h_{tf} = -0.5$.

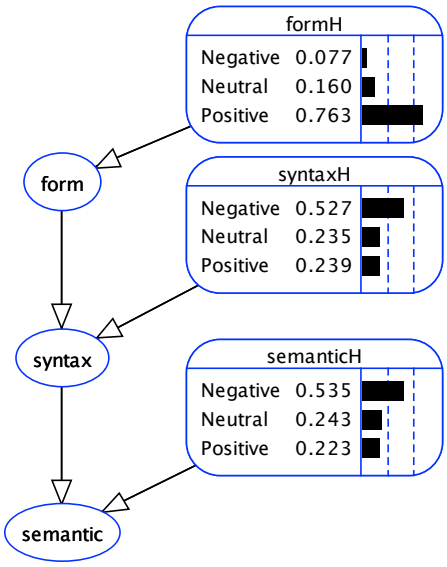


Abbildung 3.6: Netz mit Histories

Die drei Kategorien ergeben sich dabei folgendermaßen: Die History ist negativ wenn der Wert kleiner -0.5 ist. Liegt der Wert zwischen -0.5 und 0.5 ist er neutral. Ist er größer als 0.5 ist er positiv. Das daraus resultierende Netz ist in Abbildung 3.6 abgebildet. Liegt beispielsweise ein positiver History Faktor für *syntax* vor so erhöht sich die Wahrscheinlichkeit für *semantische* Korrektheit auf 48 Prozent während sie sich bei einem negativen Faktor auf 14 Prozent verringert.

Nicht alle Aufgaben besitzen die selbe Komplexität. Abhängig von der Schwierigkeit der einzelnen Aufgaben kann das zu erwartende Resultat unterschiedlich ausfallen. Dies wird im Netz 3.7 durch den neuen Knoten „difficulty“ modelliert. Dieser wird, ähnlich wie die History Knoten, in drei Bereiche eingeteilt. Da die Daten vor dem Beginn dieser Arbeit erhoben wurden, fiel die Wahl auf eine automatische Einschätzung der Schwierigkeit für die einzelnen Aufgaben. Zukünftig könnte die Einstufung beim Erstellen der Aufgaben vom Aufgabensteller angegeben werden, um die Aussagekraft zu erhöhen. Die Schwierigkeit einer Aufgabe wird als „low“ angesehen, sofern mindestens 66 Prozent der Studenten diese Aufgabe lösen konnten. Die Abgaben sind somit semantisch korrekt. Die Schwierigkeit der Aufgabe wird als „medium“ angesehen, wenn zwischen 33 und 66 Prozent der Studenten die Aufgabe lösen konnten. Waren weniger als 33 Prozent der Studenten in der Lage die Aufgabe zu lösen, so wurde die Schwierigkeit als „high“ eingeschätzt. Die Berechnung der Schwierigkeit erfolgte somit nach der Auswertung der Abgaben. 39 Prozent der Aufgaben wurden als „medium“ deklariert und 61 Prozent als „high“. Keine der gestellten Aufgaben wurde nach dieser Regel als leicht eingeschätzt. Nur auf Grundlage der Schwierigkeit verändert sich die Wahrscheinlichkeit für *semantische* Korrektheit, bei mittlerer Schwierigkeit auf 42 Prozent. Geht man von einer schweren Aufgabe aus, so beträgt diese 17 Prozent.

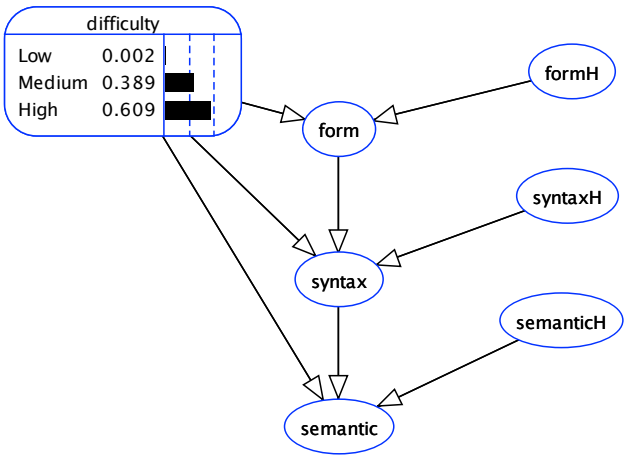


Abbildung 3.7: Netz mit Histories und Difficulty

Zukünftig könnte das Netz um weitere Knoten erweitert werden, insbesondere um persönliche Eigenschaften, welche in der Projektverwaltungswebsite momentan nicht vorliegen. Da es für einige Veranstaltungen verpflichtende Voraussetzungen gibt, könnten so von

Studenten weitere Daten der Plattform zugänglich gemacht werden, die ebenfalls in einem zukünftigen Netz integriert werden könnten. Hierunter fallen unter anderem Klausurnoten.

3.3 Gruppenbildung

Die Gruppenbildung soll für verschiedene Szenarien nutzbar sein. Hierzu zählt die Bildung von homogenen, heterogenen sowie gemischten Gruppen. Diese können unter anderem für Gruppenarbeiten wie Programmierpraktika oder Übungsgruppen genutzt werden. Des Weiteren soll es möglich sein Peer Review Konstellationen zu generieren. Hierbei ist das Ziel, dass die besseren Studenten die schlechteren unterstützen sollen. Außerdem soll darauf geachtet werden, dass alle Gruppen maximal gefüllt sind. Dies bedeutet, dass mehrere Gruppen mit einem fehlenden Mitglied, einer Gruppe mit mehreren fehlenden Mitgliedern, gegebenenfalls sogar nur einem einzigen, vorzuziehen sind. Um möglichst flexibel hinsichtlich zukünftiger Faktoren zu sein, werden die Faktoren zu einem einzelnen Wert addiert. Dies kann eine beliebige Funktion sein, somit kann die Gewichtung nach Wunsch selbst vorgenommen werden. Diese sollte dabei als Eingabe einen Studenten nehmen und eine Zahl ausgeben.

```
function: Student => Int
```

Listing 3.1: Abbildungsfunktion

Ausgehend vom Ergebnis des Kompetenzschätzers könnte die Funktion so aussehen:

```
    def f(u: User): Int = {
      (predictions(u.id) * 10).toInt
}
```

Listing 3.2: Beispiel Abbildungsfunktion Student zu Integer

In der Datenstruktur predictions sind hierbei die Vorhersagen zu den einzelnen Usern (Studierenden) enthalten. Diese liegen im Bereich von 0-1. Mithilfe des Faktors 10 werden die User entzerrt, was im späteren Verlauf für die heterogene Gruppenbildung noch wichtig wird.

Homogene Gruppierung Durch die Nutzung eines einzelnen Wertes verringert sich die Komplexität des Problems signifikant. Die Liste an Studenten, die in eine Gruppe eingeteilt werden soll, wird zunächst auf- oder absteigend anhand der ermittelten Werte sortiert. Anschließend muss berechnet werden wie viele volle und nicht volle Gruppen benötigt werden.

```
val groupCount = math.ceil(seq.size / size.toDouble).toInt
var nonFullGroupCount = seq.size % size
if (nonFullGroupCount != 0) {
   nonFullGroupCount = size - nonFullGroupCount
}
```

Listing 3.3: Berechnung der Anzahl von vollen und nicht vollen Gruppen

Zunächst wird ermittelt, ob die Studenten (seq) ohne Rest auf die gewünschte Gruppenstärke (size) verteilt werden können. Ist dies nicht der Fall, wird ausgehend vom Rest der Division die Anzahl an nicht vollen Gruppen mit exakt einem Mitglied weniger ermittelt. Anschließend können nun die vollbesetzten und nicht vollbesetzten Gruppen mit Mitgliedern befüllt werden.

Heterogene Gruppierung Das Ziel bei einer heterogenen Verteilung ist, dass jede Gruppe in etwa gleich „stark“ ist. Jeder Student besitzt einen Wert, der ihm zugeordnet wurde. Alle Werte sämtlicher Mitglieder einer Gruppe werden zusammengezählt und so der Gruppenwert gebildet. Zudem wird der optimale Wert für jede Gruppe ermittelt. Dies berechnet sich wie folgt:

```
1   val perfectGroupScore = sum / groupCount
```

Listing 3.4: Berechnung des optimalen Scores

Die Summe aller Werte sämtlicher Studenten dividiert durch die Anzahl an Gruppen. Je geringer die Abweichung der einzelnen Gruppen zu diesem Wert ist, desto ausgeglichener sind die Gruppen. Ähnlich, wie bei der homogenen Gruppierung, muss die Anzahl an vollen und nicht vollen Gruppen ermittelt werden. Anschließend werden die Studenten zufällig auf die einzelnen Gruppen verteilt. Als Nächstes wird versucht Gruppen mit sehr hohem Gesamtwert mit Gruppen mit sehr niedrigem Gesamtwert auszugleichen. Dies wird durch folgenden Code beschrieben:

```
1  while(true){
2     // find indices from groups to swap elements
3     val indices = getIndices(groups)
4     if (indices == null) {
5        // if there is no pair
6        break
7     }
8     // try to swap one element each to reduce distance to perfect
       score
9     val groupTupel = swap(indices._1, indices._2)
10    currentGroups(indices._1) = groupTupel._1
11    currentGroups(indices._2) = groupTupel._2
12
13    val newScore = calcGroupScore(currentGroups).sum
14    if (newScore > score) {
15       // if score got worse we are finished
16       break
17    }
18    score = newScore
19    bestGroups = currentGroups.clone()
20 }
```

Listing 3.5: Gekürzter Code zur Erstellung heterogener Gruppen

Dabei müssen einige Gegebenheiten beachtet werden. Zunächst wird immer versucht die Gruppen mit dem höchsten bzw. niedrigsten Wert auszugleichen. Hierbei wird nach der Kombination von Elementen gesucht, die den größten Effekt beim Ausgleich hat. Aufgrund von ungünstigen Konstellationen kann es vorkommen, dass ein Tausch von Mitgliedern die Abweichung vom optimalen Wert nicht verbessert. Findet sich keine Konstellation, die das Ergebnis verbessert, so werden die beiden Gruppen für einen Tausch untereinander vorübergehend gesperrt. Konnten keine weiteren Gruppen zum Tausch gefunden werden, entweder weil alle den optimalen Score haben oder weil alle Möglichkeiten momentan gesperrt sind, wird die Schleife beendet. Ebenso verhält es sich sollte sich der Score verschlechtern. Konnten zwei Gruppen erfolgreich Mitglieder tauschen, so werden alle gesperrten Kombinationen, in denen eine der beiden Gruppen vorkommt, wieder entsperrt.

Peer Reviews Ziel hierbei ist es, dass jeder Abgabe zwei Studenten, sowie jedem Studenten zwei Abgaben zur Korrektur zugewiesen werden. Für die Generierung von Peer Reviews ist es wieder nötig die Liste an Studenten, ausgehend des von ihnen zugewiesen Wertes, zu sortieren. Nach der Sortierung der Liste werden die einzelnen Paare erstellt. Dabei muss darauf geachtet werden, dass etwaige Spezialfälle für die rekursive Ausführung behandelt werden. Dies sind zunächst die Fälle, in denen die Liste weniger als 2 Elemente enthält. In diesem Fall kann keine Zuordnung durchgeführt werden. Enthält die Liste nur 2 Elemente so kann lediglich ein Paar gebildet werden. Enthält die Liste 3, 4, 5 oder 6 Elemente ist dies der letzte Schritt. Enthält sie noch mehr als 6 Elemente werden immer die beiden ersten und letzten Elemente der Liste entfernt und untereinander Zuweisungen erstellt. Ausgehend von der Liste abcd werden die Paarungen ac, ad, bc und bd erstellt. Die Spezialfälle 5 und 6 sind nötig um in dem darauffolgenden Schritt die Fälle für 1 oder 2 Elemente zu vermeiden. Ansonsten können die gewünschten Vorgaben nicht erreicht werden.

KAPITEL 4

Implementierung

In diesem Kapitel wird auf die Implementierung der einzelnen Bestandteile der Arbeit eingegangen. Zunächst wird in Kapitel 4.1 eine kurze Einführung zur Backstage Plattform gegeben. Anschließend wird in Kapitel 4.2 auf den `CompilingService` eingegangen. Dieser stellt ein Interface zur Nutzung des Compiling Services Coconut 2 bereit, durch welchen einzelne Programme verschiedener Sprachen mit REST Anfragen kompiliert und ausgeführt werden können. In Kapitel 4.3 wird die Implementierung des Prädiktors, des Schätzers, erläutert. Dabei wird das bereits in Backstage vorhandene `PredictionFramework` genutzt. Daneben wird für die Realisierung des Bayesschen Netzes das Weka Framework verwendet. Abschließend wird in Kapitel 4.4 der Service zur Gruppenbildung behandelt. Für alle drei Bestandteile wird deren Funktion und Aufbau beschrieben, sowie deren Nutzung aufgezeigt.

4.1 Einführung

Bei der Backstage[1] Plattform handelt es sich um ein Forschungsprojekt der Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen der Ludwig-Maximilians-Universität München. Die Plattform bietet einen digitalen Backchannel für große Vorlesungen. In der neuen Version Backstage 2 wurde die Plattform unter anderem um den bereits angesprochenen Teil zur Projektverwaltung erweitert.

Durch diesen lassen sich exemplarisch Kurse wie Vorlesungen, Übungen und Praktika verwalten. Ein Kurs entspricht dabei einem Projekt. Jedes Projekt kann noch weitere Projekte als sogenannte Unterprojekte (auch Subprojekte) besitzen. Diese können zum Beispiel verwendet werden, um einzelne Übungen und deren Dokumente sowie Abgaben der einzelnen Studenten zu gruppieren.

Der Teil der Plattform zur Projektverwaltung ist in Scala geschrieben und nutzt unter anderem das `Play Framework`.[2] Zur Kommunikation zwischen Client und Server stellt dieses `RESTful Webservices` bereit.

In die Backstage Plattform wurden bereits einige studentische Arbeiten integriert. Darunter der Compiling Service Cocunut 2, mit welchem die Studenten ihre Abgaben im Frontend

[1] Informationen zu erreichen unter: `http://backstage.pms.ifi.lmu.de`
[2] Das Play Framework ist zu finden unter: `https://www.playframework.com`

testen können. Dieser wird nun auch für das Backend genutzt. Dabei wird dem Service eine JSON-Repräsentation der zu kompilierenden Dateien zugeschickt. Diese besteht aus den einzelnen Dateien sowie einigen weiteren Informationen. Der Service kompiliert diese und führt die angegebene Hauptdatei sofern möglich aus. Anschließend werden die Ergebnisse retourniert.

Innerhalb einer anderen studentischen Arbeit wurde das sogenannte `Prediction Framework` entworfen. Dieses ermöglicht die Integration verschiedener Prädiktoren in das Gesamtprojekt. Aus diesem Framework werden mehrere Bestandteile verwendet:

Observation Als Beobachtung werden die einzelnen Ereignisse bezeichnet, die beobachtet werden können. Außerdem schließt ein Prädiktor, ausgehend von einer oder mehrerer Beobachtungen, auf zukünftige Ereignisse. In diesem Fall sind das die einzelnen Abgaben der Studenten. Das Ergebnis entspricht der Beobachtung.

Prediction Tag Als `Prediction Tag` werden die Ereignisse zusammengefasst, auf die ein Prädiktor schließen kann. Diese können identisch mit den Beobachtungen sein, dies muss jedoch nicht zwingend sein. In diesem Fall sind sie identisch. Der Prädiktor soll ausgehend von Abgaben zukünftige Abgaben vorhersagen.

Behaviour Log Im sogenannten `Behaviour Log` werden für die einzelnen Veranstaltungen die Beobachtungen abgespeichert. In diesem Fall sind das Abgaben der Studenten.

4.2 Compiling Service

Mithilfe des Compiling Services können Code Abgaben verschiedener Programmiersprachen ausgewertet werden. Hierzu wird zunächst eine Abgabe aufbereitet und in ein JSON Objekt transferiert. Anschließend wird das Objekt an den Compile Service Coconut geschickt. Sobald von diesem das Ergebnis erhalten wurde, werden die Ergebnisse ausgewertet. Dabei werden die einzelnen Aufträge asynchron mithilfe von Scala „Futures[3]“ abgearbeitet. Dadurch können auch mehrere Abgaben gleichzeitig ausgewertet werden. Um eine korrekte Berechnung der „History“ Werte zu gewährleisten, muss die Auswertung in derselben Reihenfolge erfolgen, wie die Übungsaufgaben gestellt wurden.

4.2.1 Aufbau

Backstage stellt Veranstaltungen, wie bereits zuvor erwähnt, als Projekte dar. Dabei können Projekte beliebig viele Unterprojekte besitzen. Bei diesen handelt es sich auch wieder um Projekte. Mit Unterprojekten lassen sich zum Beispiel Übungsblätter und deren zugehörige Dokumente gruppieren.

Für ein Projekt können sogenannte „Assignments“ angelegt werden, die beispielsweise einzelnen Aufgaben eines Übungsblattes entsprechen. Für jeden studentischen Teilnehmer des Projektes werden dann automatisch Aufträge erstellt diese Aufgabe zu bearbeiten. Es gibt verschiedene Typen von Aufgaben. Für diese Arbeit sind lediglich Programmieraufgaben relevant. Bei diesen kann der abgegebene Code ausgewertet werden.

Ausgehend von der Projektstruktur von Backstage können sowohl die Abgaben von gesamten Projekten (einer gesamten Veranstaltung), einzelnen Subprojekten (eines Übungsblattes) sowie auch individuelle Aufgaben kompiliert und ausgewertet werden, sofern diese bearbeitet wurden. Um alle Aufgaben eines gesamten Projektes auszuwerten, werden sequentiell alle Subprojekte sowie deren Aufgaben ausgewertet. Dabei werden jeweils Futures erstellt, die von den jeweilig vorherigen Subprojekten sowie Aufgaben abhängen. Dadurch

[3] Die Scala Doku für Futures ist zu erreichen unter: `https://docs.scala-lang.org/overviews/core/futures.html`

ist die korrekte Abarbeitungsreihenfolge gewährleistet. Zu Beginn wird für jede gestellte Aufgabe geprüft, ob es sich bei dieser Aufgabe um eine Programmieraufgabe handelt. Ist dies nicht der Fall, wird diese ignoriert. Folgend werden die einzelnen Abgaben auf Fehler geprüft. Diese einzelnen Zustände und Fehler sind in der Enumeration `CSStatus` abgebildet.

```
1  object CSStatus extends Enumeration {
2  type CSStatus = Value
3  val NoError, WrongUnitType, NoTestFound, ParsingError, Error,
       Skipped = Value
4  }
```

Listing 4.1: Enumeration CSStatus

Zunächst wird überprüft, ob die Abgabe auch einen Inhalt enthält. Ist dies nicht der Fall, so hat der Student die Aufgabe nicht bearbeitet; `skipped`. Nachfolgend wird der Typ der Abgabe geprüft. Handelt es sich dabei nicht um eine auswertbare Codeabgabe, wird diese als `WrongUnitType` deklariert. Anschließend wird ein Auftrag zum Kompilieren erstellt.

```
1  case class CompilingServiceAssignment(var mainFileName:
       String, var files: Map[String, String], arg: String,
       language: String) {
2  def toJson: JsObject = {...}
3  }
```

Listing 4.2: Klasse CompilingServiceAssignment

Dieser enthält die Informationen, die später zum Kompilieren nötig sind. Dies ist zum einen der Name der Hauptdatei sowie eine Map mit allen abgegebenen Dateien und ihrer Namen, zum anderen gegebenenfalls Argumente, die dem Compiler übergeben werden sollen, sowie die Programmiersprache. Bei der Hauptdatei handelt es sich für gewöhnlich um die Testdatei, mit welcher die Abgaben ausgewertet werden. Kann für eine Aufgabe keine Testdatei gefunden werden, so wird der Auftrag abgebrochen und der Status `NoTestFound` gesetzt.

Konnten alle nötigen Informationen aus den verschiedenen Datenbanken extrahiert werden, wird der Auftrag in ein JSON Objekt umgewandelt und an den `Compiling Service` gesendet. Sobald dieser den Auftrag fertig bearbeitet hat, wird das Ergebnis abgefragt und daraus ein `CompilingServiceResult` erstellt.

```
1  case class CompilingServiceResult(cSStatus: CSStatus, success:
       Boolean, stdout: String, end: Int, stderr: String, start:
       Int) {
2  }
```

Listing 4.3: Klasse CompilingServiceResult

Dabei gibt der Service zurück, ob der Auftrag kompiliert werden konnte, sowie die Standardausgabe und die Standardfehlerausgabe. Der `CSStatus` wird auf `NoError` gesetzt. Dies bedeutet, dass die Abgabe eine korrekte Form hat. Anschließend wird das Resultat ausgewertet. Sofern die Abgabe kompiliert werden konnte, wird die Standardausgabe untersucht. Die Testdateien haben immer einen ähnlichen Aufbau. Sofern ein Test erfolgreich war, wird ein String „TestOk“ ausgegeben. Schlägt ein Test fehl wird der String „TestNotOk “ ausgegeben. Die Vorkommen dieser Strings werden gezählt und abschließend das Endresultat `InterpretedCompilingServiceResult` erstellt. Dies enthält wieder den `CSStatus`, ob die Kompilierung erfolgreich war und die Anzahl an durchgeführten sowie erfolgreichen Tests.

```
case class InterpretedCompilingServiceResult(cSStatus: CSStatus,
    compilationSuccessful: Boolean, testCount: Int,
    successfulTests: Int, var difficulty: Difficulty.Difficulty) {
}
```

Listing 4.4: Klasse InterpretedCompilingServiceResult

Nachdem alle Abgaben einer Aufgabe ausgewertet wurden, wird zusätzlich die Schwierigkeit dieser Aufgabe berechnet. Hierfür wird der Prozentsatz der fehlerfreien Abgaben in Bezug auf alle Abgaben gebildet. Die Einteilung in die drei Kategorien *negativ, neutral* und *positiv* erfolgt dabei wie in Kapitel 3 beschrieben in Stufen von 33 Prozent.

4.2.2 Nutzung

Zur Nutzung des `CompilingServices` kann dieser in den benötigten Controllern injiziert werden. Dabei handelt es sich um eine häufig verwendete Vorgehensweise des `Play Frameworks`. Der Service benötigt eine Datenbankverbindung sowie Zugänge zum `WS-Client`, `BackstageUnitService` und zur Konfiguration. Des Weiteren benötigt er einen `ExecutionContext`. Der `CompilingService` kann ausgehend von den einzelnen Projekten und Subprojekten genutzt werden. Dafür stellt das Backend zwei Routen zur Verfügung, über die der Vorgang initiiert werden kann. Diese sind in Tabelle 4.1 aufgeführt. Sie rufen die jeweiligen Methoden im `ProjectController` auf. In diesen werden zusätzlich noch die Ergebnisse als Beobachtungen festgehalten und in die jeweiligen `BehaviourLogs` gespeichert.

PUT	/projects/compileProject/*projectId
PUT	/projects/compileSubProject/*rootProjectId/*subProjectId

Tabelle 4.1: Compiling Service Routen

4.3 Predictor

Der `Predictor` implementiert das in Kapitel 3.2 vorgestellte Bayessche Netz. Hierdurch ist es möglich Vorhersagen auf Grundlage vergangener Abgaben für zukünftige Abgaben zu treffen. Dafür wird das Data-Mining Framework Weka[4] genutzt. Weka bietet zum einen eine Java Bibliothek, die in bestehende Projekte integriert werden kann, zum anderen statistische Tools zur Datenanalyse und Datenverarbeitung; unter anderem ein Tool um Bayessche Netze zu erstellen und abzuspeichern. Einige von ihnen werden am Ende dieses Abschnitts kurz erläutert. Zudem wird der `Predictor` in das bereits bestehende `Prediction Framework` der Plattform integriert.

4.3.1 Aufbau

Zu Beginn wird die Netzstruktur für den `Predictor` geladen. Hierfür können die von Weka erstellten XMLBIF Dateien (XML-based BayesNets Interchange Format) genutzt werden. Dabei handelt es sich um spezielle XML Dateien. Des Weiteren muss die Datenstruktur des Netzes gesetzt werden. Zu diesem Zweck bieten sich entweder Wekas ARFF Dateien (Attribute-Relation File Format) oder aber eine manuelle Erstellung an. In den ARFF Dateien sind Trainingsdaten sowie die Struktur der Daten hinterlegt. Die Verwendung der beiden

[4] Weka ist erhältlich unter `https://www.cs.waikato.ac.nz/ml/weka/`

Dateitypen wird am Ende des Kapitels behandelt. Nachfolgend können die CPTs des Netzes anhand der Werte in der ARFF Datei gesetzt werden. Somit sind bereits Trainingsdaten vorhanden und der `Predictor` kann genutzt werden.

Der `Predictor` kann dabei ausgehend von einer Menge an `Obeservations` auf einen `Prediction Tag` schließen. Dieser `Predictor` schließt von einem oder mehreren `CompetenceLeveln` auf ein zukünftiges `CompetenceLevel`.

```
case class CompetenceLevel(form_correct: TypeBool.Value,
    syntax_correct: TypeBool.Value, difficulty:
    Difficulty.Difficulty, formH: Double, syntaxH: Double,
    semanticH: Double, semantic_correct: TypeBool.Value) extends
    Tag {
}
```

Listing 4.5: Klasse CompetenceLevel

Die Datenstruktur enthält alle Variablen des Netzes. Die Historywerte sind hier als Zahl repräsentiert und werden vor der Nutzung im Netz noch in nominale Attribute umgewandelt. Bei den Typen der restlichen Variablen handelt es sich um `Enums`, die die Ausprägungen der einzelnen Variablen abbilden. Als Beispiel ist das `Enum Difficulty` angegeben. `enumReads` und `enumWrites` werden dabei für das `Prediction Framework` benötigt.

```
object Difficulty extends Enumeration {
   type Difficulty = Value
   val Low, Medium, High = Value
   implicit val enumReads: Reads[tags.Difficulty.Value] =
      Reads.enumNameReads(Difficulty)
   implicit val enumWrites = Writes.enumNameWrites
}
```

Listing 4.6: Enumeration Difficulty

Die Struktur der Daten kann auch ohne Datei definiert werden. Dafür bietet Weka die `Instances` Klasse an. Zunächst ist es nötig, alle `Attribute` (Variablen) zu definieren. Hierfür wird der Name des Attributes sowie dessen Ausprägung benötigt. Es sind Beispiele für `form_correct` und `semantic_correct` angegeben. Die Definition der restlichen Attribute funktioniert analog. Da es sich bei Weka um ein Java Framework handelt, müssen Java Typen wie beispielsweise Java Listen übergeben werden. Das Endresultat wird in Weka gesondert behandelt und als `ClassAttribute` bezeichnet. Es sollte immer das letzte Attribut sein.

```
val fvNominalValBool: util.List[String] =
    TypeBool.values.toList.map(t => t.toString).asJava
...
val names: Array[String] = implicitly[Manifest[CompetenceLevel]]
                    .runtimeClass.getDeclaredFields.map(_.getName)
val Attribute1 = new Attribute(names(0), fvNominalValBool)
...
val ClassAttribute = new Attribute(names(6), fvNominalValBool)

val attributes = new util.ArrayList[Attribute](Array(Attribute1,
    Attribute2, Attribute3, Attribute1H, Attribute2H, Attribute3H,
    ClassAttribute).toList.asJava)
```

```
10 val instances = new Instances("Rel", CompetenceLevel.attributes,
       0)
11 instances.setClassIndex(instances.numAttributes() - 1)
```

Listing 4.7: Erstellung der Struktur für den Datensatz

Zur Nutzung im Netz muss ein `CompetenceLevel` final in eine `Instance` umgewandelt werden. Dafür wird zunächst eine leere `Instance` erzeugt und ihre Struktur mithilfe von `instances` gesetzt. Nachfolgend werden die Werte übertragen. Da es sich bei nominalen Attributen um `Strings` handelt, bieten sich die zuvor verwendeten `Enums` an.

Zur Vorhersage von zukünftigen Übungen kann nun eine Menge an Übungen übergeben werden. Liegen ausreichend viele Trainingsfälle vor, so wird das Netz mit den neuen Daten trainiert. Alternativ werden die in der ARFF Datei hinterlegten Daten genutzt. Die Mindestmenge von vorliegenden Daten ist konfigurierbar und beträgt standardmäßig 100 Fälle. Zum Setzen der Evidenz werden, wie zum Trainieren, Variablen vom Typ `Instance` genutzt. Deren Struktur wird gesetzt und anschließend die gewünschte Evidenz für die verschiedenen Attribute eingetragen. Nachfolgend werden die Wahrscheinlichkeiten für das Auftreten der einzelnen Fälle für das gewünschte Attribut, in der Regel das Klassenattribut, berechnet und zurückgegeben.

4.3.2 Nutzung

Der Predictor kann zum einen über das `Prediction Framework` genutzt werden und zum anderen über den Aufruf folgender Methode:

```
1 def predict(instance: Instance, cL: CompetenceLevel):
      Prediction[CompetenceLevel] = {
2 }
```

Listing 4.8: Methode zum Predicten

In diesem Fall kann die Evidenz für die einzelnen Variablen beliebig gesetzt werden. Über das `Prediction Framework` werden nur die Historywerte gesetzt.

4.3.3 Weka

Weka bietet neben der für den Predictor genutzten Java Library auch verschiedene GUI Werkzeuge an. Die beiden wichtigsten für dieses Projekt sind dabei der „Explorer“ sowie der „Bayes Net Editor“. Daneben stellt Weka im Allgemeinen Implementierungen von verschiedenen Lernalgorithmen sowie auch eine Vielzahl von Werkzeugen zum Transformieren von Datensätzen, unter anderem zur Diskretisierung und Abtastung, zur Verfügung. Im Folgenden wird kurz der Arbeitsablauf zum Erstellen eines Bayesschen Netzes mithilfe von Weka erläutert.

Da es sich bei Weka um ein Java Programm handelt, wird Java zur Ausführung von Weka benötigt. Nach dem Start von Weka öffnet sich ein Menü, aus welchem die einzelnen Applikationen von Weka ausgewählt werden können. Das erste für dieses Projekt benötigte Werkzeug, der Explorer, befindet sich bei der genutzten Version 3.8.2 direkt rechts unter Applications. Der Bayes Net Editor befindet sich stattdessen unter Tools.

Explorer Nach dem Öffnen des Weka Explorers kann nun eine Datendatei ausgewählt werden, wie zum Beispiel eine CSV Datei. Anschließend wird eine Übersicht der Daten erstellt. Unter anderem werden die einzelnen Attribute, deren Typ sowie ihre Ausprägungen angezeigt und die Vorkommen dieser gezählt. Dies ist für den in dieser Arbeit verwendeten Datensatz in Abbildung 4.1 abgebildet.

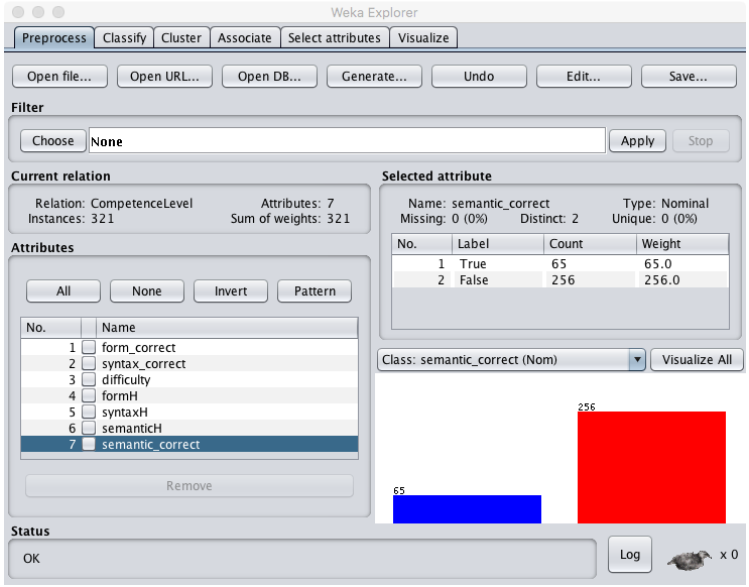


Abbildung 4.1: Weka Explorer mit geladenem Datensatz

Bei der Arbeit mit Weka ist aber nach Möglichkeit ein anderes Dateiformat als das ARFF Format zu bevorzugen. Dabei handelt es sich ebenfalls um ein durch Kommas getrenntes Format. Allerdings besitzt es im Gegensatz zu einer CSV Datei noch einen Header mit Informationen zu den Daten. Dies ist einerseits der Name des Datensatzes und andererseits alle Ausprägungen aller Attribute in folgender Form:

```
@attribute difficulty {Low,Medium,High}
```

Wie in Kapitel 3.2 erwähnt wurde keine Aufgabe mit einer niedrigen Schwierigkeit eingestuft. Ausgehend von einer CSV Datei würde diese Ausprägung wegfallen, da sie nicht vorhanden ist. Weka kann eine CSV Datei in eine ARFF Datei umwandeln. In diesem Fall wäre dennoch eine Nachbearbeitung von Hand nötig, um die fehlende Ausprägung nachzutragen. Hierzu ist lediglich die Ausprägung zu ergänzen. Es ist wichtig, dass die Reihenfolge der Ausprägung in allen verwendeten Dateien sowie auch im Code identisch ist. Fehlt ein Wert bei einer Instanz, wird dies durch ein Fragezeichen markiert.

Extrahiert man die Daten aus einem Programm, wie bei dieser Arbeit, ist es möglich, dies mit der Library direkt als ARFF Datei zu erledigen.

Neben der Übersicht über den Datensatz können im Explorer direkt auch verschiedene Lernalgorithmen angewendet und evaluiert werden. Zur Erstellung des Bayesschen Netzes wird lediglich die ARFF Datei benötigt.

Bayes Net Editor Nach dem Öffnen kann die zuvor erstellte ARFF Datei geladen werden. Es werden nun für alle Attribute Knoten erzeugt. Sofern die Namen zu lang sind, werden diese durch Nummern ersetzt. Dabei entspricht die Zahl der Position des Attributs in der Datei. Daraufhin können für jeden Knoten die Eltern zugewiesen werden. Dies erfolgt über einen Rechtsklick auf den Knoten. Um ein ansprechenderes Layout des Netzes zu erhalten, kann der Button „Layout Graph“ genutzt werden. Die Knoten können auch von Hand verschoben werden, allerdings ist dies mitunter recht arbeitsaufwendig. Es sollte darauf geachtet werden, dass ein Knoten ausgewählt wurde, bevor er verschoben wird. Die korrekte Auswahl kann an den 4 gruppierten Punkten um ihn herum erkannt werden. Zum anderen sollte ein Knoten nicht durch das Netz durchgezogen werden, ansonsten wird das

Layout des Netzes teilweise zurückgesetzt.

Sind alle Eltern-Kind Beziehungen eingetragen, müssen als letztes noch die CPTs gesetzt werden. Diese können entweder manuell eingetragen oder basierend auf den Daten in der ARFF Datei generiert werden. Hierzu muss unter Tools Learn CPT ausgewählt werden. Ohne CPTs kann das Netz nicht in der benötigten BIFXML Datei abgespeichert werden. Unter Tools können außerdem die Margins der einzelnen Knoten angezeigt werden. Somit können bereits in diesem Tool verschiedene Evidenzen gesetzt und deren Auswirkungen beobachtet werden.

Die erstellte BIF Datei kann nun zum Beispiel im Explorer verwendet werden um einen Bayes Net Klassifizierer zu nutzen, oder aber um die hinterlegte Version für Backstage durch eine erweiterte Version zu ersetzen.

4.4 Grouping

Durch den Group Building Service lassen sich, wie in Kapitel 3.3 beschrieben, verschiedene Gruppenformen bilden. Es ist möglich sowohl homogene sowie heterogene Gruppen zu bilden. Durch eine manuelle Aufteilung und Kombination von beiden sind auch gemischte Ansätze möglich. Dies könnte verwendet werden, um ausschließlich gezielt Studenten mit bestimmten Eigenschaften heterogen einzuteilen. Zudem lassen sich Peer Review Zuteilungen bilden. Die zugrunde liegenden Methoden sind generisch geschrieben, somit können durch eine Ordnungsfunktion beliebige Objekte zur Gruppenbildung genutzt werden. Sollen mehrere Ordnungsfunktionen mit unterschiedlichen Attributen genutzt werden, können diese in eine einzelne Funktion zusammengeführt, sowie je nach Bedarf, gewichtet werden.

4.4.1 Aufbau

Für die Bildung der Lerngruppen können verschiedene Parameter angegeben werden. Erstens ob die Bildung homogen oder heterogen erfolgen soll. Zweitens wie viele Mitglieder die Gruppen enthalten sollen und drittens ob für jede Gruppe direkt optional ein eigenes Subprojekt angelegt werden soll. Für die Peer Review Bildung gibt es keine konfigurierbaren Parameter.

Anschließend wird die jeweilige Gruppenbildung durchgeführt. Für die heterogene Gruppenbildung wird dabei zwingend eine Funktion benötigt, die die Studenten auf eine Zahl abbildet. Im Vergleich dazu reicht der homogenen Bildung auch ein `Ordering` ebenso wie der Peer Review Bildung. Die Gruppenbildung erfolgt dabei wie in Kapitel 3.3 beschrieben. Die Bildung von homogenen Gruppen sowie die Bildung der Peer Review Zuteilungen ist dabei relativ einfach, wohingegen die Bildung der heterogenen Gruppen vergleichsweise aufwendig ist. Abschließend werden jeweils Sequenzen von Sequenzen von beispielsweise Studenten erstellt und zurückgegeben; Liste aller Gruppen.

4.4.2 Nutzung

Zur Nutzung des Group Building Services ist eine Datenbankverbindung nötig. Anhand dieser werden, falls gewünscht, die Subprojekte für die einzelnen Gruppen direkt erstellt. Die Erstellung der Peer Review Zuteilungen ist bereits in das Modul `Orchestration` der Backstage Plattform integriert. Zur Bildung von homogenen und heterogenen Gruppen kann die Methode `buildGroup` genutzt werden.

```
1 buildGroup(subs:Boolean,groupSize: Int, hetero: Boolean,
    projectId: String,seq:Seq[User],function: User => Int):
    Seq[Seq[User]]{
```

```
2 | ...
3 | }
```

Listing 4.9: Methode zum Bilden von Gruppen

Neben den bereits angesprochenen Parametern zur Erstellung von Subprojekten, der Gruppengröße, die Art der Bildung sowie der Ordnungsfunktion, werden zudem noch die Projekt Id sowie die Liste der zu verteilenden Studenten benötigt.

KAPITEL 5

Evaluierung

Abschließend wird das verwendete Modell für die Klassifikation in diesem Kapitel evaluiert. Dafür werden zunächst Methoden und Metriken vorgestellt, die für die Evaluierung genutzt werden. Im Anschluss daran werden verschiedene Szenarien für den Klassifikator betrachtet und seine Performance untersucht. Diese wird abschließend analysiert und bewertet.

5.1 Methodik

Nachfolgend werden unterschiedliche Verfahren, die bei der Auswertung des Schätzers genutzt werden, kurz vorgestellt. Dies sind erstens Metriken, die auf den Werten einer Konfusionsmatrix resultieren, zweitens eine grafische Auswertung mit ROC Kurven sowie drittens ein Verfahren zur Einteilung der Daten in Test und Trainingsdaten.

5.1.1 Metriken

Um die Güte eines Schätzers bewerten zu können benötigt es verschiedene Metriken, welche die unterschiedlichen Gütekriterien eines Klassifikators berücksichtigen. Dabei ist zu bedenken, dass ein Klassifikator oder exemplarisch auch ein medizinischer Test nicht für alle Kriterien optimiert werden kann, da diese sich untereinander beeinflussen. Ein medizinischer Test, ein Spezialfall eines Klassifikators, kann entweder darauf optimiert werden möglichst keinen Kranken zu übersehen oder keinen Gesunden als krank zu bezeichnen, aber in der Regel ist es nicht möglich beides mit dem selben Test zu erreichen. Die folgenden Definitionen basieren auf Powers [13] und Fawcett [5].

Eine der einfachsten Metriken zur Einschätzung eines Klassifikators sind die Anteile der richtig sowie falsch klassifizierten Fälle. Diese berechnen sich wie in den Formeln 5.1 und 5.2 gezeigt.

$$\text{Anteil korrekt} = \frac{\text{Anzahl richtiger Vorhersagen}}{\text{Anzahl aller Vorhersagen}} \tag{5.1}$$

$$\begin{aligned}\text{Anteil inkorrekt} &= \frac{\text{Anzahl falscher Vorhersagen}}{\text{Anzahl aller Vorhersagen}} \\ &= 1 \text{ - Anteil korrekt}\end{aligned} \tag{5.2}$$

Bei einer binären Einteilung kann, wie in der Konfusionsmatrix in Abbildung 5.1 abgebildet ist, einer von vier Fällen auftreten. Die Vorhersage ist richtig positiv, falsch negativ, falsch positiv oder richtig negativ. Für jeden Fall werden alle Auftreten gezählt. Durch das Addieren der grünen Zellen in der Abbildung, den korrekt vorhergesagten Fällen, kann die Anzahl aller korrekt vorhergesagten Fälle bestimmt werden. Die zusammengehörigen Fälle, die richtig oder falsch vorhergesagt werden, liegen jeweils auf einer der Diagonalen. Zur Bestimmung beispielsweise aller positiven Resultate, werden die richtig positiven sowie falsch negativen Fälle zusammengezählt. Dies funktioniert für die restlichen Werte analog. Die Berechnungen folgen dem Prinzip der Vierfeldertafel.

		Resultat	
	Gesamt	Resultat positiv	Resultat negativ
Vorhersage	Vorhersage positiv	Richtig positiv	Falsch positiv
	Vorhersage negativ	Falsch negativ	Richtig negativ

Abbildung 5.1: Einfache Konfusionsmatrix

Richtig positiv Das Resultat und die Vorhersage sind positiv, die Vorhersage des Klassifikators war richtig. Im Englischen wird dies als true postive bezeichnet; abgekürzt TP.

Falsch negativ Die Vorhersage war negativ, das Resultat aber positiv. Die Vorhersage des Klassifikators war falsch, Fehler 1. Art. Im Englischen wird dies als false negative bezeichnet; abgekürzt FN.

Falsch positiv Die Vorhersage war positiv, das Resultat aber negativ. Die Vorhersage des Klassifikators war falsch, Fehler 2. Art. Im Englischen wird dies als false postive bezeichnet; abgekürzt FP.

Richtig negativ Das Resultat und die Vorhersage sind negativ, die Vorhersage des Klassifikators war richtig. Im Englischen wird dies als true negative bezeichnet; abgekürzt TN.

Anhand der vier Werte lässt sich bereits ein erster Überblick über die Klassifizierung gewinnen. Zur Einschätzung eines Klassifikators gibt es noch verschiedene Maße, die sich anhand der Werte aus der Konfusionsmatrix berechnen lassen. Die Zusammenhänge sind in der erweiterten Konfusionsmatrix in Abbildung 5.2 zu sehen.

Trefferquote / True positive rate (TPR) Die Trefferquote bezeichnet den Anteil von wirklich positiven Fällen, die korrekt vorhergesagt werden. Der Anteil von tatsächlich Kranken, der erkannt wurde. Sie wird auch Empfindlichkeit, Sensitivität oder recall genannt.

Missrate / False negative rate (FNR) Die Missrate ist der Anteil von wirklich positiven Fällen, die fälschlicherweise als negativ eingestuft werden. Der Anteil von tatsächlich Kranken, der nicht erkannt wurde.

Spezifität / True negative rate (TNR) Die Spezifität bezeichnet den Anteil von wirklich negativen Fällen, die korrekt vorhergesagt werden. Der Anteil an tatsächlich Gesunden der als gesund erkannt wurde. Sie wird auch kennzeichnende Eigenschaft oder inverse recall genannt.

Ausfallrate / False positive rate (FPR) Die Ausfallrate ist der Anteil von wirklich negativen, die fälschlicherweise als positiv klassifiziert werden. Der Anteil an tatsächlich Gesunden, der als krank erkannt wurde. Sie wird im englischen auch false alarm rate oder fallout genannt.

Genauigkeit / Accuracy (ACC) Die Genauigkeit gibt den Anteil aller korrekt klassifizierter Fälle an; auch Korrektklassifikationsrate genannt. In Bezug auf das Medizinbeispiel entspricht die Genauigkeit dem Anteil an richtig positiven und negativen Tests in Bezug auf alle durchgeführten Tests. Das Gegenteil, der Anteil an falsch klassifizierten Fällen, wird als Falschklassifikationsrate bezeichnet, beide addieren sich zusammen auf 100 Prozent.

		Resultat			
	Gesamt	positiv	negativ	$PRE = \frac{TP + FN}{Gesamt}$	$ACC = \frac{TP + TN}{Gesamt}$
Vorhersage	positiv	TP	FP	$PPV = \frac{TP}{TP + FP}$	$FDR = \frac{FP}{TP + FP}$
	negativ	FN	TN	$FOR = \frac{FN}{FN + TN}$	$NPV = \frac{TN}{FN + TN}$
		$TPR = \frac{TP}{TP + FN}$	$FPR = \frac{FP}{FP + TN}$		
		$FNR = \frac{FN}{FN + TP}$	$TNR = \frac{TN}{TN + FP}$		

Abbildung 5.2: Erweiterte Konfusionsmatrix

Prävalenz / Prevelanz Die Prävalenz gibt den Anteil an positiven Resultaten in Bezug auf alle durchgeführten Tests an. Der Anteil an Kranken von allen getesteten Personen. In Abbildung 5.2 abgekürzt durch PRE.

Relevanz / Positive Predictive Value (PPV) Die Relevanz bezeichnet den Anteil der vorhergesagten positiven Fälle, die richtig positive Ergebnisse sind. Der Anteil mit positivem Testergebnis, der auch wirklich krank ist. Sie wird auch Wirksamkeit oder precision genannt.

Falscherkennungsrate / False Discovery Rate (FDR) Die Falscherkennungsrate ist das Komplement zur Relevanz. $1 = PPV + FDR$

Segreganz / Negative predictive Value (NPV) Die Segreganz bezeichnet den Anteil der vorhergesagten negativen Fälle, die richtig negativ Ergebnisse sind. Der Anteil mit negativem Testergebnis, der auch wirklich gesund ist. Sie wird auch Trennfähigkeit genannt.

False omission rate (FOR) Die false omission rate ist das Komplement zur Segreganz. $1 = NPV + FOR$

F1-Maß / F1-score Das F1-Maß kombiniert Relevanz und Trefferquote. Es stellt das gewichtete harmonische Mittel der beiden dar. Die Berechnung erfolgt analog der Formel 5.3. Daneben gibt es allgemein noch das F_α-Maß, welches eines der beiden Bestandteile stärker gewichtet.

$$\frac{2}{\frac{1}{TPR} + \frac{1}{PPV}} \tag{5.3}$$

MCC (Matthews correlation coefficient) Das MCC-Maß berücksichtigt sowohl wahre als auch falsche positive und negative Fälle. Die Formel ist unter 5.4 angegeben. Im Allgemeinen wird es als ausgewogenes Maß betrachtet, welches auch bei verschieden großen Klassen verwendet werden kann. Der resultierende Wert liegt zwischen -1 und 1. Der Wert 1 bedeutet dabei, dass eine perfekte Einteilung vorliegt. Ein Wert von -1 bedeutet das Gegenteil. Eine zufällige Verteilung würde den Wert 0 erzeugen.

$$\frac{TP * TN - FP * FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{5.4}$$

5.1.2 Kreuzvalidierung

Bei einer ausreichend großen Datenmenge wäre es möglich, eine Testmenge zur Validierung des Klassifikators abzutrennen. Häufig liegen jedoch nicht ausreichend große Datenmengen vor. Dadurch kann es vorkommen, dass nicht alle Klassen ausreichend in der Datenmenge repräsentiert sind. Somit können mögliche Ergebnisse eine hohe Varianz aufweisen. In diesem Fall lässt sich beispielsweise die Kreuzvalidierung nutzen. Hierbei wird ein Teil der Daten für das Training benutzt und ein anderer Teil zum Testen. Anschließend wird ein anderer Teil zum Training und zum Testen genutzt usw.

Häufig wird bei der Kreuzvalidierung das sogenannte k-fach Verfahren angewendet. Bei diesem werden die Daten in K möglichst gleich große Teile geteilt. Nun werden K-1 Teile zum Training genutzt und der letzte Teil zum Testen. Ein Beispiel für eine Aufteilung ist für K = 5 in Abbildung 5.3 gegeben. Somit werden 5 mal 80 Prozent der Daten zum Training des Klassifikators genutzt und die verbleibenden 20 Prozent als Testdaten.

1	2	3	4	5
Training	Test	Training	Training	Training

Abbildung 5.3: Beispiel für 5-fach Kreuzvalidierung nach Hastie et al. [6, S. 242]

Die Ergebnisse der einzelnen Durchläufe werden für das Endresultat zusammengeführt, indem deren Durchschnitt gebildet wird. Um möglichst gute Ergebnisse zu erzielen, sollten die einzelnen Teile möglichst gleichmäßig, mit einer geringen Varianz bezüglich der Klassen, verteilt sein. Dies wird dann als k-fache stratifizierte Kreuzvalidierung bezeichnet. Dadurch wird die Varianz der Abschätzung verringert, vgl. Hastie et al. [6, S. 241ff.]

5.1.3 Receiver Operating Characteristic (ROC) Kurve

Receiver Operating Characteristic ist ein Begriff, der bei der Signalerfassung verwendet wird, um den Kompromiss zwischen Trefferquote und Ausfallrate über einem verrauschten Kanal

zu charakterisieren. Eine ROC-Kurve wertet die Leistung eines Klassifikators basierend auf TP und FP aus. Die vertikale Achse reflektiert die Trefferquote (TPR) und die horizontale Achse die Ausfallrate (FPR). Die einzelnen Punkte der Kurve entsprechen den Werten des Klassifikators bei einem gewissen „Threshold“. Dieser wird von Weka passend bestimmt. Der jeweilige Bereich ist unter den Kurven angegeben.

Der Nutzen eines ROC Diagramms wie in Abbildung 5.4 liegt darin, dass das bevorzugte Ergebnis eines Klassifikators darin besteht, eine niedrige FPR und eine hohe TPR zu haben. Beginnend von links auf der FPR-Achse, nähert sich der TPR-Wert eines guten Klassifizierers nahe 1 auf dem Weg zur rechten Seite der Achse, mit nur einer kleinen Änderung in der FPR. Je näher sich die ROC-Kurve entlang der vertikalen Achse befindet und sich in der Nähe des Punkts (0,1) der oberen linken Achse des Diagramms nähert, desto besser ist der Klassifizierer.

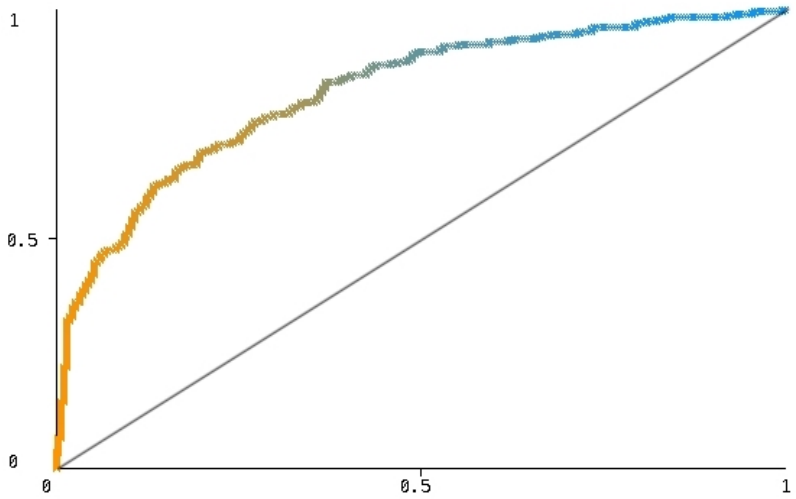


Abbildung 5.4: Beispiel für eine ROC Kurve
Threshold: 0-1

Eine nützliche Metrik zur Bewertung des Klassifizierers ist die Fläche unter der ROC-Kurve, dies wird als „Area Under Curve“ kurz AUC bezeichnet. Bei der Untersuchung der Achsen ist ersichtlich, dass das theoretische Maximum für die Fläche 1 beträgt. Ein Wert von 0.5 entspricht einem zufälligen Raten. Dies entspräche der Diagonalen. Ein Wert von 0 würde bedeuten, dass alle Fälle falsch eingeschätzt wurden, vgl [15, S. 186f. und 226ff.]. Der AUC Wert der Kurve in Abbildung 5.4 beträgt 0.82.

5.2 Resultate

Nachfolgend werden verschiedene Auswertungen für diverse Szenarien des Schätzers präsentiert. Für die Auswertungen wurde eine 10-fach Kreuzvalidierung mit Hilfe von Weka durchgeführt; diese Option befindet sich unter anderem im Explorer von Weka. Sofern möglich führt Weka dabei eine stratifizierte Kreuzvalidierung aus.

Für jede Auswertung ist eine Tabelle angegeben. Im oberen linken Bereich befindet sich die Konfusionsmatrix aus der das Auftreten der einzelnen Fälle entnommen werden kann. Der Aufbau entspricht der einfachen Konfusionsmatrix in Abbildung 5.1.

Rechts neben der Konfusionsmatrix sind sowohl die Anzahl an korrekten sowie inkorrekten Vorhersagen absolut und prozentual angegeben. Diese berechnen sich wie in

den Formeln 5.1 und 5.2 dargestellt. Zudem können die absoluten Werte auch aus den Diagonalen der Konfusionsmatrix ausgelesen werden.

Unter den beiden Tabellenteilen befindet sich eine weitere Tabelle mit den in Kapitel 5.1.1 beschriebenen Maßen, die sich aus den Werten der Konfusionsmatrix berechnen lassen; siehe Abbildung 5.2. Außerdem ist die ROC-Kurve für den positiven Fall in einem Graph abgebildet. Diese stammt ebenfalls aus Weka.

Zu Beginn wird der Schätzer mit dem vollen Datensatz betrachtet. Als Klasse (vorhergesagte Variable) wird *semantisch korrekt* gewählt und eine Auswertung durchgeführt.

252	66	182
70	66	4
178	0	178

korrekte Vorhersagen	244	0,968
inkorrekte Vorhersagen	4	0,015

Trefferquote (TPR)	1	Missrate (FNR)	0
Ausfallrate (FPR)	0,022	Spezifität (TNR)	0,978
Prävalenz	0,262	Genauigkeit (ACC)	0,968
Relevanz (PPV)	0,943	Falscherkennungsrate (FDR)	0,057
Segreganz (NPV)	1	False omission rate (FOR)	0
F1-Maß	0,971	MCC	0,960

Tabelle 5.1: Auswertung für semantisch korrekt mit vollem Datensatz

Auffällig ist das sehr gute Ergebnis des Schätzers mit fast 97 Prozent korrekten Vorhersagen. Bedingt durch das Design des Netzes mit der Verwendung der Knowledge Space Theory und den verwendeten Daten ist dies nicht verwunderlich und war folglich zu erwarten. Der verwendete Datensatz enthält unbearbeitet alle Werte zu allen Knoten des Netzes. Somit sind bereits zwei der drei Stufen der Wissensstruktur bekannt; siehe Abbildung 3.4. Dadurch muss der Schätzer lediglich entscheiden ob eine Abgabe die *syntaktisch korrekt* war auch semantisch korrekt ist. Sofern die Abgabe nicht syntaktisch korrekt war, kann sie nach Definition auch nicht semantisch korrekt sein.

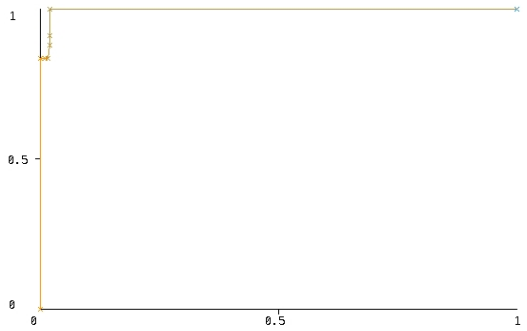


Abbildung 5.5: ROC Kurve mit ganzem Datensatz
AUC = 0.987
Threshold: 0-1

In Tabelle 5.1 sind die einzelnen Werte für die Auswertung angegeben. Eine weitere Auffälligkeit ist in der Konfusionsmatrix zu finden. Deren Werte addieren sich nicht auf den Gesamtwert auf. Dies liegt daran, dass der Schätzer 4 Fälle nicht zuordnen konnte. In

Abbildung 5.5 ist zudem die ROC Kurve für die Auswertung abgebildet. Die Fläche unter der Kurve beträgt 0,987 und ist ähnlich wie die restlichen Werte nahe am Maximum.

Dieses Einsatzszenario, die Vorhersage auf Grundlage von *syntaktisch korrekt*, wird vermutlich selten verwendet, soll jedoch als Beispiel für die Flexibilität des Netzes dienen. Nachfolgend werden nun drei realistischere Einsatzszenarien betrachtet. In diesen werden die Vorhersagen lediglich auf Basis der „Historywerte“ und der erwarteten Schwierigkeit einer Aufgabe getroffen, folglich eine Vorhersage für eine zukünftige Abgabe.

5.2.1 Vorhersage von „semantisch korrekt“

Die Hauptaufgabe für den Schätzer ist die Vorhersage ob eine zukünftig Abgabe eines Studenten *semantisch korrekt* ist oder nicht. Dies soll anhand seiner bisherigen Leistungen und anderen Parametern wie der Schwierigkeit der Aufgabe bestimmt werden. Ebenfalls wie bei der vorherigen Auswertung wird als Klasse wieder *semantisch korrekt* gewählt. Dieses mal werden jedoch die Daten angepasst um das gewünschte Szenario zu erzeugen. Hierfür werden die Werte der Variablen für *Form korrekt* und *syntaktisch korrekt* auf unbekannt gesetzt.[1] Die Daten der Auswertung sind in Tabelle 5.2 angegeben.

252	66	186
66	39	27
186	27	159

korrekte Vorhersagen	198	0,786
inkorrekte Vorhersagen	54	0,214

Trefferquote (TPR)	0,591	Missrate (FNR)	0,409
Ausfallrate (FPR)	0,145	Spezifität (TNR)	0,855
Prävalenz	0,262	Genauigkeit (ACC)	0,786
Relevanz (PPV)	0,591	Falscherkennungsrate (FDR)	0,409
Segreganz (NPV)	0,855	False omission rate (FOR)	0,145
F1-Maß	0,591	MCC	0,446

Tabelle 5.2: Vorhersage von semantisch korrekt

In diesem Szenario erreicht der Schätzer knapp 79 Prozent richtige Vorhersagen. Auffällig ist, dass die Trefferquote deutlich schlechter ist als die Spezifität. Der Schätzer erkennt demzufolge negative Resultate besser als positive. Einschränkend ist anzumerken, dass deutlich mehr negative als positive Fälle vorhanden sind. Lediglich 26 Prozent der Fälle sind positiv. Dem folgend wird auch ein niedriger Relevanz Wert von knapp 60 Prozent erreicht. Negative Fälle werden, wie bereits erwähnt, besser erkannt. Die Segreganz beträgt etwa 85 Prozent.

Das F1-Maß beträgt knapp 0,6 auf einer Skala von 0-1. Der MCC liegt im positiven Bereich bei etwa 0,45. Die ROC Kurve ist in Abbildung 5.6 abgebildet. Die Fläche unter der Kurve beträgt 0,782.

5.2.2 Vorhersage von „Form korrekt“

Neben der Vorhersage ob eine Abgabe *semantisch korrekt* ist, können auch, wie bereits beschrieben, Vorhersagen für die anderen Variablen getätigt werden. Dabei bieten sich zum Beispiel die zwei weiteren Bestandteile der Wissensstruktur an.

[1] Dies wird in einer ARFF Datei durch ein „?“ repräsentiert.

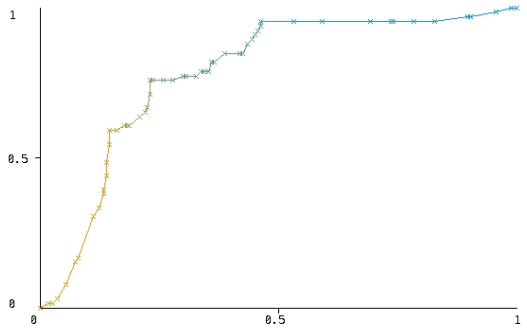


Abbildung 5.6: ROC Kurve für Klasse semantisch korrekt
AUC = 0,782
Threshold: 0,02-0,64

Die erste Hürde, die ein Student bewältigen muss, ist es die Abgabe *Form korrekt* abzugeben. Die Auswertung für dieses Szenario ist in Tabelle 5.3 zu finden. Hier ist nun *Form korrekt* als Klasse gewählt. Die Daten müssen wieder etwas angepasst werden. Die Werte für *syntaktisch korrekt* und *semantisch korrekt* werden auf unbekannt gesetzt.

252	227	25
236	222	14
16	5	11

korrekte Vorhersagen	233	0,925
inkorrekte Vorhersagen	19	0,075

Trefferquote (TPR)	0,978	Missrate (FNR)	0,022
Ausfallrate (FPR)	0,560	Spezifität (TNR)	0,440
Prävalenz	0,901	Genauigkeit (ACC)	0,925
Relevanz (PPV)	0,941	Falscherkennungsrate (FDR)	0,059
Segreganz (NPV)	0,688	False omission rate (FOR)	0,313
F1-Maß	0,959	MCC	0,512

Tabelle 5.3: Vorhersage von Form korrekt

Im Vergleich zu den beiden vorangegangenen Auswertungen gibt es dieses mal deutlich mehr positive Resultate. Die Prävalenz beträgt 90 Prozent. Insgesamt konnten fast 93 Prozent aller Fälle korrekt vorhergesagt werden. Die Trefferquote beträgt hohe 98 Prozent während die Spezifität nur 44 Prozent beträgt. Für dieses Szenario gelingt die Vorhersage von positiven Resultaten besser als von negativen.

Das F1-Maß liegt nahe am Maximalwert mit 0,96. Der Matthews Correlation Coefficient liegt dagegen auf einem ähnlichen Niveau wie bei der Vorhersage für *semantisch korrekt*, da er neben den korrekt klassifizierten Fällen auch die falschen Fälle stärker mit einbezieht.

In Abbildung 5.7 ist wieder die ROC Kurve für die Auswertung angegeben. Auffällig hierbei ist das schwache Abschneiden links unten, während rechts oben fast perfekt ist. Die Fläche unterhalb der Kurve beträgt 0,76. Während auf den ersten Blick der Schätzer für *Form korrekt* sehr performant wirkt. Bei genauerem betrachten sind Schwächen bei der Vorhersage von negativen Fällen ersichtlich. Dies ist auch bei den Werten wie dem MCC oder der AUC zu erkennen.

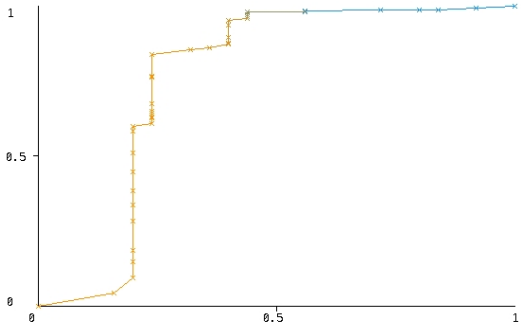


Abbildung 5.7: ROC Kurve Klasse Form korrekt
AUC = 0,757
Threshold: 0,017-1

5.2.3 Vorhersage von „syntaktisch korrekt“

Eine Stufe höher in der Wissensstruktur steht die syntaktische Korrektheit. Auch für dieses Szenario wurde eine Auswertung durchgeführt. In diesem Fall werden die Werte für die Variablen *Form korrekt* und *semantisch korrekt* auf unbekannt gesetzt. Die Ergebnisse der Auswertung sind in Tabelle 5.4 aufgelistet. Hierbei gibt es wieder eine deutlich geringere Anzahl an positiven Resultaten im Gegensatz zu der Auswertung von *Form korrekt*. Die Prävalenz beträgt etwa 28 Prozent.

Für dieses Szenario wurden 77 Prozent aller Fälle richtig vorhergesagt. Trefferquote und Segreganz sind beide auf einem ähnlich hohen Wert zwischen 75 und 80 Prozent. Dadurch ist ersichtlich, dass der Schätzer gleichmäßigere Resultate sowohl für positive wie auch für negative Fälle liefert. Lediglich die Relevanz schneidet mit einem Wert von 57 Prozent schlechter ab. Die Segreganz beträgt sogar hohe 90 Prozent.

252	70	182
97	55	42
155	15	140

korrekte Vorhersagen	195	0,774
inkorrekte Vorhersagen	57	0,226

Trefferquote (TPR)	0,786	Missrate (FNR)	0,214
Ausfallrate (FPR)	0,231	Spezifität (TNR)	0,769
Prävalenz	0,278	Genauigkeit (ACC)	0,774
Relevanz (PPV)	0,567	Falscherkennungsrate (FDR)	0,433
Segreganz (NPV)	0,903	False omission rate (FOR)	0,097
F1-Maß	0,659	MCC	0,511

Tabelle 5.4: Vorhersage von syntax

Das F1-Maß und der MCC Wert liegen auf einem ähnlichen Niveau wie bei der Vorhersage für *semantisch korrekt*. Ersteres hat einen Wert von 0,66, zweiteres einen Wert von 0,51. In Abbildung 5.8 ist die ROC Kurve für die Auswertung eingetragen. Die Fläche unterhalb der Kurve beträgt 0,79.

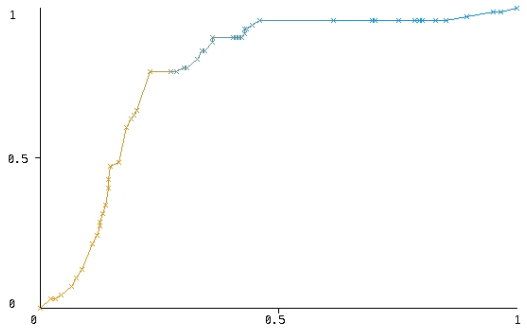


Abbildung 5.8: ROC Kurve für Klasse syntaktisch korrekt
AUC = 0,787
Threshold: 0,022-0,65

5.2.4 Vorhersage von „semantisch korrekt“ ohne difficulty Knoten

In diesem Szenario wird das Netz des Schätzers verändert. Es entspricht dem Netz in Abbildung 3.6. In diesem fehlt der *difficulty* Knoten. Es wird wieder *semantisch korrekt* vorhergesagt. Durch diese Auswertung kann der Unterschied der Performance mit und ohne des Knotens verglichen werden. In Tabelle 5.5 sind die Ergebnisse der Auswertung eingetragen. In der Konfusionsmatrix ist ein vergleichsweise hoher False Negative Wert zu erkennen. Zudem wurden von 66 positiven Resultaten lediglich 16 als solche erkannt. 71 Prozent aller Vorhersagen waren korrekt. Wie bereits bei der Auswertung mit *difficulty* Knoten beträgt die Prävalenz 26 Prozent.

Aufgrund der niedrigen erkannten positiven Resultate ist die Trefferquote mit lediglich 24 Prozent sehr niedrig. Die Spezifität liegt bei 88 Prozent. Auch die Relevanz erreicht nur einen niedrigen Wert von 42 Prozent. Die Segreganz erreicht 77 Prozent. Sowohl das F1-Maß mit 0,31 als auch der MCC mit 0,15 liegen auf einem niedrigen Niveau.

252	66	186
38	16	22
214	50	164

korrekte Vorhersagen	180	0,714
inkorrekte Vorhersagen	72	0,286

Trefferquote (TPR)	0,242	Missrate (FNR)	0,758
Ausfallrate (FPR)	0,118	Spezifität (TNR)	0,882
Prävalenz	0,262	Genauigkeit (ACC)	0,714
Relevanz (PPV)	0,421	Falscherkennungsrate (FDR)	0,579
Segreganz (NPV)	0,766	False omission rate (FOR)	0,234
F1-Maß	0,308	MCC	0,153

Tabelle 5.5: Vorhersage von semantisch korrekt ohne difficulty

In Abbildung 5.9 ist die ROC Kurve für das Netz ohne *difficulty* Knoten dargestellt. Trotz der vergleichsweise niedrigen Werte für das F1-Maß und den MCC beträgt die Fläche unter der Kurve 0,717.

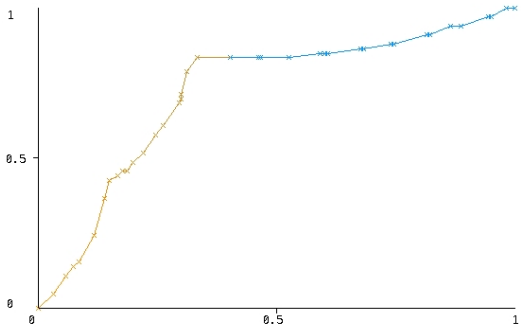


Abbildung 5.9: ROC Kurve ohne difficulty
AUC = 0,717
Threshold: 0,073-0.53

5.3 Bewertung

Abschließend werden die Ergebnisse der einzelnen Auswertungen miteinander verglichen. Zu Beginn werden die drei Stufen der Wissensstruktur betrachtet. Deren Ergebnisse sind in Tabelle 5.6 gegenübergestellt.

	form	syntax	semantik
Prävalenz	0,901	0,278	0,262
korrekte Vorhersagen	0,925	0,774	0,786
Trefferquote (TPR)	0,978	0,786	0,591
Spezifität (TNR)	0,440	0,769	0,855
Relevanz (PPV)	0,941	0,567	0,591
Segreganz (NPV)	0,688	0,903	0,855
F1-Maß	0,959	0,659	0,591
MCC	0,512	0,511	0,446

Tabelle 5.6: Übersicht über die drei Stufen der Wissensstruktur

Für alle drei Stufen wurden mehr als 75 Prozent sämtlicher Fälle richtig klassifiziert, für Form korrekt sogar über 90 Prozent. Allerdings wies der Klassifikator bei allen Stufen in einem gewissen Bereich Probleme auf und schnitt hier schlechter ab. Während die Vorhersage für Form korrekt eine schlechte Spezifität und eine schlechtere Segreganz aufweist, ist bei syntaktisch und semantisch die Relevanz der schlechteste Wert, semantisch zusätzlich auch die Trefferquote. Auffällig ist, dass dies jeweils der Bereich mit geringer Fallanzahl ist. 90 Prozent der Fälle weisen eine korrekte Form auf, jedoch nur etwas mehr als 25 Prozent sind syntaktisch und gegebenenfalls semantisch korrekt.

Die Verteilung an positiven sowie negativen Fällen sind jeweils stark unausgewogen. Es ist folglich möglich, dass der Klassifikator mit einer größeren Datenmenge noch bessere Resultate erzielen kann. Andererseits, wie bereits eingangs des Kapitels erwähnt, ist es häufig der Fall, dass Klassifikatoren in bestimmten Bereichen besser abschneiden als in anderen. Ähnlich wie medizinische Tests, lassen sich diese nicht auf alle Bereiche optimieren.

Das F1-Maß weist für die Vorhersage von Form korrekt einen sehr hohen Wert auf. Die Werte für die anderen beiden Variablen sind deutlich niedriger. Allerdings liegen die Werte für den Matthews Correlation Coefficient deutlich näher zusammen. Da dieser neben der Trefferquote auch die negativen Fälle berücksichtigt liegen die Ergebnisse nahe aneinander.

Daneben wurde auch die Performance des Schätzers ohne den „difficulty“ Knoten ausgewertet. In Tabelle 5.7 sind die Ergebnisse sowohl für das Netz mit Knoten als auch ohne Knoten eingetragen. Hierbei wurden die Vorhersagen jeweils für semantisch korrekt erstellt. Mit dem Knoten wurden etwa 7 Prozent mehr Fälle richtig vorhergesagt. Auffällig ist das deutlich schlechtere Abschneiden hinsichtlich der Trefferquote. Auch für die Relevanz und die Segreganz sind die Werte niedriger. Lediglich die Spezifität zeigte sich besser beim Netz ohne den Knoten. Folglich sind die beiden schlechtesten Werte Trefferquote und Relevanz beim Netz ohne den Knoten nochmals schlechter. Beziehungsweise wurde durch die Hinzunahme des Knotens der Klassifikator in diesen Bereichen teils deutlich verbessert. Sollten zukünftig weitere Daten in die Plattform integriert werden, kann es sich folglich lohnen, das Netz um weitere Variablen zu erweitern.

Das F1-Maß des Netzes ohne den Knoten beträgt lediglich etwa die Hälfte des F1-Maßes des Netzes mit dem Knoten. Ähnlich verhält sich Matthews Correlation Coefficient. Dieser beträgt lediglich ein Drittel des Wertes des Netzes mit „difficulty“ Knoten.

	mit Difficulty	ohne Difficulty
Prävalenz	0,262	0,262
korrekte Vorhersagen	0,786	0,714
Trefferquote (TPR)	0,591	0,242
Spezifität (TNR)	0,855	0,882
Relevanz (PPV)	0,591	0,421
Segreganz (NPV)	0,855	0,766
F1-Maß	0,591	0,307
MCC	0,446	0,153

Tabelle 5.7: Vergleich für Netz mit und ohne „difficulty“ Knoten

Neben den durchgeführten Auswertungen wären insbesondere noch Auswertungen anderer Veranstaltungen von Interesse. Allerdings gibt es hierzu aktuell keine nutzbaren Daten. Die Vorlesungen an der Fakultät für Informatik der Ludwig-Maximilians-Universität werden nicht immer vom selben Lehrstuhl durchgeführt. Außerdem werden nicht in allen Veranstaltungen des Lehrstuhls genügend Programmieraufgaben in den Übungen gestellt, um diese für eine Auswertung zu nutzen. Folglich bietet es sich an, sobald passende Daten vorliegen, eine weitere Auswertung zu erstellen und beispielsweise die Ergebnisse der Veranstaltungen zu vergleichen.

KAPITEL 6

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, eine automatische Lerngruppenbildung zu entwickeln und zu implementieren. Voraussetzung und Grundlage hierzu war die Nutzung einer Kompetenzschätzung mit der Auflage diese in eine bereits bestehende Projektverwaltungswebsite zu integrieren. Als Basis für die Kompetenzschätzung sollten lediglich die in der Plattform bereits vorhandenen Daten genutzt werden. Dabei handelte es sich um Daten, die aus von Studenten bearbeiteten Programmieraufgaben gewonnen wurden.

Zu Beginn der Arbeit wurden zunächst verwandte Forschungen zur Gruppenbildung und zur Kompetenzeinschätzung von Studenten betrachtet. Dabei fiel auf, dass die meisten Studien sowohl Daten von persönlichen Leistungen, wie etwa Noten etc., als auch von persönlichen Eigenschaften, wie etwa das Alter etc., nutzten. Mithilfe dieser Daten wurden die Studenten in den Studien eingeschätzt und in Gruppen eingeteilt. Bei einer dieser betrachteten Studien resultierte eine heterogene Einteilung der Studenten in einem schlechteren Abschneiden als dies von den Forschern erwartet wurde, während die homogene Gruppe besser als erwartet abschnitt.

Nachfolgend wurden für diese Arbeit wichtige Konzepte untersucht. Dies war zum einen die Knowledge Space Theory, mit der das Wissen der Studenten in aufeinander aufbauenden Stufen gegliedert wurde. Zum anderen Bayessche Netze, die als Basis für die Kompetenzschätzung dienen. Beide Konzepte wurden durch Beispiele kurz erläutert.

Anschließend erfolgte im dritten Kapitel die Betrachtung der verwendeten Daten. Von den etwa 750 gestellten Aufgaben wurden während der Veranstaltung etwa 320 bearbeitet und abgegeben. Dies entsprach einer Quote von 43 Prozent. Nicht alle Aufgaben eigneten sich zur Auswertung. Somit konnten etwa 250 Abgaben genutzt werden. Bei lediglich 26 Prozent der Abgaben handelte es sich um korrekte Abgaben.

Darauf folgend wurde die Konstruktion des Netzes für den Kompetenzschätzer erläutert. Dieses besteht erstens aus den drei Wissensstufen „Form korrekt“ (sofern eine Abgabe im richtigen Format abgegeben wurde und auswertbar ist), „syntaktisch korrekt“ (sofern eine Abgabe kompiliert) und „semantisch korrekt“ (sofern eine Abgabe alle Tests besteht). Zum zweiten aus drei „Historywerten“, die das Abschneiden des jeweiligen Studenten in den vorherigen Aufgaben repräsentiert. Sowie drittens einen Wert, der die Schwierigkeit der Aufgabe beschreibt.

Im Anschluss wurde die Gruppenbildung erläutert. Hierbei wurden unterschiedliche Einsatzmöglichkeiten berücksichtigt. Es ist möglich sowohl homogene, heterogene und gemischte Gruppen als auch Zuordnungen zu Peer Reviews vorzunehmen. Dazu muss

eine Ordnung der Studenten angegeben werden. Diese kann zum Beispiel auf der Kompetenzschätzung beruhen. Im vierten Kapitel wurde auf die Implementierung der einzelnen Bestandteile eingegangen. Dabei wurde eine Anbindung an einen Compileservice, der Kompetenzschätzer sowie die Gruppenbildung implementiert. Es erfolgte die Vorstellung des verwendeten Frameworks Weka. Abschließend wurde im fünften Kapitel eine Evaluierung des Kompetenzschätzers durchgeführt und dabei die Performance des Schätzers für verschiedene Vorhersagen ermittelt, sowie aufgezeigt, wie sich die Qualität des Kompetenzschätzers durch die Repräsentation der Schwierigkeit einer Aufgabe im Netz verbessert.

Für diese Arbeit wurde die Projektverwaltungswebsite um folgende Funktionen erweitert:

- Durch die Anbindung an den bereits bestehenden Compileservice können Programmieraufgaben automatisch ausgewertet werden.
- Mit dem Kompetenzschätzer ist es möglich die Resultate der Abgaben der Studenten bei zukünftigen Aufgaben abzuschätzen.
- Mit der flexiblen Implementierung des Gruppenbildungsmoduls können Gruppen nicht nur auf Basis der Kompetenzschätzung eingeteilt werden, sondern auch nach beliebigen anderen Kriterien.

Zukünftig sollten weitere Evaluierungen durchgeführt werden. Insbesondere bietet es sich an, den Kompetenzschätzer in weiteren Veranstaltungen zu testen. An der Fakultät für Informatik an der Ludwig-Maximilians-Universität München werden die einzelnen Vorlesungen nicht immer vom selben Lehrstuhl durchgeführt. Nicht alle Vorlesungen weisen geeignete Programmieraufgaben zur Auswertung auf. Deshalb konnte eine solche Auswertung während der Bearbeitung der vorliegenden Arbeit mangels passender Veranstaltungen nicht durchgeführt werden.

Unter Umständen kann der Kompetenzschätzer durch Berücksichtigung weiterer Daten noch verbessert werden; wie dies durch die Hinzunahme der Schwierigkeit der Aufgaben geschah. In den aufgelisteten Studien zu Beginn der Arbeit war ersichtlich, dass eine Leistungseinschätzung nicht nur von Daten zu persönlichen Leistungen sondern auch von Daten zu persönlichen Eigenschaften profitieren kann. Diese sind momentan jedoch nicht Bestandteil der Plattform und könnten im Bedarfsfall noch ergänzt beziehungsweise erhoben werden.

Ferner bietet es sich an, unter anderem für die Bildung von Praktikumsgruppen, weitere Faktoren zu berücksichtigen. Für eine erfolgreiche Gruppenarbeit sind neben den Fähigkeiten der einzelnen Mitglieder auch deren Motivation ein wichtiges Kriterium. Hierfür könnte der Schätzer erweitert oder aber besser ein weiterer Schätzer erstellt werden, der Daten aus der Plattform nutzt um die Motivation abzuschätzen. Denkbar wäre dafür die Erhebung von Daten über die Regelmäßigkeit von Abgaben oder auch über die verbleibende Zeit bis zur Deadline der Abgabe. Dabei könnten auch andere Aufgaben als Programmieraufgabe sowie zusätzlich fachfremde Aufgaben genutzt werden.

Außerdem sollte beobachtet werden, ob bei einer heterogenen Gruppierung der Studenten auch die erwarteten Resultate erzielt werden. Zum Beispiel ist es gewünscht ein besseres Lernen für die Studenten zu erreichen oder aber wie bei der Studie von Carrell at al. [2] gezielt eine gewisse Gruppe von Studenten zu verbessern. In dieser Studie trat jedoch ein gegenteiliger Effekt auf, die Studenten verschlechterten sich. Allerdings handelt es sich bei Gruppen für Programmierpraktika um in der Regel deutlich kleinere Gruppen als in der Studie, meist im Bereich von 3-6 Studenten. Somit sollte der bei der Studie beobachtete Effekt von Untergruppen nicht oder weniger stark auftreten. Durch die heterogene Verteilung der Studenten soll zudem für alle Gruppen ein möglichst fair verteiltes Wissen erreicht werden. Hierdurch können zum einen fairere Aufgaben gestellt werden und zum anderen ist der Betreuungsaufwand für die einzelnen Gruppen besser abschätzbar.

Appendix

Die Homepage der Veranstaltung ist unter folgendem Link zu finden:
`https://www.pms.ifi.lmu.de/lehre/pum/17ws18/`.

Der Sourcecode der implementierten Software ist zu finden unter:
`https://gitlab.pms.ifi.lmu.de/niels/cwdl-projects`
in dem Branch *group_diversification* (Letzter commit: 490b851d)

Literaturverzeichnis

[1] AHADI, ALIREZA, RAYMOND LISTER, HEIKKI HAAPALA und ARTO VIHAVAINEN: *Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance*. In: *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*. ACM Press, 2015.

[2] CARRELL, SCOTT E., BRUCE I. SACERDOTE und JAMES E. WEST: *From Natural Variation to Optimal Policy? The Importance of Endogenous Peer Group Formation*. Econometrica, 81(3):855–882, 2013.

[3] DOIGNON, JEAN-PAUL und JEAN-CLAUDE FALMAGNE: *Knowledge Spaces*. Springer Berlin Heidelberg, 1999.

[4] ERTEL, WOLFGANG: *Grundkurs Künstliche Intelligenz*. Springer Fachmedien Wiesbaden, 2016.

[5] FAWCETT, TOM: *An introduction to ROC analysis*. Pattern Recognition Letters, 27(8):861–874, Juni 2006.

[6] HASTIE, TREVOR, ROBERT TIBSHIRANI und JEROME FRIEDMAN: *The Elements of Statistical Learning*. Springer New York, 2009.

[7] HECKERMAN, DAVID, DAN GEIGER und DAVID M. CHICKERING: *Learning Bayesian networks: The combination of knowledge and statistical data*. Machine Learning, 20(3):197–243, September 1995.

[8] JENSEN, FINN V.: *Bayesian networks basics*. `http://fitelson.org/269/Jensen_BNB.pdf`. besucht am 08.09.2018.

[9] KAMBOURI, MARIA, MATHIEU KOPPEN, MICHAEL VILLANO und JEAN-CLAUDE FALMAGNE: *Knowledge assessment: tapping human expertise by the QUERY routine*. International Journal of Human-Computer Studies, 40(1):119–151, Januar 1994.

[10] KHASANAH, ANNISA USWATUN und HARWATI: *A Comparative Study to Predict Student's Performance Using Educational Data Mining Techniques*. IOP Conference Series: Materials Science and Engineering, 215:012036, Juni 2017.

[11] OUNNAS, ASMA, HUGH C DAVIS und DAVID E MILLARD: *A Framework for Semantic Group Formation in Education*. Journal of Educational Technology & Society, 12(4):43–55, 2009.

[12] PEARL, JUDEA: *Probabilistic Reasoning in Intelligent Systems - Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, Calif, 1988.

[13] POWERS, DAVID MARTIN: *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. Journal of Machine Learning Technologies, 2011.

[14] RUSSELL, STUART J. und PETER NORVIG: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[15] SERVICES, EMC EDUCATION: *Data Science and Big Data Analytics*. John Wiley & Sons Inc, 2015.

[16] VILLANO, MICHAEL: *Probabilistic student models: Bayesian Belief Networks and Knowledge Space Theory*. In: *Intelligent Tutoring Systems*, Seiten 491–498. Springer Berlin Heidelberg, 1992.