Daniel Simon Siemmeister

Erprobung unterschiedlicher Machine Learning Modelle zur Schätzung der Prüfungsaktivität von Studierenden

Masterarbeit

zur Erlangung des akademischen Grades
Master of Education

im Studium Lehramt Sekundarstufe Allgemeinbildung
im Entwicklungsverbund Süd-Ost

vorgelegt an der
Karl-Franzens-Universität Graz

unter der Anleitung von
Univ.-Prof. Dr. Gunther Leobacher

ausgeführt am
Institut für Mathematik und wissenschaftliches Rechnen

Graz, 26. März 2022

Danksagungen

An dieser Stelle möchte ich mich bei Herrn Prof. Gunther Leobacher bedanken, der es mir ermöglicht hat, meine Masterarbeit zu diesem spannenden Thema zu verfassen. Neben der unbürokratischen Zusammenarbeit möchte ich mich für die fachliche Unterstützung und den Fokus auf das Wesentliche bedanken.

Weiters möchte ich mich bei meiner Familie bedanken, die mich während meines Studiums sowohl finanziell als auch zwischenmenschlich unterstützt hat, was ich nicht als selbstverständlich empfinde.

Graz, am 26. März 2022

Inhaltsverzeichnis

Abkürzungsverzeichnis

CV	Crossvalidierung. 46
ECTS	European Credit Transfer and Accumulation System. (1 ECTS enspricht einem Arbeitsaufwand von 25 Echtstunden à 60 Minuten). 8
FN	Studierende, die als nicht prüfungsaktiv klassifiziert werden, aber es tatsächlich waren. 49
FP	Studierende, die als prüfungsaktiv klassifiziert werden, aber es tatsächlich nicht waren. 49
KNN	Künstliche Neuronale Netzwerke. 42
LQM	Leistungs- und Qualitätsmanagement (Abteilung der Universität Graz). 8
MAE	Mean Absolut Error. 48
RMSE	Root Mean Squared Error. 48
TN	Studierende, die als nicht prüfungsaktiv klassifiziert werden und es tatsächlich nicht waren. 49
TP	Studierende, die als prüfungsaktiv klassifiziert werden und es tatsächlich waren. 49

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die vorliegende Arbeit wurde im Rahmen eines Auftrags des Leistungs- und Qualitätsmanagement (Abteilung der Universität Graz) (LQM) verfasst. Sie hat auf der Vorarbeit des LQM aufgebaut und sollte diese weiterführen und vertiefen.

Alle drei Jahre muss eine österreichische Universität dem Bundesministerium für Bildung und Forschung bekanntgeben, wie viele *prüfungsaktive* Studierende sie voraussichtlich in **drei Jahren** in der Zukunft haben wird. Umso genauer eine Universität diese Zahl vorhersagen kann, desto besser. Auf der einen Seite bekommt die jeweilige Universität vom Bundesministerium weniger Budget zugesprochen und kann somit den Universitätsbetrieb nicht optimal finanzieren, wenn sie zu wenig prüfungsaktive Studierende vorhersagt. Auf der anderen Seite bekommt sie zuerst mehr Budget zugesprochen und muss dieses aber später wieder zurückzahlen, wenn die jeweilige Universität zu viel prüfungsaktive Studierende vorhersagt. Das kann dann bei größeren geplanten Projekten, wie beispielsweise Laboren, zum Abbruch des Projektes führen und somit einen Schaden verursachen.

Generell gilt eine studierende Person in einem Jahr als prüfungsaktiv, wenn sie:

- 16 oder mehr ECTS in diesem Jahr positiv absolviert hat, oder sie
- 8 oder mehr Semesterwochenstunden positiv absolviert hat, oder sie
- in diesem Studienjahr ihr Studium positiv abschließt.

Andernfalls gilt die Person als *nicht* prüfungsaktiv. ECTS bedeutet European Credit Transfer and Accumulation System. (1 ECTS enspricht einem Arbeitsaufwand von 25 Echtstunden à 60 Minuten) (ECTS)

1.1 Bisherige Herangehensweise

Vom LQM wurde mittels unterschiedlicher Machine Learning Ansätze eine Schätzung der Prüfungsaktivität für das kommende Jahr auf der Ebene der Studierenden gemacht. Hierbei wurde versucht, für jeden Studierenden vorherzusagen, ob er im darauffolgenden Jahr prüfungsaktiv sein wird oder nicht. Diese Modelle wurden eingesetzt, um Merkmalskombinationen von Studierenden herauszufinden, die auf eine hohe Wahrscheinlichkeit der Prüfungsaktivität schließen lassen. Weiters wurden Ansätze auf aggregierter Ebene überlegt. Das bedeutet, dass man nicht die Prüfungsaktivität jedes einzelnen Studierenden vorhersagen will. Stattdessen versucht man den Anteil an einer Menge von Studierenden, die prüfungsaktiv sein werden, vorherzusagen. Die Ansätze auf aggregierter Ebene sind bisher noch nicht vollständig formuliert und ausprobiert worden.

Bisher wurde das Problem indirekt bearbeitet, indem man probiert hat, die Prüfungsaktivität immer für das folgende Jahr vorherzusagen. Man hat keine Vorhersagen in die weitere Zukunft gemacht. Dabei hat man versucht die Prüfungsaktivität anhand der Merkmale von Studierenden zu erklären.

1.2 Problemstellungen

Ausgehend von Daten von Studierenden in der Vergangenheit will man eine Vorhersage der **Anzahl** der prüfungsaktiven Studierenden in den kommenden Jahren durchführen. Insbesondere soll die Anzahl der prüfungsaktiven Studierenden in **drei Jahren** in der Zukunft vorhergesagt werden. Die Schwierigkeit besteht darin, dass man zum Zeitpunkt, wo man die Schätzung durchführt, nur Daten der derzeit Studierenden im ersten, zweiten, dritten Studienjahr und höheren Jahren zur Verfügung hat. Davon ausgehend soll mehr als ein Jahr, eben drei Jahre in die Zukunft, versucht werden die Anzahl an prüfungsaktiven Studierenden zu schätzen.

Beispielsweise hat man im Jahr 2021 die Daten der zu dieser Zeit inskribierten Studierenden in ihrem jeweiligen Studienjahr zur Verfügung. Nun wird versucht, aufgrund dieser Daten vorherzusagen, wie viele prüfungsaktive Studierende es im Jahr 2024 geben wird. Dabei weiß man jedoch nicht, wie viele Personen im Jahr 2022 und 2023 zu studieren beginnen werden.

Zusätzlich sind die Modelle, welche für diese Vorhersage eingesetzt werden, auf Daten aus vergangen Studienjahren gebildet worden. Deshalb sind sämtliche Schätzungen auf der Annahme aufgebaut, dass etwaige Zusammenhänge von bestimmten Merkmalskombinationen aus der Vergangenheit, die Auswirkungen auf die Prüfungsaktivität einer studierenden Person haben, sich auch in die Zukunft übertragen zu lassen.

Das Problem lässt sich in zwei voneinander unabhängigen Komponenten unterteilen. Erstens versucht man aus den Daten von Studierenden, die man im Jahr der Schätzung zur Verfügung hat, die Anzahl der prüfungsaktiven Personen in drei Jahren zu schätzen. Hier kennt man die Anzahl der Studierenden und auch ihre Merkmalskombinationen. Zweitens soll man die neu inskribierenden Studierenden in den kommenden beiden Jahren in die Schätzung miteinbeziehen. Von diesen Personen hat man jedoch weder die Anzahl noch die Merkmalskombinationen.

Jede studierende Person wird in jedem Jahr mit m unterschiedlichen Merkmalen beschrieben. Beispiele dafür sind *„durchschnittliche ECTS bisher“*, *„kumulierte ECTS“*, *„Geschlecht“* oder auch *„positiv absolvierte ECTS“*. Alle verwendeten Eigenschaften der Studierenden sind im Kapitel Daten in einer Tabelle dargestellt. Die meisten dieser Merkmale werden zur Vorhersage verwendet und wenige von ihnen dienen als bereits vorhandener Zielwert. Um die Darstellung zu vereinfachen, fasst man für jede studierende Person S_i die Merkmale $E_i^{(j)}$ mittels eines Vektors zusammen:

$$S_i = \begin{bmatrix} E_i^{(1)} \\ E_i^{(2)} \\ \vdots \\ E_i^{(m)} \end{bmatrix},$$

wobei S_i mit $i = 1, \ldots, n$ für eine studierende Person in einem bestimmten Jahr steht und $1, \ldots, m$ die Merkmale nummeriert.

Weil die Vorhersage der prüfungsaktiven Studierenden über mehrere Jahre erfolgt, bezeichnet man mit $t \in \{0, 1, 2, 3\}$ die Zeit in Jahren, ab dem Vorhersagezeitpunkt. Es gibt Merkmale von Studierenden, die sich im Laufe der Zeit verändern. Beispiele hierfür sind *„kumulierte ECTS“* und *„ECTS im Jahr zuvor“*. Man fasst nun alle Studierenden im Jahr t in einem Zustand $Z^{(t)}$ zusammen:

$$Z^{(t)} := \{S_1^{(t)}, \ldots, S_{n(t)}^{(t)}\},$$

wobei t den jeweiligen Zeitpunkt angibt. Weiters stellt $n(t)$ die Anzahl der Studierenden im Jahr t dar. Wichtig ist auch, dass für jede studierende Person die Eigenschaft $E^{(1)}$

das aktuelle Studienjahr dieser Person im Jahr t darstellt.

Weil für $t > 0$ auch immer neu inskribierende Studierende in den jeweiligen Zuständen dazukommen, von denen man zum Zeitpunkt der Schätzung weder Anzahl noch individuelle Merkmalskombinationen kennt, kann man für $t > 0$, $Z^{(t)}$ in $Z^{(t)}_{neu}$ und $Z^{(t)}_{alt}$ unterteilen. $Z^{(t)}_{neu}$ steht für die neuinskribierenden Studierenden im Jahr t. In $Z^{(t)}_{alt}$ sind diejenigen Studierenden gemeint, die bereits in den Jahren zuvor inskribiert waren. Zusammenfassend gilt $Z^{(t)} = Z^{(t)}_{neu} + Z^{(t)}_{alt}$.

Problem 1 (von nun an P1 genannt) besteht darin, ausgehend von vorhandenen Daten der Studierenden zum Zeitpunkt $t = 0$, eine Schätzung der Anzahl an prüfungsaktiven Studierenden im Zustand $Z^{(3)}_{alt}$ zu machen.

Problem 2 (von nun an P2 genannt) stellt die Schätzung der Anzahl an prüfungsaktiven Studierenden im Zustand $Z^{(3)}_{neu}$ dar. Von diesen Studierenden kennt man zuvor weder die genaue Anzahl noch die Merkmalskombinationen.

Es ist entscheidend hervorzuheben, dass es nicht notwendig ist, für jeden einzelnen Studierenden zu wissen, ob er zum Zeitpunkt $t = 3$ prüfungsaktiv sein wird oder nicht. Vielmehr geht es darum, die absolute Häufigkeit an prüfungsaktiven Studierenden zum Zeitpunkt $t = 3$ zu schätzen. Durch diese Eigenschaft der Problemstellung ergibt sich der Fall, dass sich je eine *False Positive* klassifizierte Person mit einer *False Negative* klassifizierten Person in der Auswertung der Schätzungen aufheben. False Positive klassifizierte Studierende sind jene, die als prüfungsaktiv vorhergesagt werden, jedoch in der Realität nicht prüfungsaktiv sein werden. Analog sind False Negativ klassifizierte Studierende jene, die als nicht prüfungsakiv vorhergesagt werden, aber in der Realität prüfungsakiv sein werden.

1.3 Ziel

Das Ziel dieser Arbeit kann wie folgt beschrieben werden. Erstens werden drei Ansätze formuliert und erprobt, um P1 zu lösen. Diese Ansätze stellen unterschiedliche Herangehensweisen an dieselbe Problemstellung aus P1 dar. Zweitens werden für P2 zwei verschiedene Ansätze formuliert, welche sich in ihrer Herangehensweise an P2 unterscheiden.

Bei allen Lösungsansätzen wird der Grundgedanke verfolgt, dass man diese Problem-

stellung mit einem Populationsmodell beschreiben kann. In diesem Modell kennt man den aktuellen Zustand und will davon ausgehend eine möglichst gute Vorhersage über die Population zu einem zukünftigen Zeitpunkt machen. Es ist wichtig zu beachten, dass es Austritte aus der Population und auch Eintritte in sie gibt. Die Austritte sind in der vorliegenden Problemstellung Studierende, die ihr Studium abschließen oder abbrechen, und Eintritte sind Personen, die zu studieren beginnen. Vor allem über die zukünftigen Eintritte hat man wenig Informationen.

Sollte die Zeitspanne, über welche die Vorhersage angewandt wird, zu groß sein, würde es keinen Sinn machen, sich mit P1 zu beschäftigen. Da man aber anhand der Daten sieht, dass dies für die Zeitspanne von drei Jahren nicht der Fall ist, ist es relevant P1 zu lösen. Zum Beispiel haben jene Studierende, die in ihr erstes oder zweites Studienjahr kommen und ihr Studium je nach Studienrichtung nicht in der Mindeststudienzeit abschließen, auch in drei Jahren die Möglichkeit prüfungsakiv zu sein.

Abschließend möchte man die besten Lösungsansätze für P1 und für P2 zusammenführen. Somit wird eine Methode formuliert, wie man die Anzahl der prüfungsaktiven Studierenden in drei Jahren bestmöglich vorhersagen kann. Um die unterschiedlichen Lösungsansätze miteinander zu vergleichen und den aussichtsreichsten Ansatz auszuwählen, werden im Kapitel Auswertung entsprechende Metriken dafür beschrieben.

1.4 Daten

Für die Erprobung verschiedener Machine Learning Modelle wurde vom LQM der Datensatz *da_242* für die Studienrichtungen *Bachelorstudium Pädagogik*, *Bachelorstudium Betriebswirtschaft* und *Diplomstudium Rechtswissenschaften* bereitgestellt. Die betrachteten Jahre sind die Studienjahre 2015/2016 bis 2019/2020, somit also fünf Jahre.

Der Datensatz beinhaltet Daten von ca. 40 000 Studienjahren. Das bedeutet, dass jede studierende Person, welche über mehrere Jahre in einer der beobachteten Studienrichtungen inskribiert war, auch mehrere Einträge im Datensatz besitzt.

Insgesamt gibt es pro Studentin oder Student und Studienjahr ca. 100 Merkmale. Für die tatsächliche Vorhersage durch die Machine Learning Modelle wurden nur wenige davon verwendet. Der Grund dafür war, dass die verworfenen Merkmale großteils unvollständig für den gesamten Datensatz waren und dass sie oft keine weiteren In-

formationen zu den verwendeten Merkmalen beinhaltet haben. Jene Merkmale, mit welchen hautpsächlich gearbeitet wurde, sind in der Tabelle 1.1 zusammengefasst und mit ihren Labels beschrieben. Die Eigenschaften, welche mit * gekennzeichnet sind, versucht man vorherzusagen.

Tabelle 1.1: Eigenschaften der Studierenden und deren Ausprägung

E_i	Name	Ausprägungen
E_1	Aktuelles Studienjahr	numerisch
E_2	Matrikelnummer	numerisch
E_3	Geschlecht	binär
E_4	Besuchter Schultyp	one-hot-encoding
E_5	Verspätet angemeldet	binär
E_6	Herkunft	one-hot-encoding
E_7	Inskription in mehreren Studien	numerisch
E_8	Jahre seit Matura	numerisch
E_9	Jahre seit 18	numerisch
E_{10}	ECTS pro Semester	numerisch
E_{11}	Vorbildung der Eltern	binär
E_{12}	ECTS im Jahr zuvor (wenn vorhanden)	numerisch
E_{13}	Studienart	one-hot-encoding
E_{14}	Erste Prüfung negativ	binär
E_{15}	Geplante Mindeststudienzeit	numerisch
E_{16}	Bisherige Studiendauer in Semestern	numerisch
E_{17}*	Studienstatus kommendes Jahr	nominal
E_{18}*	ECTS dieses Jahr	numerisch
E_{19}*	Semesterwochenstunden dieses Jahr	numerisch
E_{20}*	Aktiv oder nicht	binär

Grundsätzlich sind die Merkmale *„weiterer Studienstatus“*, *„ECTS in diesem Jahr“*, *„Semesterwochenstunden in diesem Jahr“* und *„prüfungsaktiv oder nicht“* erst nach einem absolvierten Studienjahr verfügbar. Somit sind diese Merkmale, die bereits bekannten Zielwerte der Daten. Das Ziel der Vorhersagemodelle ist es, aus vorhandenen Trainingsdaten eine verlässliche Regel zu lernen, um diese Werte für neue Daten ohne gegebene Zielwerte zu schätzen. Die anderen Merkmale können, je nach Nützlichkeit zur Vorhersage, als Input in den Modellen verwendet werden oder nicht.

Die Merkmale unterscheiden sich auch hinsichtlich ihrer Veränderbarkeit im zeitlichen Verlauf. Eigenschaften wie *„Studienrichtung“*, *„Geschlecht“*, *„Herkunft“* oder *„besuchter Schultyp“* verändern sich im Laufe der Zeit nicht und bleiben über die Studienzeit gleich. Im Gegensatz dazu ändern sich *„kumulierte ECTS“* oder *„ECTS im Jahr zuvor“* in jedem Studienjahr.

Deswegen werden Ansätze ausprobiert, die nur auf unveränderbaren Merkmalen beru-

hen. Weiters werden Ansätze formuliert, wo versucht wird die veränderlichen Merkmale bestmöglich zu verwenden.

Die Prüfungsaktivität kann, wie oben beschrieben, mehrere Einflussfaktoren haben. Einer dieser Faktoren ist der Abschluss des Studiums. Das ist in zweierlei Hinsicht interessant. Auf der einen Seite bedeutet ein Studienabschluss, dass man zwar in diesem Studienjahr prüfungsaktiv sein wird, jedoch danach auch sein Studium beendet hat. Dadurch kann man in weiterer Folge in den darauffolgenden Jahren nicht mehr prüfungsaktiv sein. Auf der anderen Seite benötigt man für einen Studienabschluss eine festgelegte Anzahl an kumulierter, positiv absolvierter ECTS. Dadurch besteht die Möglichkeit, nur durch den Abschluss des Studiums prüfungsaktiv zu sein erst, wenn man genügend ECTS erreicht hat, und nicht schon ab dem ersten Studienjahr. Da die Anzahl an ECTS, die man für einen Abschluss erreichen muss, nach Studienrichtung variiert, ist in Abbildung 1.1 der Anteil an prüfungsaktiven Studierenden nach Studienjahr dargestellt.

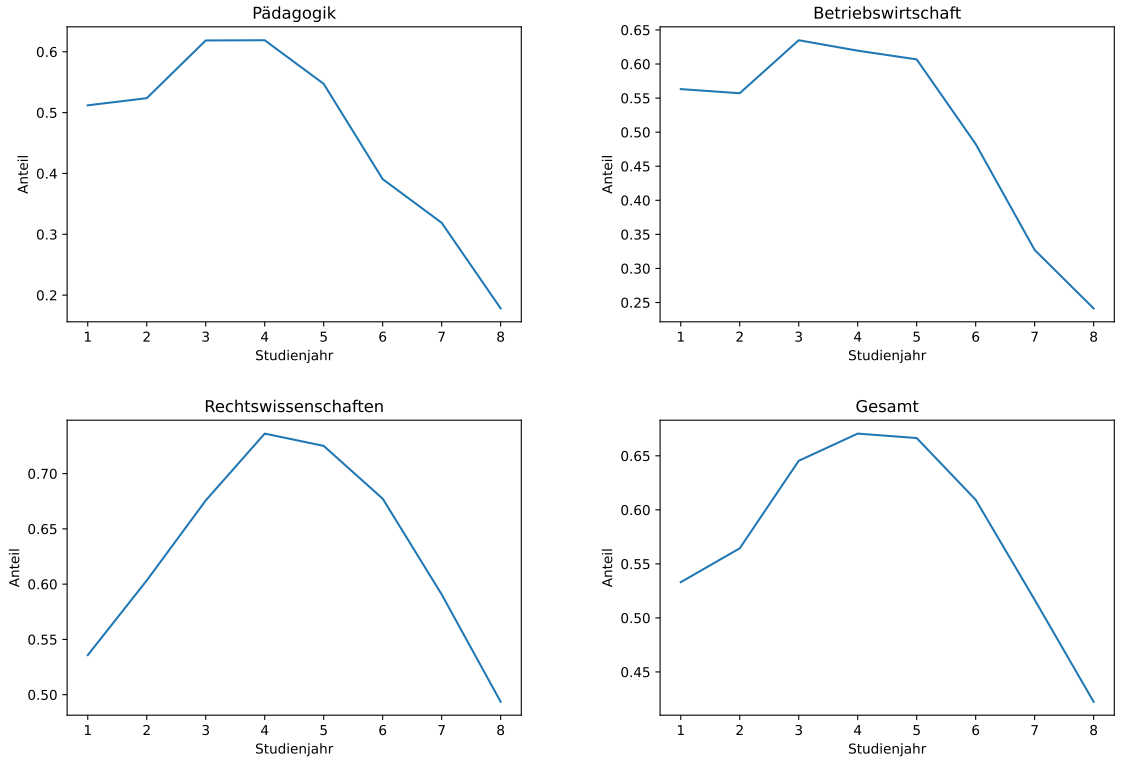

Abbildung 1.1: Der Anteil an prüfungsaktiven Studierenden an insgesamt Studierenden, die noch bis ins jeweilige Studienjahr verblieben sind. In der Grafik *Gesamt* wird ein gewichteter Anteil nach Studienrichtung gezeigt.

Eine Grundannahme in sämtlichen erprobten Ansätzen ist, dass Studierende, von denen man Anzahl und Merkmalskombinationen genau kennt, auch in einem Zeitrahmen von drei Jahren in der Zukunft auch noch einen beachtlichen Anteil an den prüfungsaktiven

Studierenden bilden werden. Diese Annahme wird gestützt, weil man, wie in Abbildung 1.2 dargestellt, sieht, dass der Anteil an prüfungsaktiven Studierenden aus höheren Studienjahren tatsächlich groß ist. Wäre das nicht der Fall, müsste man sich nicht mit P1 auseinandersetzen.

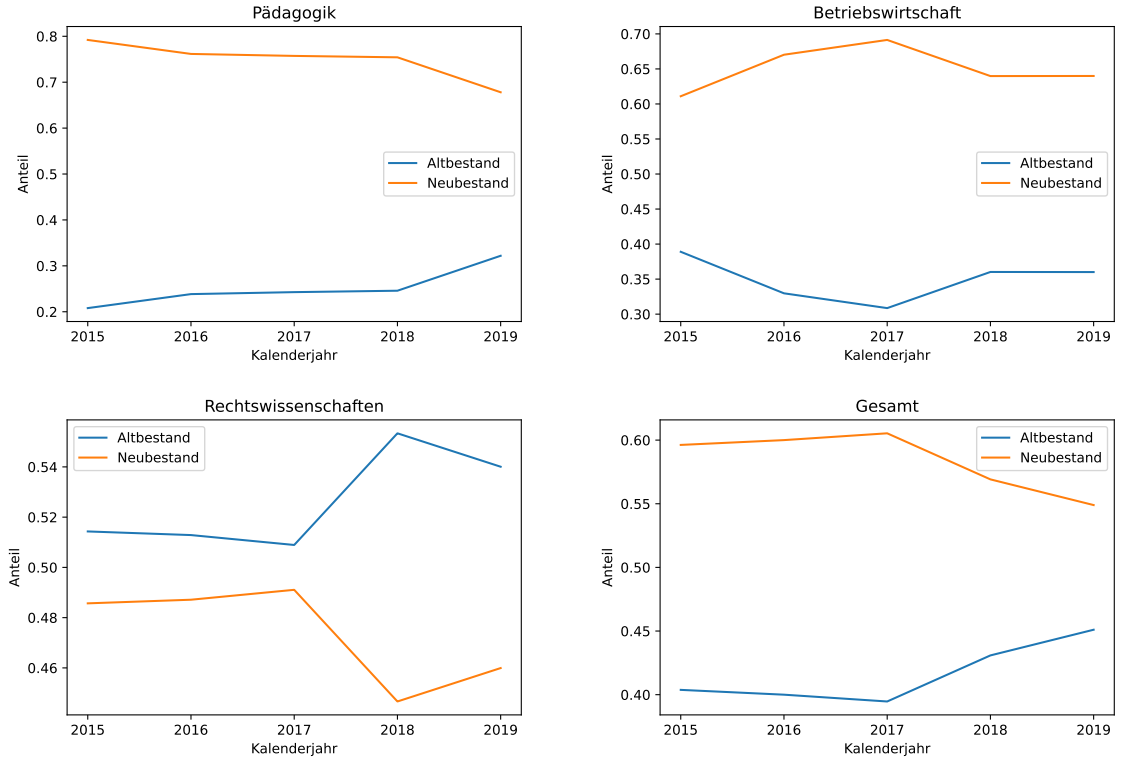

Abbildung 1.2: Der Anteil an prüfungsaktiven Studierenden, welcher bereits im vierten oder einem höheren Studienjahr vorliegt (Altbestand) und der Anteil, welcher erst im dritten oder einem niedrigeren Studienjahr ist (Neubestand).Die Grafik *Gesamt* stellt einen gewichteten Anteil nach Studienrichtung dar.

Weil man sich in P2 mit den zukünftigen Studienbeginnern beschäftigt, ist es wichtig zu wissen, wie sich diese Zahl im Verlauf der Zeit verändert. In Abbildung 1.3 sieht man, wie sich diese Zahlen je nach Studienrichtung und als Zusammenfassung aller Studienrichtungen verändern.

Um einen besseren Einblick in die Zahlen der Studierenden nach Studienjahr und Kalenderjahr zu bekommen, sind in Tabelle 1.2 sämtliche Zahlen nach Studienrichtung angeführt.

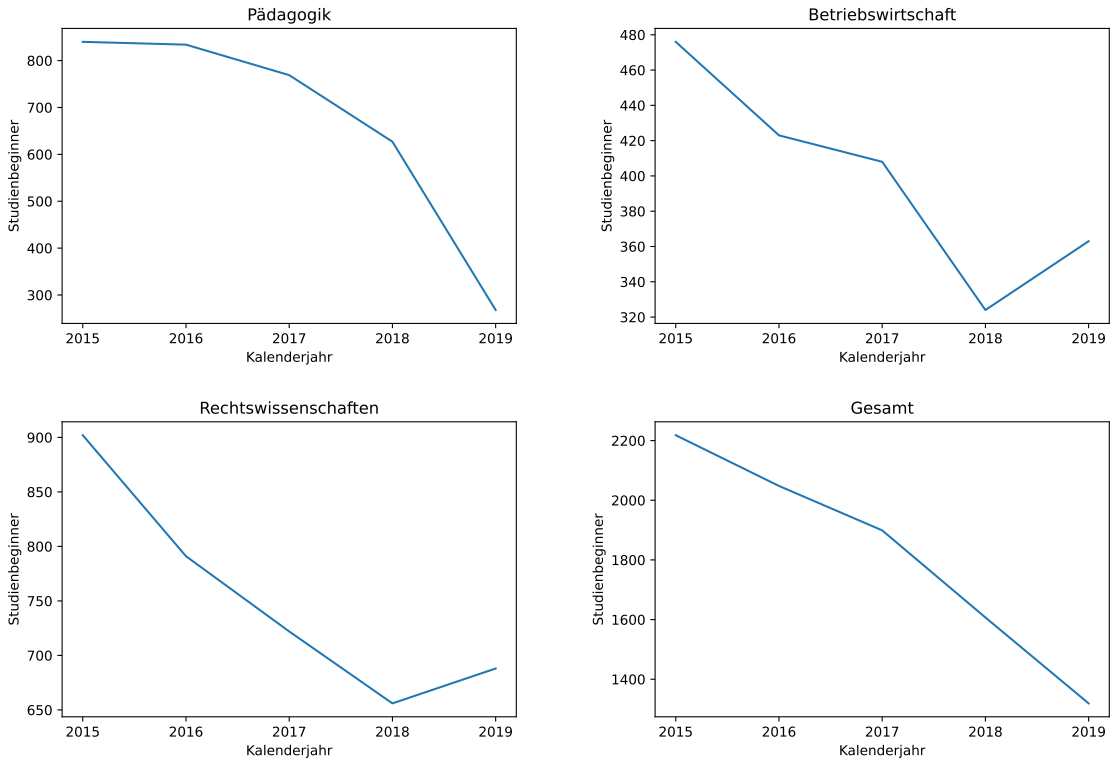

Abbildung 1.3: Hier werden die Anzahlen an Studienbeginnern je nach Fach und Kalenderjahr dargestellt.

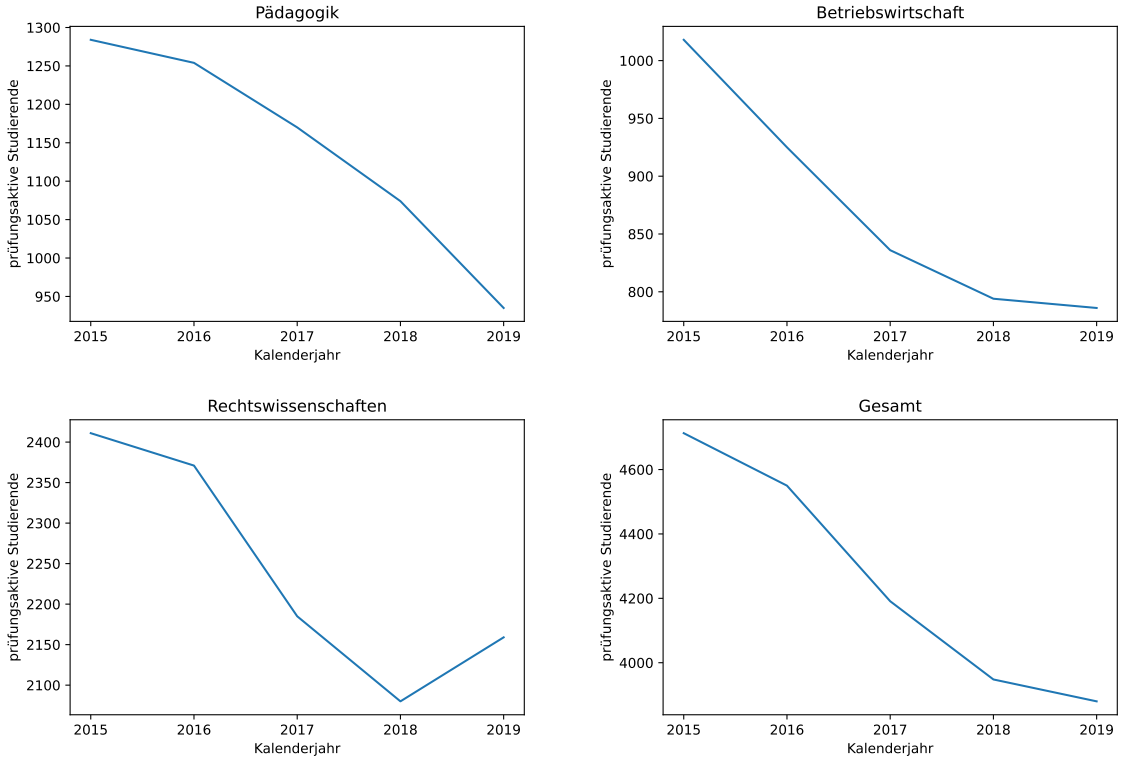

Abbildung 1.4: Hier wird die Anzahl aller prüfungsaktiven Studierenden nach Kalenderjahr dargestellt. Das ist jene Zahl, die anschließend in der Zukunft geschätzt werden soll.

Tabelle 1.2: Anzahl der Studierenden nach Studienrichtung, Kalenderjahr und Studienjahr

		Jahr 1	Jahr 2	Jahr 3	Jahr 4	Jahr 5	Jahr 6	Jahr > 7	**Gesamt**
2015/16	JUS	902	667	475	409	397	382	1253	4485
	BWL	476	353	231	355	153	104	399	2071
	PAD	804	567	415	270	115	48	115	2370
2016/17	JUS	791	668	479	404	363	333	1285	4323
	BWL	423	357	308	177	181	73	374	1893
	PAD	834	538	419	282	122	57	126	2378
2017/18	JUS	722	579	501	401	368	291	1257	4119
	BWL	408	314	286	228	75	83	324	1718
	PAD	769	578	419	279	124	68	139	2376
2018/19	JUS	656	541	413	413	360	294	1153	3830
	BWL	324	292	254	202	113	49	270	1504
	PAD	627	453	444	268	120	56	128	2106
2019/20	JUS	688	502	386	363	367	296	1113	3715
	BWL	363	242	244	192	96	66	231	1434
	PAD	268	415	335	292	122	50	138	1620
Gesamt Jahre	JUS	3759	2957	2254	1990	1855	1596	6061	20472
	BWL	1994	1558	1323	1154	618	375	1598	8620
	PAD	3302	2551	2032	1391	603	279	656	10850
Gesamt		9091	7066	5609	4535	3076	2250	8315	39942

2 Methoden und Modelle

In diesem Abschnitt werden Lösungsansätze für P1 und P2 und deren Implementierung vorgestellt. Weiters werden die unterschiedlichen Machine Learning Modelle erklärt, die in den Lösungsansätzen verwendet werden. Abschließend wird dargelegt, wie die unterschiedlichen Ansätze und auch die einzelnen Machine Learning Modelle verglichen und ausgewertet werden.

2.1 Ansätze für Problem 1

P1 stellt die Schätzung der Anzahl der prüfungsaktiven Studierenden im Jahr $t = 3$ aus den bestehenden Studierenden im Jahr $t = 0$ dar. Von diesen Studierenden kennt man die Anzahl und die Merkmalskombination jeder einzelnen Person.

2.1.1 Ansatz 1

In Ansatz 1 versucht man die Prüfungsaktivität der Studierenden Jahr für Jahr zu modellieren und so viel Information wie möglich weiterzuverwenden. Insbesondere soll für das jeweilige darauffolgende Studienjahr der ECTS-Wert vorhergesagt werden. Viele weitere Eigenschaften der studierenden Person können dann aus diesem ECTS-Wert abgeleitet werden.

Der Ausgangspunkt ist, dass sich für alle $S_i \in Z^{(t)}$ der erste Eintrag $E_i^{(1)}$ (aktuelles Studienjahr) um eins erhöht. Nun wird versucht eine Funktion zu finden, welche jeder studierenden Person $S_i \in Z^{(t)}$ einen passenden ECTS-Wert vorhersagt. Somit kann für diese Person der Übergang nach $Z_{alt}^{(t+1)}$ beschrieben werden und man hat alle Einträge zur Verfügung, die man auch von den zuvor gegebenen Daten hatte. Der Ansatz besteht darin, die Funktion wie folgt zu bilden.

$$F(S_i) = \begin{cases} h_1(S_i), & \text{für } E_i^{(1)} = 1 \\ h_2(S_i), & \text{für } E_i^{(1)} \geq 2 \end{cases}$$

Die Funktionen h_1 und h_2 sind Schätzfunktionen, die einer gewissen Merkmalskombination einer studierenden Person in einem Studienjahr einen ECTS-Wert zuordnen.

Studierende im ersten Studienjahr werden gesondert betrachtet, weil für sie keine Einträge mit ECTS-Werten vorhanden sind. Aufgrund dessen gibt es für sie weniger Inputwerte. Nun gibt es verschiedene Möglichkeiten h_1 und h_2 auszuwählen. In dieser Arbeit werden folgende Machine Learning Modelle ausprobiert:

- Multiple Lineare Regression
- Random Forest Modelle
- Support Vector Machines
- Künstliche neuronale Netzwerke

Jedes Modell bekommt als Input die Eigenschaften einer studierenden Person. Anhand dieser Eigenschaften ist es das Ziel, möglichst genau vorherzusagen, wie viele ECTS diese Person im kommenden Studienjahr erreichen wird.

Je nachdem wie gut die einzelnen Vorhersagefunktionen für die vorliegende Problemstellung funktioneren und angepasst werden können, wählt man anhand einer Metrik, die später beschrieben wird, jenes Modell aus, welches am besten die jeweiligen ECTS-Werte vorhersagen kann. Wichtig ist, dass jedes dieser Modelle eine Regression der ECTS für das aktuelle Studienjahr durchführt. Aufgrund der geschätzten ECTS kann dann entschieden werden, ob die Person prüfungsaktiv ist oder nicht. Die Regression der ECTS, anstelle einer Klassifikation nach *prüfungsaktiv* oder *prüfungsinaktiv* wird deshalb gewählt, weil man im darauffolgenden Studienjahr diese geschätzten ECTS als Input für die Schätzung verwenden will.

Um die tatsächliche Schätzung der prüfungsaktiven Studierenden in drei Jahren durchzuführen, werden für die aktuellsten Daten die ECTS jeweils im darauffolgenden Jahr vorhergesagt. Das wird für drei Jahre in der Zukunft durchgeführt. Dabei baut man ab dem zweiten vorhergesagten Jahr bereits auf einer Schätzung auf. Danach kann

man für jeden Eintrag anhand der geschätzten ECTS entscheiden, ob er im Jahr $t = 3$ prüfungsaktiv sein wird oder nicht.

Weil man Vorhersagen aufgrund von bereits geschätzten Daten durchführt, kann es zu einer Fehlerfortpflanzung kommen. Es gilt nun herauszufinden, ob sich diese in Grenzen hält oder ob sich dieser Ansatz als unbrauchbar erweist.

2.1.2 Ansatz 2

Im zweiten Ansatz beachtet man, dass es Eigenschaften gibt, die sich während der gesamten Studienzeit nicht verändern. Alle veränderlichen Merkmale, wie beispielsweise *ECTS im Jahr zuvor*, werden nicht betrachtet. Diese Eigenschaften und deren Ausprägungen sind:

- Geschlecht (männlich, weiblich)
- Schulbesuch (AHS, BHS, andere)
- Herkunft (Steiermark, Österreich ohne Steiermark, Deutschland, Ausland ohne Deutschland)
- Studienrichtung (Rechtswissenschaften, Betriebswirtschaft, Pädagogik)

Somit ergeben sich 72 verschiedene Kombinationen. Man kann nun alle 72 verschiedenen Kombinationen betrachten. Da alle anderen Eigenschaften nicht betrachtet werden, unterscheiden sich die Kombinationen in den ausstehenden Eigenschaften nicht und man kann für jede Kombination gleich vorgehen.

Es wird von zeitlich veränderbaren Zuständen $Z^{(t)}$ ausgegangen, wobei jeder dieser Zustände einer Menge von Studierenden entspricht. Anstatt eine Funktion von $Z^{(t)}$ nach $Z_{alt}^{(t+1)}$ zu verwenden, ist hier der Ansatz, einen stochastischen Prozess $X = (X_r)_{r\in\{0,1,\ldots,k\}}$ zu definieren, welcher einzelne Studierende über die Zeit ihres Studiums $r \in \{0, 1, \ldots, k\}$ beschreibt.

Wichtig ist hier die Unterscheidung zwischen den Zuständen $Z^{(t)}$ und dem Prozess $X = (X_r)_{r\in\{0,1,\ldots,k\}}$. $Z^{(t)}$ gibt die Menge der Studierenden zur gewünschten Zeit t ab einem festgelegten Zeitpunkt an. X_r gibt die Zustände der einzelnen Studierenden

im jeweiligen Studienjahr r an, indem sich die studierende Person gerade befindet. Das bedeutet, dass der Prozess X auf der Ebene eines jeden Studierenden abläuft, wohingegen die Zustände $Z^{(t)}$ die aggregierte Menge der gesamten Studierenden im Jahr t beschreibt.

Eine weitere Annahme in diesem Ansatz ist die *Markov Eigenschaft* des Prozesses X_r. Sie besagt, dass der Zustand, in dem sich die studierende Person befindet, die gesamte Information für den weiteren Verlauf der studierenden Person im Prozess X beinhaltet [2, Seite 340].

Der Prozess $X = (X_r)_{r\in\{0,1,\dots,k\}}$ hat ab den Jahren $r \geq 1$ folgende mögliche Zustände:

- **a**: steht für Studierende die zwar prügungsaktiv waren, aber nicht für das kommende Jahr inskribiert sind. Das heißt aufgrund eines Abschlusses, eines Abbruchs oder aufgrund einer Pausierung des Studiums.

- **b**: steht für Studierende die prüfungsinaktiv waren, aber nicht für das kommende Jahr inskribiert sind. Das heißt aufgrund eines Abbruches oder aufgrund einer Pausierung des Studiums.

- **c**: steht für Studierende die prüfungsaktiv waren, und auch für das nächste Jahr inskribiert sind.

- **d**: steht für Studierende die prüfungsinaktiv waren, aber dennoch weiterhin für das nächste Jahr inskribiert sind.

Jede studierende Person muss sich in einem dieser Zustände befinden. Von dort ausgehend gibt es für diese Person gewisse Wahrscheinlichkeiten, in welchem Zustand sie im kommenden Jahr sein wird. Aus diesem Grund müssen alle Übergangswahrscheinlichkeiten $p_r^{(xy)}$ für eine bestimmte Kategorie abgeschätzt werden, wobei $x \in \{c, d\}$ und $y \in \{a, b, c, d\}$ ist. Jeder Studierende dieser Kombination muss sich in einem dieser Zustände befinden und hat dann die angegebenen Übergangswahrscheinlichkeiten für den neuen Zustand im kommenden Studienjahr. Abbildung 2.1 beschreibt den Prozess X grafisch.

Dieser Ansatz profitiert von der Eigenschaft der Problemstellung, dass nur die absolute Häufigkeit an prüfungsaktiven Studierenden auf aggregierter Ebene gefragt ist. Es ist nicht wichtig zu wissen, ob eine Studentin oder ein Student in einem bestimmten Studienjahr prüfungsaktiv war oder nicht. Somit konvergiert die Schätzung der prüfungs-

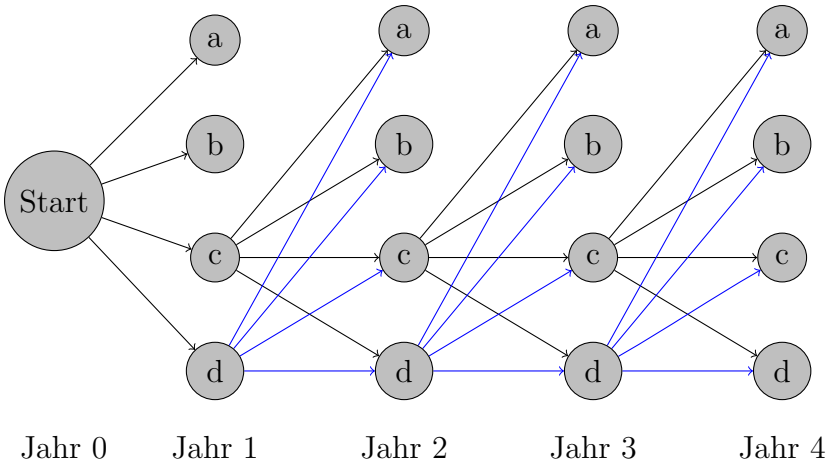

Abbildung 2.1: Der Prozess $X = (X_r)_{r\in\{0,1,\dots,k\}}$ ist hier für den Studienbeginn und die ersten vier Studienjahre X_0, X_1, X_2, X_3, X_4 dargestellt. X_r eines jeweiligen Studierenden im Jahr r kann folgende Werte annehmen: $X_r \in \{a, b, c, d\}$. Die Übergangswahrscheinlichkeiten sind entlang der blauen und schwarzen Linien zu erkennen.

aktiven Studierenden fast sicher gegen die tatsächliche Anzahl der prüfungsaktiven Studierenden, wenn die Anzahl der geschätzen Personen groß wird. Diese Bedingung ist in dieser Problemstellung erfüllt.

2.1.3 Ansatz 3

Im dritten Ansatz wird versucht, passende Übergangswahrscheinlichkeiten zu finden. Man versucht die Wahrscheinlichkeit zu finden, mit der eine studierende Person zu einem gewissen Zeitpunkt t in der Zukunft prüfungsaktiv sein wird. Zum Beispiel wird die Wahrscheinlichkeit gesucht, mit der eine studierende Person *in drei Jahren in der Zukunft* prüfungsaktiv sein wird oder nicht.

In Ansatz 2 wurde jede studierende Person auf vier Eigenschaften beschränkt. In diesem Ansatz sollen mehr Eigenschaften genutzt werden. Das bedeutet, es werden auch die Informationen über absolvierte ECTS aus der Vergangenheit verwendet.

Um dieses Ziel zu erreichen, werden Machine Learning Modelle verwendet, welche grundsätzlich zur Klassifizierung dienen. Diese Klassifizierung wird mithilfe einer berechneten Wahrscheinlichkeit und eines manuell bestimmten Schwellwertes durchgeführt. Da man bei der vorliegenden Problemstellung jedoch keine exakte Klassifizierung benötigt, sondern ausschließlich die Anzahl an prüfungsaktiven Studierenden wissen

will, wird nur die berechnete Wahrscheinlichkeit verwendet, ohne zu klassifizieren. Man summiert für alle Studierenden deren geschätzte Wahrscheinlichkeiten prüfungsaktiv zu sein auf und erhält somit die erwartete Anzahl an prüfungsaktiven Studierenden.

Es ist ein Vorteil neben vielen unterschiedlichen Klassen, welche durch diskrete Eigenschaften entstehen (analog zu Ansatz 2), auch kontinuierliche Datenpunkte von Studierenden verwenden zu können. Somit werden beispielsweise auch Eigenschaften wie *„kumulierte ECTS“* und *„ECTS im Jahr zuvor“* verwendet. Damit wird erreicht, dass möglichst viel Information verwendet wird, was bei einer reinen Wahrscheinlichkeitsberechnung wie in Ansatz 2 nicht möglich ist.

Es wird eine Funktion $F_t(\cdot)$ gesucht, welche für alle Studierenden eine Wahrscheinlichkeit p_t ausgibt, mit der sie im Jahr t prüfungsaktiv sein werden oder nicht. Es braucht für unterschiedliche Zeitspannen mehrere Funktionen, die in ihrem Aufbau gleich sind. Das bedeutet, man muss für jede Zeitspanne von t Jahren, die unterschiedlich lang sein kann, eine neue Funktion trainieren. Die Vorhersagefunktionen sind wie folgt aufgebaut:

$$F_t(S_i) = \begin{cases} h_1^{(t)}(S_i), & \text{für } E_i^{(1)} = 1 \\ h_2^{(t)}(S_i), & \text{für } E_i^{(1)} \geq 2 \end{cases} \in (0, 1)$$

Die Unterscheidung in h_1 und h_2 ist wieder notwendig, da man im ersten Studienjahr noch keine Informationen über ECTS in den vorangegangen Jahren hat. Weiters ist der Output der Funktion eine Wahrscheinlichkeit $p_t \in (0, 1)$.

Es werden folgende Machine Learning Modelle in diesem Ansatz verwendet:

- Logistische Regression
- Support Vector Machine Modelle
- Random Forest Modelle
- Künstliche Neuronale Netzwerke

Es wird anschließend das Modell ausgewählt, welches nach einer unten beschriebenen Metrik, die Anzahl der prüfungsaktiven Studierenden im Jahr $t = 3$, am besten vorhersagen kann.

Um eine Vorhersage in der Praxis durchzuführen, werden die aktuellsten Daten verwendet und für jede studierende Person wird die Wahrscheinlichkeit geschätzt, mit der sie im Jahr $t = 3$ prüfungsaktiv sein wird. Danach werden alle geschätzten Wahrscheinlichkeiten summiert und man erhält den Erwartungswert an prüfungsaktiven Studierenden in drei Jahren.

2.2 Ansätze für Problem 2

P2 stellt die Schätzung der prüfungsaktiven Studierenden von zukünftigen Studienbeginnern in drei Jahren in der Zukunft dar. Von diesen Studierenden kennt man weder Anzahl noch Merkmalskombinationen. Es handelt sich jedoch immer um Neuinskripenten, die hinzukommen.

2.2.1 Ansatz 1

Im ersten Ansatz wird versucht die Zahl aller neu inskribierenden Personen in den folgenden zwei Kalenderjahren zu schätzen. Hierzu wird probiert, mittels einer Regression aus Daten von vergangen Jahren, den Trend der Anzahl von neu inskribierenden Personen fortzusetzen. Diese Schätzung der Anzahl an neu hinzukommenden Studierenden beinhaltet eine gewisse Unsicherheit, da die Anzahl an neu inskribierenden Personen von vielen unterschiedlichen Faktoren abhängig sein kann, von denen man aber keine Informationen zur Verfügung hat.

Wenn man einen Wert für die Anzahl der kommenden Studienbeginnerinnen und Studienbeginner geschätzt hat, wird für ihre Merkmalskombination angenommen, dass diese **gleich** mit jenen Merkmalskombinationen der neu inskribierenden Personen aus dem letzten gegebenen Jahr ist. Das bedeutet, man wählt eine Stichprobe mit Zurücklegen der Größe der geschätzten Anzahl aus, welche aus den Merkmalen von Studienbeginner:innen aus dem letzten Jahr besteht, von denen man die Daten noch zur Verfügung hat.

Nachdem man mit der geschätzten Anzahl und den angenommen Merkmalskombinationen neue fiktive Studienbeginner:innen für die kommenden beiden Jahre erstellt hat, kann man schätzen, ob sie in einer gewissen Zeitspanne in der Zukunft prüfungsaktiv sein werden oder nicht. Diese Schätzung wird mit jener Methode durchgeführt, die sich

für P1 als erfolgreich erwiesen hat.

2.2.2 Ansatz 2

Im zweiten Ansatz werden die Studierenden nach folgenden unveränderbaren Merkmalen eingeteilt:

- Geschlecht
- Herkunft
- besuchter Schultyp
- Studienrichtung

Dadurch ergeben sich 72 Kombinationen. Nun wird für jede dieser Kombinationen die Anzahl an zukünftigen neu inskribierenden Personen mittels einer Regression aus Daten von vergangenen Jahren versucht vorherzusagen. Das bedeutet, dass man in diesem Ansatz den Verlauf der Anzahl an neu inskribierenden Personen in jeder dieser Klassen versucht zu berücksichtigen. Somit können mögliche Informationen von Klassen, die sich anders entwickeln als alle Klassen gemeinsam, beachtet werden. Jedoch ist die Schätzung der Anzahl an neu inskribierenden Personen aller Klassen mit Unsicherheit behaftet, da diese Zahl von vielen unterschiedlichen Faktoren abhängig sein kann, von denen man keine Informationen besitzt.

Nachdem man die Anzahl für alle Klassen geschätzt hat, kann man eine Menge von fiktiven neu inskribierenden Personen bilden. Diese haben eine Merkmalskombination, welche aus vier Merkmalen besteht. Nun kann man für jede fiktive Person schätzen, ob sie in einer Zeitspanne in der Zukunft prüfungsaktiv sein wird oder nicht. Diese Vorhersage wird mit der Methode durchgeführt, welche sich für P1 als erfolgreich erwiesen hat.

2.3 Machine Learning Modelle

In P1 und P2 ist das Ziel, von Daten aus der Vergangenheit Informationen zu gewinnen und diese für Vorhersagen in die Zunkunft zu verwenden. Man versucht gewisse Muster in den Daten der Studierenden zu erkennen und anhand dieser eine Aussage über deren Prüfungsaktivität in den kommenden Jahren zu treffen. Diese Aufgabe ist für Menschen aufgrund der großen Anzahl an Daten schwer bewältigbar. Aus diesem Grund werden Machine Learning Modelle verwendet, um dieses Ziel zu erreichen.

In dieser Arbeit werden ausschließlich Machine Learning Modelle verwendet, die man als *supervised learning* bezeichnet. Das bedeuted, man verfügt über Daten der Form $(\mathbf{x}_i, y_i)$ mit $i = 1, \ldots, n$. Dabei stellt $\mathbf{x}_i \in \mathbb{R}^d$ den Eigenschaftsvektor der Inputdaten und $y_i \in \mathbb{R}$ den bereits vorhandenen, tatsächlichen Zielwert (oder Output) dar. Die Anzahl der Daten wird mit n beschrieben. Alle Daten mit denen die Schätzfunktion optimal gebildet werden soll, werden als *Trainingsdaten* bezeichnet. Anhand der bekannten Outputs zu den jeweiligen Inputs kann das Modell durch einen Trainingsalgorithmus mit jedem Beispiel verbessert werden. Man spricht dabei vom *Training* des Modells [20, Seiten 19 bis 25].

Das Ziel eines jeden Machine Learning Modells ist es, eine passende Regel zu erkennen, die nicht nur möglichst viele Trainigsdaten richtig abbilden kann, sondern vor allem gut auf neue, unbekannte Daten generalisiert [21, Seite 371].

Die Inputdaten in der vorliegenden Problemstellung sind die Eigenschaftsvektoren der Studierenden in dem jeweiligen Studienjahr. Die Outputklassen oder Outputwerte sind entweder die Klassifizierung *prüfungsaktiv*, *nicht prüfungsaktiv* oder die erreichten *ECTS pro Jahr*. In den folgenden Absätzen wird Machine Learning genauer erklärt und formal beschrieben.

Wir bezeichnen $\mathcal{X}$ als Menge der Inputdaten und $\mathcal{Y}$ als Menge der Outputdaten. Es wird eine unbekannte Verteilung $\mathcal{D}_{\mathbf{X}}$ über $\mathcal{X}$ und zusätzlich eine Funktion $f(\cdot)$ angenommen mit der $Y = f(\mathbf{X}) + \epsilon$ gilt, wobei ϵ als *Rauschen* bezeichnet wird. Dabei wird ϵ für jeden Datenpunkt als unabhängig und identisch verteilt angenommen mit $\mathbb{E}[\epsilon] = 0$, $\text{Var}(\epsilon) = \sigma^2 \geq 0$. Um Trainingsdaten zu bekommen, wird eine zufällige Stichprobe $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ mit n Datenpunkten gebildet. Es werden alle $\mathbf{x}_i$ aus der Verteilung $\mathcal{D}_{\mathbf{X}}$ gezogen und es wird angenommen, dass $y_i = f(\mathbf{x}_i) + \epsilon_i$ gilt.

Wenn $\mathbf{X}$ eine Zufallsvariable mit Verteilung $\mathcal{D}_{\mathbf{X}}$ ist und ϵ eine Zufallsvariable ist, dann

wird durch $Y = f(\mathbf{X}) + \epsilon$ eine weitere Zufallsvariable definiert. Wir nehmen also an, dass S eine n-elementige Stichprobe der Verteilung von $(\mathbf{X}, Y)$ ist. Wenn Y eine diskrete Verteilung hat, spricht man von einem Klassifizierungsproblem. Ansonsten, wenn Y eine stetige Verteilung besitzt, nennt man es Regressionsproblem. Die gemeinsame Verteilung von $\mathbf{X}$ und Y nennen wir $\mathcal{D}$.

Die folgenden Definitionen sind angelehnt an Shalev (2014) [20, Seiten 33 bis 35]. Ausgehend davon soll eine Vorhersagefunktion $h(\cdot; \mathbf{w})$ innerhalb einer parametrisierten Funktionenklasse $\mathcal{H} = \{h(\cdot; \mathbf{w}) | \mathbf{w} \in \mathbf{W}\}$ gefunden werden. h_S hängt zum einen von der Wahl der Funktionenklasse und zum anderen von der Auswahlmethode aus dieser Klasse ab. Diese Faktoren werden durch einen *Algorithmus* $\mathcal{A}$ zusammengefasst. Damit definiert man $h_S = \mathcal{A}(S)$, wobei die Menge S eine Stichprobe der Verteilung $\mathcal{D}$ mit der Größe n darstellt.

Um eine Vorhersagefunktion h_S zu finden, welche die wahre Funktion f möglichst gut annähert, muss definiert werden, was es bedeutet, dass h_S *nahe* an f liegt. Um zu beschreiben, wie nahe h_S an f liegt, wird die *loss*-Funktion ℓ mit

$$\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_+$$

definiert. $\mathcal{H}$ entspricht der Funktionklasse, aus der h_S gewählt werden soll und $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ der Menge aller möglichen Daten. Mithilfe von ℓ lässt sich die *wahre risk*-Funktion $L_\mathcal{D}$ mit

$$L_\mathcal{D}(\mathcal{A}) = \mathbb{E}[\ell(\mathcal{A}(S), (X, Y))]$$

definieren, wobei mit $S \backsim \mathcal{D}^{\otimes n}$, $\mathcal{A}(S) \in \mathcal{H}$ und $(\mathbf{X}, Y) \in \mathcal{Z}$. Jeder Machine Learning Algorithmus versucht diese Funktion so klein wie möglich zu halten.

Da für die Bildung von h_S nur die endliche Menge $S \in \mathcal{Z}^n$ an Trainingsdaten zur Verfügung steht, wird die *empirische risk*-Funktion L_S mit

$$L_S(h_S) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_S, (\mathbf{x}_i, y_i))$$

gebildet. Oft ist ein Algorithmus dahingehend gebildet worden, dass er das empirische Risiko minimiert mit der Hoffnung, dass damit auch das wahre Risiko klein wird.

Es hängt nun vom Algorithmus ab, wie gut die wahre risk-Funktion approximiert werden kann. Je nach Wahl der Funktionenklasse und des Auswahlverfahrens kann man ein Vorwissen über die Problemstellung miteinfließen lassen. Wenn Einschränkungen für die Funktionenklasse aufgrund des Vorwissens vorgenommen werden, kann ein sys-

tematischer Approximationsfehler, auch *Bias* genannt, entstehen.

Einerseits kann man eine sehr allgemeine Funktionenklasse auswählen, wodurch $\mathcal{A}(S)$ alle Daten innerhalb von S korrekt reproduziert. Somit hängt die Bildung von h_S stark von der Stichprobe S ab, und h_{S_1} und h_{S_2}, wobei $S_1 und S_2$ disjunkt sind, liefern unterschiedliche Werte für Inputdaten, die in keiner der beiden Stichproben enthalten sind. Zwar kann der Wert der empirischen risk-Funktion klein gehalten werden, aber dennoch liefert die gebildete Vorhersagefunktion große Werte für die wahre risk-Funktion. Das ist ein Beispiel dafür, dass es mehr benötigt, um eine gute Vorhersagefunktion zu bilden, als die empirische risk-Funktion zu minimieren.

Andererseits kann man die Funktionenklasse zu sehr beschränken, sodass der Bias groß ist und dadurch auch die empirische risk-Funktion für die Vorhersagefunktion hohe Werte liefert. Es können zwei Fälle auftreten [21, Seite 374]:

- Falls für h_S das empirische Risiko gering ist, jedoch hohe Werte für das wahre Risiko erzielt werden, spricht man von *Overfitting*.

- Wenn für h_S selbst das empirische Risiko (und somit auch das wahre Risiko) hoch ist, spricht man von *Underfitting*.

Um Overfitting und Underfitting und eine mögliche Herangehensweise daran näher zu beschreiben, beschränkt sich die Argumentation ab hier auf Regressionsprobleme und somit gilt $y \in \mathbb{R}$. Es wird zuerst der Zusammenhang zwischen der loss-Funktion und der Verteilung $\mathcal{D}_{Y|\mathbf{X}}$ beschrieben. Danach wird das wahre Risiko umgeformt, um es intuitiver verstehen zu können. Wenn man anschließend für einen Algorithmus das wahre Risiko berechnen will, kann man durch diese Umformung zwischen Overfitting und Underfitting unterscheiden.

Man legt mit der Wahl von ϵ eine Klasse von Verteilungen fest, deren Elemente durch Parameter, wie beispielsweise dem Erwartungswert oder dem Median, beschrieben werden. ϵ soll entsprechend der realen Problemstellung angenommen werden. Beispielsweise hat man bei Messprozessen ein Vorwissen über die Verteilung der Messfehler. Die Vorhersagefunktion h_S approximiert schlussendlich einen der Parameter von $\mathcal{D}_{Y|\mathbf{X}}$. Welchen Parameter der bedingten Verteilung die Vorhersagefunktion approximieren soll, hängt mit der angenommenen Form von ϵ zusammen. Es ist sinnvoll, dass bei normalverteiltem Rauschen h_S den bedingten Erwartungswert von $\mathcal{D}_{Y|\mathbf{X}}$ approximiert, weil die Normalverteilung mit dem Erwartungswert parametrisiert wird. Andererseits soll h_S bei laplaceverteiltem Rauschen den bedingten Median von $\mathcal{D}_{Y|\mathbf{X}}$ approximieren,

weil die Laplaceverteilung mit dem Median parametrisiert wird.

Es ist wichtig hervorzuheben, dass die Wahl des entsprechenden Parameters auch mit der Wahl der loss-Funktion zusammenhängt. Im Folgenden wird gezeigt, welche loss-Funktionen für den bedingten Erwartungswert und bedingten Median gewählt werden müssen.

Wenn für eine Zufallsvariable U mit einer stetigen Dichte $f_U(u) > 0$ und einem endlichen Erwartungswert die Funktion $g(\phi) = \mathbb{E}[r(U - \phi)]$ minimiert wird, kann man mit der Wahl von $r(\cdot)$ festlegen, welchen Wert das Minimum annimmt. Es wird nun gezeigt, dass (1) bei der Wahl von $r(U - \phi) = (U - \phi)^2$ das Minimum von $g(\phi)$ an der Stelle $\phi = \mathbb{E}[U]$ angenommen wird. Danach wird gezeigt, dass (2) bei der Wahl von $r(U - \phi) = |U - \phi|$ das Minimum von $g(\phi)$ an der Stelle $\phi = m(u)$ angenommen wird, wobei $m(u)$ der Median von U ist und durch $\int_{-\infty}^{m(u)} f_U(u)\, du = \frac{1}{2}$ definiert ist.

Beweis zu (1):
Um die Funktion $g(\phi) = \mathbb{E}[(U-\phi)^2]$ zu minimieren betrachten wir ihre erste und zweite Ableitung.

$$g'(\phi) = \frac{d}{d\phi}\mathbb{E}[(U - \phi)^2] = \mathbb{E}[-2(U - \phi)] = -2\mathbb{E}[U] + 2\phi$$

Hier dürfen Differenzierung und Erwartungswert vertauscht werden, weil alle Bedingungen des Satzes *Differentiation unter dem Integralzeichen* nach Elstrodt (1996) erfüllt sind [10, Kapitel 4, Satz 5.7]. Der oben berechnete Ausdruck wird an der Stelle $\phi = \mathbb{E}[U]$ null und ist dadurch ein kritischer Punkt. Darüber hinaus ist die zweite Ableitung nach ϕ größer null:

$$g''(\phi) = \frac{d}{d\phi}(-2\mathbb{E}[U] + 2\phi) = 2 > 0$$

Somit ist die gefundenw Stelle ein Minimum von $g(\phi)$. □

Beweis zu (2):
Zuerst setzen wir für den Ausdruck $g(\phi) = \mathbb{E}[|U - \phi|]$ die Definition des Erwartungs-

wertes ein und formen diese um.

$$\begin{aligned}\mathbb{E}[|U-\phi|] &= -\int_{-\infty}^{\phi}(u-\phi)f_U(u)\,du + \int_{\phi}^{\infty}(u-\phi)f_U(u)\,du \\ &= -\int_{-\infty}^{\phi} uf_U(u)\,du + \phi\int_{-\infty}^{\phi} f_U(u)\,du + \int_{\phi}^{\infty} uf_U(u)\,du - \phi\int_{\phi}^{\infty} f_U(u)\,du \\ &= -\int_{-\infty}^{\phi} uf_U(u)\,du + \phi\int_{-\infty}^{\phi} f_U(u)\,du + \mathbb{E}[U] \\ &\quad -\int_{-\infty}^{\phi} uf_U(u)\,du - \phi\Big(1-\int_{-\infty}^{\phi} f_U(u)\,du\Big)\end{aligned}$$

Nun leiten wir diesen Ausdruck nach ϕ ab und setzen ihn gleich null:

$$\begin{aligned}g'(\phi) = \frac{d}{d\phi}g(\phi) &= -\phi f_U(\phi) + \int_{-\infty}^{\phi} f_U(u)\,du + \phi f_U(\phi) \\ &\quad - \phi f_U(\phi) - 1 + \phi f_U(\phi) + \int_{-\infty}^{\phi} f_U(u)\,du \\ &= 2\int_{-\infty}^{\phi} f_U(u)\,du - 1 \stackrel{!}{=} 0 \\ &\Leftrightarrow \int_{-\infty}^{\phi} f_U(u)\,du = \frac{1}{2}\end{aligned}$$

Dieser Ausdruck entspricht der Definition des Medians von U und einem kritischen Punkt von $\phi \mapsto \mathbb{E}[|U-\phi|]$. Weil die zweite Ableitung

$$g''(\phi) = f_U(\phi) > 0$$

die Dichte von U ist, die immer positiv ist, handelt es sich um ein Minimum von $g(\phi)$. □

Im folgenden Abschnitt versucht man das wahre Risiko besser zu verstehen. Es wird der Argumentation von Bishop (2006) gefolgt [3, Seiten 147 bis 152]. Dafür ist es notwendig einerseits den *erwarteten Output* $\bar{y}$ zu definieren:

$$\bar{y}(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X}=\mathbf{x}] = \int_y y f_{Y|\mathbf{X}}(y|\mathbf{x})\,dy.$$

Andererseits benötigt man die *erwartete Vorhersagefunktion* $\bar{h}$, welche, vorausgesetzt eines Algorithmus mit:

$$\begin{aligned}\bar{h} = \mathbb{E}[\mathcal{A}(S)] &= \int_{\mathbb{R}^{(d+1)n}} h_s p_S(s)\,ds \\ &= \int_{\mathcal{Z}^n} \mathcal{A}(\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_n,y_n)\}) \prod_{k=1}^{n} f_{\mathbf{X},Y}(\mathbf{x}_k,y_k)\,d\mathbf{x}_1 dy_1 \ldots d\mathbf{x}_n dy_n\end{aligned}$$

gegeben ist.

Den erwarteten Output kann man verstehen als Erwartungswert aller möglichen y, gegeben ein festes $\mathbf{x}$. Um die erwartete Vorhersagefunktion zu bilden, geht man davon aus, dass die Funktion h_S von der zufälligen Stichprobe S abhängig ist. Man kann sich vorstellen, dass man unendlich viele Stichproben S zieht und für jedes S bekommt man eine andere Funktion h_S. Danach wird auf ein $\mathbf{x}$ jede dieser unterschiedlichen Funktionen h_S angewendet und der Funktionswert wird anschließend gemittelt.

Ab hier beschränken wir uns auf die loss-Funktion $\ell(h, (\mathbf{x}, y)) = (h(\mathbf{x}) - y)^2$, weil dadurch die Argumentation vereinfacht wird. Das bedeutet, dass h_S den bedingten Erwartungswert von Y gegeben $\mathbf{X}$ approximiert, wie oben gezeigt wurde. Nun wollen wir die wahre risk-Funktion von $\mathcal{A}$ betrachten:

$$\mathcal{L}_{\mathcal{D}}(\mathcal{A}) = \mathbb{E}[(h_S(\mathbf{X}) - Y)^2)],$$

wobei gilt, dass $(\mathbf{X}, Y) \backsim \mathcal{D}$ und $S \backsim \mathcal{D}^{\otimes n}$. Das Ziel ist es, diesen erwarteten Fehler so umzuformen, dass wir ihn in Teile zerlegen können, die verständlicher sind.

$$\begin{aligned}
&\mathbb{E}[(h_S(\mathbf{X}) - Y)^2] \\
&= \mathbb{E}[\Big((h_S(\mathbf{X}) - \bar{h}(\mathbf{X})) + (\bar{h}(\mathbf{X}) - Y)\Big)^2] \\
&= \mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))^2] + \mathbb{E}[(\bar{h}(\mathbf{X}) - Y)^2] + 2\underbrace{\mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))(\bar{h}(\mathbf{X}) - Y)]}_{=0} \\
&= \mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))^2] + \mathbb{E}[(\bar{h}(\mathbf{X}) - Y)^2] \\
&= \mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))^2] + \mathbb{E}[\Big((\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X})) + (\bar{y}(\mathbf{X}) - Y)\Big)^2] \\
&= \mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))^2] + \mathbb{E}[(\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X}))^2] + \mathbb{E}[(\bar{y}(\mathbf{X}) - Y)^2] \\
&+ 2\underbrace{\mathbb{E}[(\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X}))(\bar{y}(\mathbf{X}) - y)]}_{=0} \\
&= \underbrace{\mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X}))^2]}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\bar{y}(\mathbf{X}) - Y)^2]}_{\text{Noise}}
\end{aligned}$$

Der letzte Summand in Zeile 3 ist null, weil:

$$\begin{aligned}
&\mathbb{E}[(h_S(\mathbf{X}) - \bar{h}(\mathbf{X}))(\bar{h}(\mathbf{X}) - Y)] \\
&= \mathbb{E}[\mathbb{E}[h_S(\mathbf{X}) - \bar{h}(\mathbf{X})](\bar{h}(\mathbf{X}) - Y)] \\
&= \mathbb{E}[(\mathbb{E}[h_S(\mathbf{X})] - \bar{h}(\mathbf{x}))(\bar{h}(\mathbf{x}) - Y)] \\
&= \mathbb{E}[(\bar{h}(\mathbf{X}) - \bar{h}(\mathbf{X}))(\bar{h}(\mathbf{X}) - Y)] \\
&= \mathbb{E}[0] \\
&= 0
\end{aligned}$$

Weiters ist der letzte Summand in Zeile 7 null, weil:

$$
\begin{aligned}
&\mathbb{E}_{X,Y}[(\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X}))(\bar{y}(\mathbf{X}) - Y)] \\
&= \mathbb{E}[\mathbb{E}[\bar{y}(\mathbf{X}) - Y|\mathbf{X} = \mathbf{x}](\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X}))] \\
&= \mathbb{E}[(\bar{y}(\mathbf{X}) - \mathbb{E}[y|\mathbf{X} = \mathbf{x}])(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))] \\
&= \mathbb{E}[(\bar{y}(\mathbf{X}) - \bar{y}(\mathbf{X}))(\bar{h}(\mathbf{X}) - \bar{y}(\mathbf{X}))]] \\
&= \mathbb{E}[0] \\
&= 0
\end{aligned}
$$

Diese drei verbliebenen Terme sind:

- Die Varianz der Vorhersagefunktion an sich. Je nach Ziehung der Stichprobe S kann h_S unterschiedlich sein.

- Der Bias^2 ist die systematische Abweichung der Vorhersagefunktion.

- Der Noise gibt an, wie schwer die Aufgabe an sich ist. Er gibt die Varianz von ϵ an. Der Noise ist die untere Grenze des wahren Risikos.

Somit besagt die Gleichung:

$$\text{Risk} = \text{Varianz} + \text{Bias}^2 + \text{Noise}$$

und das wahre Risiko kann in drei verständliche Teile eingeteilt werden. Da alle drei Teile keine linearen Funktionen sind, versucht man die *Hyperparameter* so zu wählen, dass das wahre Risiko so klein wie möglich wird. Das ist jener Bereich an Komplexität der Funktionenklasse, der zwischen Overfitting und Underfitting liegt.

Bei Machine Learning Algorithmen und Modellen versucht man neben der Optimierung der Parameter des entsprechenden Modells, auch die Hyperparameter der Modellarchitektur zu optimieren. Der entscheidende Unterschied zwischen Hyperparametern und Parametern ist, dass die Hyperparameter vor dem Training des Modells festgelegt werde müssen und die Parameter während des Trainings optimiert werden [22].

Formal legen Hyperparameter eine Klasse von Funktionen, *innerhalb* der zuvor festgelegten Funktionenklasse $\mathcal{H}$ eines Machine Learning Algorithmus, fest. Beispielsweise wird festgelegt, ob man bei einer linearen Regression eine Linearkombination der Inputvariablen, oder auch deren Quadrate zulässt. Diese Entscheidung muss vor dem Training getroffen werden.

Sowohl bei einer Regressions- als auch bei einer Klassifikationsaufgabe ist das Ziel der jeweiligen Prediktorfunktion h_S eines Machine Learning Algorithmus, ein Muster in den Daten zu finden, welches für Menschen nicht erkennbar wäre. Oft ist es für Menschen schwierig die Vorgehensweise des trainierten Algorithmus nachzuvollziehen.

Nun werden jene Machine Learning Algorithmen kurz beschrieben, welche in der vorliegenden Problemstellung zum Einsatz gekommen sind.

2.3.1 Multiple Lineare Regression

Das folgende Kapitel folgt der Argumentation von Bishop, 2006 [3, Kapitel 3.1]. Hier wird versucht die bedingte Verteilung von $\mathcal{D}_{Y|\mathbf{X}}$ mit einer bestimmten Klasse an Vorhersagefunktionen bestmöglich zu beschreiben. Wiederum folgen $\mathbf{X}$ und Y einer gemeinsamen Verteilung $\mathcal{D}$, wobei man Y mit $Y = f(\mathbf{X}) + \epsilon$ darstellen kann. Ab hier beschränken wir uns darauf, dass $\mathbf{X}$ nur eindimensional ist, um die Notation zu vereinfachen.

Es gibt viele Möglichkeiten die Verteilung von Y zu beschreiben. Die einfachste Wahl wäre es, $\mathbb{E}[Y]$ zu verwenden. Nur so verwendet man nicht die Information, die man durch eine Realisierung von X erhält. Eine bessere Lösung ist es, den bedingten Erwartungswert $\mathbb{E}[Y|X]$ zu verwenden. Somit benützt man auch die Information aus der Realisierung von X. Man kann beispielsweise auch, wie oben gezeigt, den bedingten Median verwenden. Diese Entscheidung sollte von der vorliegenden Problemstellung abhängen. Die Funktion

$$x \mapsto f(x) := \mathbb{E}[Y|X = x] = \int y \cdot p_{Y|X}(y|x)\, dy$$

wird als Regressionsfunktion bezeichnet [25, Seite 209]. Im Folgenden wird nun versucht, die Funktion f möglichst gut zu schätzen.

Nun gibt es viele Möglichkeiten, die Funktionklasse für den Prediktor h_S zu wählen. Bei der linearen Regression nimmt man an, dass die Funktion linear in wenigen Parametern ist. Beispiele sind:

$$h_S(x) = a + bx$$

oder

$$h_S(x) = a + bx + cx^2.$$

Beide gehören zur linearen Regression. Man sieht, dass die Funktion eine Linearkom-

bination von sogenannten *Basis Functions* ist, und somit linear in ihren Parametern ist. Von nun an wird die Argumentation am einfachsten Beispiel von $h_S(x) = a + bx$ fortgeführt.

Die Klasse von Funktionen, aus der ein passender Schätzer h_S gefunden werden kann, wird durch diese Annahme sehr eingeschränkt. Das führt zu einer Reduktion der Varianz, was möglichem Overfitting entgegenwirkt. Es kann aber sein, dass die Funktion aus dieser Klasse für die Problemstellung zu simpel ist und man dadurch die wahre Funktion f nicht genau genug beschreiben kann.

Nun kann man einen Schätzer für den bedingten Erwartungswert bilden. Weil der Erwartungswert von X die Funktion $g(\phi) = \mathbb{E}[(X - \phi)^2]$ minimiert, kann man den *Least Squares Schätzer* wie folgt definieren:

$$(\hat{a}, \hat{b}) = \underset{(a,b)\in\mathbb{R}^2}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - a - bX_i)^2$$

wobei $S = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ ist.

Wie oben gezeigt, berechnet man mit dem Least Squares Schätzer den bedingten Erwartungswert von Y, gegeben X. Wenn man die Summe der absoluten Abstände minimiert, bekommt man einen Schätzer für den bedingten Median. Man verwendet die Tatsache, dass der Median von $Y|X$ die Funktion $g(\phi) = \mathbb{E}[|Y|X - \phi|]$ minimiert. Hierfür müsste aber die Regressionsfunktion anders definiert werden.

Es ist hilfreich, dass es Lösungen in geschlossener Form gibt, die $\sum_{i=1}^{n}(Y_i - a - bX_i)^2$ minimieren. Somit kann man die Schätzer leicht berechnen. Weiters ist es, wie oben erwähnt, möglich, die abhängige Variable Y auch von mehreren unabhängigen Variablen beschreiben zu lassen, indem $\mathbf{X}$ einen Zufallsvektor beschreibt. Dadurch erhält man die *Multiple Lineare Regression* mit der Form

$$Y = \mathbf{X}^T\beta + c.$$

Die Vorgehensweise, um einen Schätzer zu finden, ist analog zu der im eindimensionalen Fall, indem man die Rechenregeln für Vektoren beachtet.

Im Falle des mehrdimensionalen Modells ist die Annahme, dass die „abhängige“ Variable Y eine Linearkombination der „unabhängigen“ Variablen (Einträge von $\mathbf{X}$), oder der *Basis Functions* von ihnen ist. Diese Annahme ist sehr einschränkend, da man im Vorhinein wenig über die gegenseitige Beeinflussung der „unabängigen“ Variablen

untereinander aussagen kann.

Die Lösungen für den eindimensionalen Fall sind [25, Kapitel 13, Satz 13.4]:

$$\hat{b} = \frac{\overline{XY} - \bar{X}\bar{Y}}{\overline{X^2} - (\bar{X})^2}$$

$$\hat{a} = \bar{Y} - \hat{b}\bar{X}$$

mit $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$, $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$, $\overline{X^2} = \frac{1}{n}\sum_{i=1}^{n} X_i^2$ und $\overline{XY} = \frac{1}{n}\sum_{i=1}^{n} X_i Y_i$.

Die Lösungen für den mehrdimensionalen Fall sind [25, Kapitel 13, Satz 13.13]:

$$\hat{\beta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbf{Y}.$$

Wobei $\mathbb{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$ ist. Man kann die letzte Gleichung wie folgt umformen:

$$\mathbb{X}\hat{\beta} = P\mathbf{Y}, P = \mathbb{X}(\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T$$

Somit erhält man eine geometrische Interpretation der Regression. $\mathbb{X}\hat{\beta}$ stellt die Orthogonalprojektion des Punktes $\mathbf{Y}$ auf jener Hyperebene dar, die von den unterschiedlichen Dimensionen in $\mathbb{X}$ aufgespannt wird.

Bei der multiplen linearen Regression ist die Hyperparameterauswahl von entscheidender Bedeutung. Es muss überlegt werden, welche *Basis Functions* auswählt werden sollen.

2.3.2 Logistische Regression

Bei der logistischen Regression wird angenommen, dass $Y \in \{0, 1\}$ und somit Bernoulli verteilt ist. Dadurch ist sie ein Klassifizierungsmodell. ($\mathbf{X}$ kann auch vektorwertig sein.)

Somit ist auch Y gegeben $\mathbf{X}$ eine Bernoulli Zufallsvariable mit ihrem Erwartungswert $\mu(x) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}] \in (0, 1)$. Man kann nicht die Annahme treffen, dass $\mu(\mathbf{x}) = \mathbf{x}^T\beta$ ist, weil diese Funktion nach $\mathbb{R}$ abbilden würde. Bei der logistischen Regression benützt man eine invertierbare Funktion γ mit $\gamma(\mathbf{x}^T\beta) = \mu(\mathbf{x}) \in (0, 1)$.

Somit ist die Funktionenklasse, aus welcher ein Prediktor h_S gewählt wird, nicht mehr

auf $h_S(\mathbf{x}) = \mathbf{x}^T\beta$ beschränkt, sondern wird auf $h_S(\mathbf{x}) = \gamma(\mathbf{x}^T\beta)$ erweitert. Die Umkehrfunktion γ^{-1} wird auch mit $\alpha(\cdot)$ bezeichnet, und *Link Function* genannt [3, Seite 180]. Durch diese Generalisierung muss die Regressionsfunktion nicht zwingend linear in den Basis Functions sein. Nur bevor die sogenannte Link Funktion auf sie angewendet wurde, ist sie linear in ihren Basis Functions.

Im Falle der Bernoulli Verteilung, welche bei der logistischen Regression als Grundannahme gilt, ist die Link Funktion mit:

$$\alpha(x) = \gamma^{-1}(x) = \log(\frac{x}{1-x}) = \log(x) - \log(1-x)$$

gegeben und wird als *logit-link* bezeichnet. Die Funktion $\gamma(x) = \frac{e^x}{1+e^x}$ wird als *logistische Funktion* bezeichnet [25, Seite 223]. Daher kommt auch der Name „logistische Regression“.

Man kann nun das Modell der logistischen Regression wie folgt formulieren: Seien $(\mathbf{X}_i, Y_i) \in \mathbb{R}^m \times \mathbb{R}$, $i = 1, \ldots, n$ unabhängige Paare von Zufallsvariablen, sodass die bedingte Verteilung von Y_i gegeben $\mathbf{X}_i = \mathbf{x}_i$ die Wahrscheinlichkeitsdichtefunktion

$$f_{\mathbf{x}_i^T\beta}(y_i) = \exp(y_i(\mathbf{x}_i^T\beta) - \log(1 + e^{(\mathbf{x}_i^T\beta)})) = \frac{e^{y_i(\mathbf{x}_i^T\beta)}}{1 + e^{(\mathbf{x}_i^T\beta)}}$$

besitzt [25, Seite 223].

Wenn man durch Konstruktion des Maximum Liklihood Schätzers den Parameter β mit $\hat{\beta}$ abschätzen kann, ist es wiederum möglich, mittels der Funktion $\gamma(\mathbf{x}^T\hat{\beta})$ den geschätzten Parameter $\hat{p}$ der Bernoulli Verteilung von Y gegeben $\mathbf{X}$ zu bilden. Nun hat man eine Wahrscheinlichkeit mit der die abhängige Variable Y gegeben $\mathbf{X}$ den Wert 1 annimmt. Um schlussendlich eine Klassifikation durchzuführen, setzt man manuell einen Schwellwert anhand welchem entschieden werden kann, ob die Variable 1 oder 0 geschätzt werden soll.

2.3.3 Random Forest Modelle

Random Forest Modelle sind eine Form des *Ensemble Learnings* und wurden erstmals im Jahr 2001 von Breiman verwendet [6]. Die individuellen Prediktorfunktionen eines Random Forest Modells nennt man *Decision Trees*. Ensemble Learning besteht darin, dass die Outputs mehrerer Prediktorfunktionen zu einem Output zusammengefasst

werden. Der Wert dieses Outputs entspricht dem Wert, den die Mehrheit der individuellen Prediktorfunktionen angenommen hat. Man kann diese Vorgehensweise mit einer Abstimmung beziehungsweise der Mehrheitsmeinung vergleichen [11, Seiten 189 bis 191].

Die Idee dabei ist, dass viele Klassifikatoren, die individuell kaum besser sind als eine Zufallsmeinung, in der Gruppe einen besseren Klassifikator darstellen. Man kann dieses Phänomen mit einer Münze vergleichen, die zu 51% mit Kopf nach oben liegen bleibt. Wirft man sie ein paar Mal, wird man nicht wirklich erkennen können, welche Seite favorisiert wird. Wenn man sie aber sehr oft wirft, bekommt man ein immer stabileres Ergebnis darüber, welche Seite favorisiert wird.

Die Zusammenführung mehrerer Decision Trees beugt auch Overfitting einzelner Prediktorfunktionen vor [20, Seiten 255 und 256]. Weiters werden bei dem verwendeten Random Forest Modell Vorgaben an die einzelnen Decision Trees gegeben. Beispielsweise werden die individuellen Decision Trees nur auf einer Teilmenge der Trainingsdaten trainiert. Hyperparameter für das Random Forest Modell sind die Anzahl der Decision Trees, die Anzahl der Trainigsdaten pro Decision Tree und auch die spezifischen Hyperparameter der Decision Trees an sich. In Abbildung 2.2 ist der schematische Aufbau eines stark vereinfachten Decision Trees dargestellt.

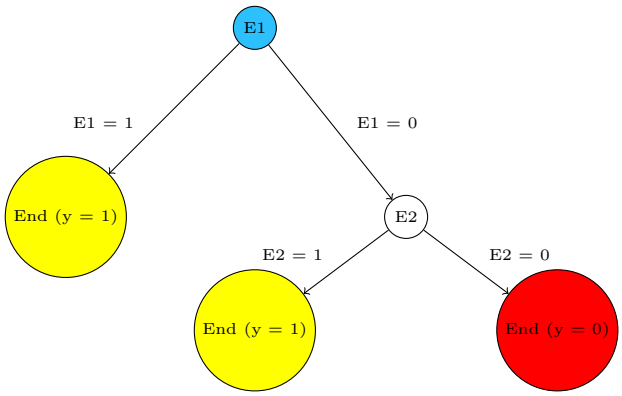

Abbildung 2.2: An jedem Knoten wird nach einem Kriterium bestmöglich entschieden. Sollte eine gewisse Tiefe erreicht sein, oder es nur mehr eine Klasse geben, ist man am *Blatt* des Decision Trees angelangt.

Nun wird das mathematische Modell eines Decision Trees genauer erklärt. Random Forest Modelle beruhen auf der Verwendung mehrerer, verschiedener Decision Trees. Grundsätzlich stellen sich bei einem Decision Tree folgende Fragen [20, Seiten 251 bis 253]:

- In welcher Inputvariable soll unterteilt werden?

- Wo soll die Unterteilung der gewählten Variable stattfinden?

Um diese Fragen zu beantworten wird im Folgenden das verwendete Decision Tree Modell mathematisch beschrieben. Die Argumentation folgt jener der Implementierung von *Scikit-Learn* [18]. Gegeben der Trainingsvektoren $\mathbf{x}_i \in \mathbb{R}^m$, $i = 1, \ldots, n$ und der entsprechenden Zielwerten y_i, geht ein Decision Tree rekursiv vor, um den Raum der Inputvariablen zu unterteilen und nach den Zielwerten zu gruppieren. Die Daten am Entscheidungsknoten v werden mit Q_v bezeichnet, wobei Q_v die Menge von Trainingsdaten $\{(\mathbf{x}_1, y_1, \ldots, (\mathbf{x}_{N_v}, y_{N_v})\}$ darstellt, welche an diesem Entscheidungsknoten noch übrig geblieben sind. Die N_v übrig gebliebenen Daten werden nach jedem möglichen Split $\theta = (j, t_v)$, bestehend aus Inputvariable j und dem Schwellwert t_v in $Q_v^{\text{left}}(\theta)$ und $Q_v^{\text{right}}(\theta)$ unterteilt. Dabei gilt:

$$Q_v^{\text{left}}(\theta) = \{(\mathbf{x}, y) | \mathbf{x}^{(j)} \leq t_v\}$$

$$Q_v^{\text{right}}(\theta) = Q_v \backslash Q_v^{\text{left}}(\theta)$$

Es wird unter allen Kandidaten θ mittels einer loss-Funktion $\ell(\cdot)$, die sich je nach Klassifizierungs- oder Regressionsaufgabe unterscheidet, entschieden, welche Wahl am besten ist:

$$G(Q_v, \theta) = \frac{N_v^{\text{left}}}{N_v} \ell(Q_v^{\text{left}}(\theta)) + \frac{N_v^{\text{right}}}{N_v} \ell(Q_v^{\text{right}}(\theta))$$

Der Kandidat $\theta^* = \text{argmin}_\theta \, G(Q_v, \theta)$ wird ausgewählt. Danach wird rekursiv vorgegegangen bis die maximale Tiefe des Decision Trees, $N_v < \min_{samples}$ oder $N_v = 1$ erreicht wurde. Hierzu werden Greedy-Algorithmen, wie beispielsweise ID3 [19] oder CART [5], verwendet.

Bei Klassifizierungsaufgaben wird für ℓ die *Entropy* verwendet. Bei Regressionsaufgaben wird entweder der *Mean Squared Error* oder der *Mean Absolute Error* verwendet.

Sei $p_{vk} = 1/N_v \sum_{y \in Q_v} \mathbb{1}_{y=k}$ die relative Häufigkeit der Klasse k am Entscheidungsknoten v, $\text{mean}_v(y) = \frac{1}{N_m} \sum_{y \in Q_m} y$ und $\text{median}_v(y) = \text{median}_{y \in Q_v}(y)$, dann sind:

Mean Squared Error:

$$\ell(Q_v) = \frac{1}{N_v} \sum_{y \in Q_v} (y - \text{mean}_v(y))^2$$

Mean Absolute Error:

$$\ell(Q_v) = \frac{1}{N_v} \sum_{y \in Q_v} |y - \underset{v}{\text{median}}(y)|$$

Entropy:

$$\ell(Q_v) = -\sum_k p_{vk} \log(p_{vk})$$

Wichtig ist hervorzuheben, dass es möglich ist, mehrmals nach einer Inputvariablen an unterschiedlichen Punkten aufzuteilen. Dadurch kann man eine Teilung in mehrere Gruppen durch mehrere binäre Teilungen ersetzen. Darüber hinaus ist der Decision Tree mit einer beliebigen Tiefe w in dem Decision Tree der Tiefe $w + 1$ enthalten.

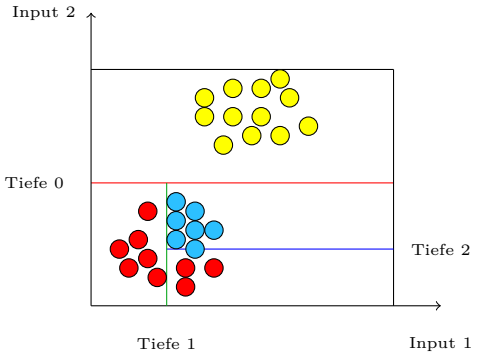

Abbildung 2.3: Darstellung der Entscheidungen eines Decision Trees auf fiktiven Daten (rot, blau, gelb). Die erste Entscheidung betrifft das Merkmal 2 (Input 2) und ist mit der roten Trennlinie dargestellt. Die Entscheidungen zwei und drei sind entsprechend mit der grünen und blauen Trennlinie dargestellt.

In Abbildung 2.3 sieht man, wie eine mögliche Unterteilung des Raumes der Inputvariablen durchgeführt werden kann. Bei einer Klassifikation eines Decision Trees kann man sich auch an jedem Endknoten nicht nur die entsprechende Klasse ausgeben lassen, sondern auch deren Wahrscheinlichkeit. Diese ist die relative Häufigkeit der Outputklasse in der Teilmenge der übriggebliebenen Trainingsbeispiele [18].

2.3.4 Support Vector Machine Modelle

Die Modellarchitektur von Support Vector Machine Modellen wurde von Vladimir Vapnik [4] in den Neunizigerjahren eingeführt. Man kann das Support Vector Machine Modell für Klassifizierungs- und auch für Regressionsprobleme verwenden. In diesem Kapitel wird die Funktionsweise des Klassifizierungsmodells beschrieben. Wir betrachten in diesem Abschnitt nur Klassifikatoren.

Bei diesem Modell wird ausschließlich versucht, die Beispieldaten nach ihren Eigenschaften linear zu separieren. Das bedeutet, man versucht für die Daten

$$S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$$

mit $\mathbf{x_i} \in \mathbb{R}^n$ und $y_i \in \{\pm 1\}$ einen Halbraum $(\mathbf{w}, b)$ zu finden mit

$$y_i = \text{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b), \forall i.$$

Dadurch treten zwei Probleme auf. Erstens gibt es sehr viele Möglichkeiten diesen Halbraum auszuwählen, falls die Daten linear separierbar sind. Zweitens, falls die Daten nicht linear separierbar sind, versucht man das durch die Berechnung weiterer Eigenschaften aus den bisher vorhandenen Eigenschaften zu schaffen, wodurch der Rechenaufwand unbewältigbar werden kann. Die Modellarchitektur der Support Vector Machines versucht diese beiden Probleme zu lösen [20, Kapitel 15].

Die folgende Erläuterung folgt jener von Busuttil [8]. Wenn die Daten linear separierbar sind, gibt es nicht nur eine Hyperebene, die dies erreicht. Um eine einzige zu definieren, die auch auf neuen Daten gut generalisiert, wählt man jene Hyperebene aus, welche die beiden Klassen mit dem größten Margin voneinander trennt. Diese Aufgabe ist ein konvexes Optimierungsproblem mit einer eindeutigen Lösung. Diese Lösung beinhaltet nur eine Teilmenge der Trainigsbeispiele, die an der Grenze zu anderen Klassen liegen. Diese Beispiele werden als *Support Vectors* bezeichnet. Wie man in Abbildung 2.4 sieht, beinhalten die Support Vektoren alle Informationen für eine weitere Klassifizierung und diese verändert sich nicht, wenn man andere Trainingsbeispiele verwendet oder nicht.

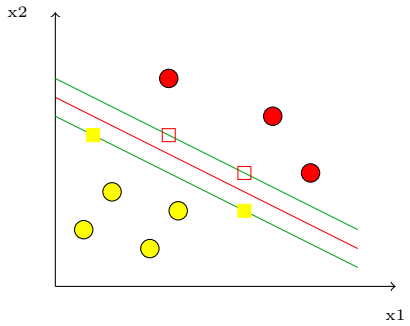

Abbildung 2.4: In dieser Grafik wird dargestellt, dass die Support Vektoren die gesamte Information enthalten.

Es ist möglich, dieses Optimierungsproblem und die Entscheidungsform in *dualer* Form

darzustellen [11, Seite 168]. Die Entscheidungsform hat dadurch folgende Form:

$$f(\mathbf{x}^*) = \text{sign}(\sum_{i=1}^{m} y_i \alpha_i \langle \mathbf{x}^*, \mathbf{x}_i \rangle + b),$$

wobei $\alpha_i \in \mathbb{R}$ als Maß an Information von $\mathbf{x}_i$ gesehen werden kann. Sollte $\mathbf{x}_i$ kein Support Vektor sein, dann gilt $\alpha_i = 0$.

Man unterteilt Support Vector Machine Klassifikatoren in *Hard-Margin* und *Soft-Margin* Klassifikatoren [20, Kapitel 15]. Diese zwei Klassen unterscheiden sich darin, dass Hard-Margin Klassifikatoren eine strenge Unterteilung zwischen $y = \pm 1$ durchführen, wohingegen Soft-Margin Klassifikatoren eine bestimmte Anzahl an Regelverletzungen zulassen. In den implementierten Modellen werden Soft-Margin Modelle verwendet, und der Hyperparameter C gibt an, wie viele Regelverletzungen zugelassen werden. Ein niedriger Wert für C bedeutet mehr Regelverstöße.

Um zu erreichen, dass die Daten einer Problemstellung linear separierbar sind, werden oft neue Eigenschaften berechnet. Das funktioniert durch die Anwendung einer Funktion $\Theta(\cdot)$ auf die vorhandenen Eigenschaften $\mathbf{x}_i$. Das Bild dieser Funktion wird als *Feature Space* (Eigenschaftsraum) bezeichnet. Es gibt mehrere Möglichkeiten, um diese Funktion zu wählen. In Abbildung 2.5 sieht man, dass durch die Anwendung von $\Theta(x) = x^2$ die Daten linear separierbar gemacht wurden.

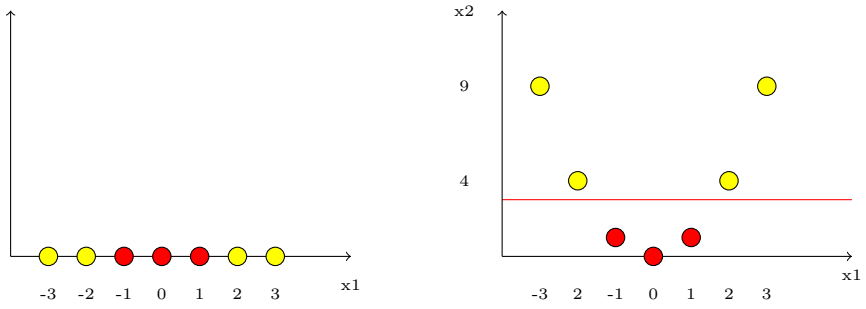

Abbildung 2.5: Darstellung, dass die Daten durch die Berechnung zusätzlicher *Features* linear separierbar wurden.

Die nächsten Absätze folgen Géron (2019) [11, Kapitel 5]. Die Entscheidungsfunktion f benützt das Innere Produkt von Vektoren aus dem Eigenschaftsraum. Die Funktion Θ auf dem Eigenschaftsraum bedeutet, dass man nun $\langle \Theta(\mathbf{x}), \Theta(\mathbf{y}) \rangle$ verwendet. Das kann einen sehr hohen Rechenaufwand mit sich ziehen. Jedoch kann durch die Anwendung des *Kernel Tricks* der Rechenaufwand in Grenzen gehalten werden. Ein Kernel ist eine Funktion der Form $k(\mathbf{x}, \mathbf{y}) = \langle \Theta(\mathbf{x}), \Theta(\mathbf{y}) \rangle$. Laut dem *Mercer Theorem* ist eine Funktion k, die gewisse Eigenschaften (Mercer Bedingungen) erfüllt, ein Kernel zu

einer zugehörigen Funktion $\Theta(\cdot)$. Um diesen Kernel zu verwenden, benötigt man die Funktion $\Theta(\cdot)$ nie und man muss sie nicht einmal kennen.

Es gibt verschiedene Kernels und die Auswahl hängt von der speziellen Aufgabe ab [8]. Im angeführten Beispiel wird $\mathbf{x} = [x_1, x_2]^T$ und $\mathbf{y} = [y_1, y_2]^T$ angenommen. Ein möglicher Kernel ist der Polynomial Kernel mit der Form $k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^d$. Für $d = 2$ gilt:

$$\langle \Theta(\mathbf{x}), \Theta(\mathbf{y}) \rangle = \langle \mathbf{x}, \mathbf{y} \rangle^2 ,$$

wobei $\Theta(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$ ist. Somit ist die Funktion $k(\mathbf{x}, \mathbf{y}) = \langle \Theta(\mathbf{x}), \Theta(\mathbf{y}) \rangle$ ein Polynomial Kernel vom Grad 2. Es ist wichtig hervorzuheben, dass bei der Berechnung nie die Funktion Θ explizit angewendet wurde und somit weniger Rechenaufwand benötigt wird.

2.3.5 Künstliche Neuronale Netzwerke

Wie der Name dieser Modellarchitektur verrät, wird bei diesem Machine Learning Modell versucht, eine Funktion zu bilden, die von der Funktionsweise von biologischen Neuronen in den Gehirnen von Menschen und Tieren inspiriert wird. Die Verbindung von biologischen Neuronen in Form von Synapsen soll durch gewichtete Verbindungen zwischen den künstlichen Neuronen nachgebildet werden. Die Entstehung dieser Algorithmen geht auf McCulloch und Pitts im Jahr 1943 zurück [24].

Künstliche Neuronale Netzwerke (KNN) stellen eine große Klasse an Funktionen dar, aus der eine passende Funktion gefunden werden soll, welche die wahre Funktion f ausgehend von der Annahme $Y = f(\mathbf{X}) + \epsilon$ approximieren soll. Die Netzwerke bestehen aus mehreren *Layern* von Neuronen. Der erste Layer hat die Dimension des Inputvektors und jedes Neuron des ersten Layers stellt den Wert der jeweiligen Eigenschaft dar. Der letzte Layer in einem Netzwerk heißt Outputlayer und hat die Dimension der entsprechenden Zielvariable. Die Layer dazwischen werden als *hidden Layer* bezeichnet und die Anzahl der hidden Layer sind mit der Anzahl von Neuronen pro hidden Layer einer der Hyperparameter, die zu jeder Problemstellung angepasst werden müssen.

Je nach Anzahl der hidden Layer und der Neuronen pro Layer spricht man von *Deep Neural Networks* oder *Multilayer Perceptrons*. Weiters wird unterschieden, ob alle Neuronen eines Layers mit allen Neuronen des darauffolgenden Layers verbunden sind oder nicht. Man nennt diese zwei Architekturen *fully-connected* oder *sparse network*. Wenn die Gewichte in den Verbindungen zwischen den Layern über mehrere Neuronen

geteilt werden, spricht man von *convolutional neural networks* [21, Kapitel 7.2].

Man kann die Vorhersagefunktion $h(\mathbf{x})$ mit ihren Parametern $\mathbf{w}$ in fünf Punkte einteilen [21, Kapitel 7.1]. Diese sind:

- **Operation**: Verkettung von mehreren Funktionen

$$h(\mathbf{x}, \mathbf{w}) = h_k(h_{k-1}(\dots h_2(h_1(\mathbf{x}, \mathbf{w}_1), \mathbf{w}_2) \dots, \mathbf{w}_{k-1}), \mathbf{w}_k)$$

- **Regel**: Kettenregel, um die Ableitungen in den Parametern $\mathbf{w}$ von F zu finden
- **Lernalgorithmus**: *Stochastic Gradient Descent*, um die Gewichte anzupassen
- **Vorgehensweise**: *Backpropagation*, um die Kettenregel auszuführen
- **Nichtlinearität**: nichtlineare *Aktivierungsfunktion* $\sigma(\cdot)$

Die Funktionen $h_1, h_2, \dots$ stellen jeweils den Output der nacheinandergereihten Layer dar.

Der Input in h_k ist der Output von h_{k-1}, notiert mit v_{k-1} der Länge n_{k-1}. Der Output von F_k wird mit v_k notiert und hat die Länge n_k. Die Funktion h_k besteht aus einem linearen und einem nichtlinearen Teil:

- Der lineare Teil: $A_k v_{k-1} + b_k$ (Hier ist b_k ein bias-Vektor).
- Der nichtlineare Teil: Aktivierungsfunktion $\sigma(\cdot)$. Zusammen bilden sie also:

$$v_k = h_k(v_{k-1}) = \sigma(A_k v_{k-1} + b_k)$$

wobei σ komponentenweise angewandt wird.

Die Matrix A_k und der bias-Vektor b_k sind die Gewichte an den Verbindungen der einzelnen Neuronen zwischen den Layern $k-1$ und k. Diese Gewichte werden während des Trainings optimiert. Für die Auswahl von σ gibt es mehrere Möglichkeiten. Diese Funktion wird an jedem der n_k Neuronen im Layer k an den Outputs von $A_k v_{k-1} + b_k$ angewandt.

Die Aktivierungsfunktion ist auch ein entscheidender Hyperparameter. Die am häu-

figsten verwendete Aktivierungsfunktion ist die *rectified liner unit function*, auch als *ReLU*-Funktion bekannt [17]. Sie und ihre erste Ableitung sind folgendermaßen definiert:

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

$$\frac{d}{dx}\mathrm{ReLU}(x) = \begin{cases} 0, & \text{für } x < 0 \\ 1, & \text{für } x > 0 \end{cases}$$

In Abbildung 2.26 ist der Aufbau eines kleinen künstlichen neuronalen Netzwerkes skizziert. Man sieht, dass es einen dreidimensionalen Inputvektor gibt, der durch das Netzwerk auf einen skalaren Output abgebildet wird. Diese Architektur, mit weiteren zwei hidden Layern, verdeutlicht, dass in jedem Neuron zuerst eine Linearkombination der Werte der Neuronen des vorherigen Layers gebildet, und anschließend die Aktivierungsfunktion angewendet wird. In diesem Netzwerk sind alle Neuronen eines Layers mit den Neuronen des vorherigen und des darauffolgenden Layers verbunden, somit ist es ein fully-connected neuronales Netzwerk.

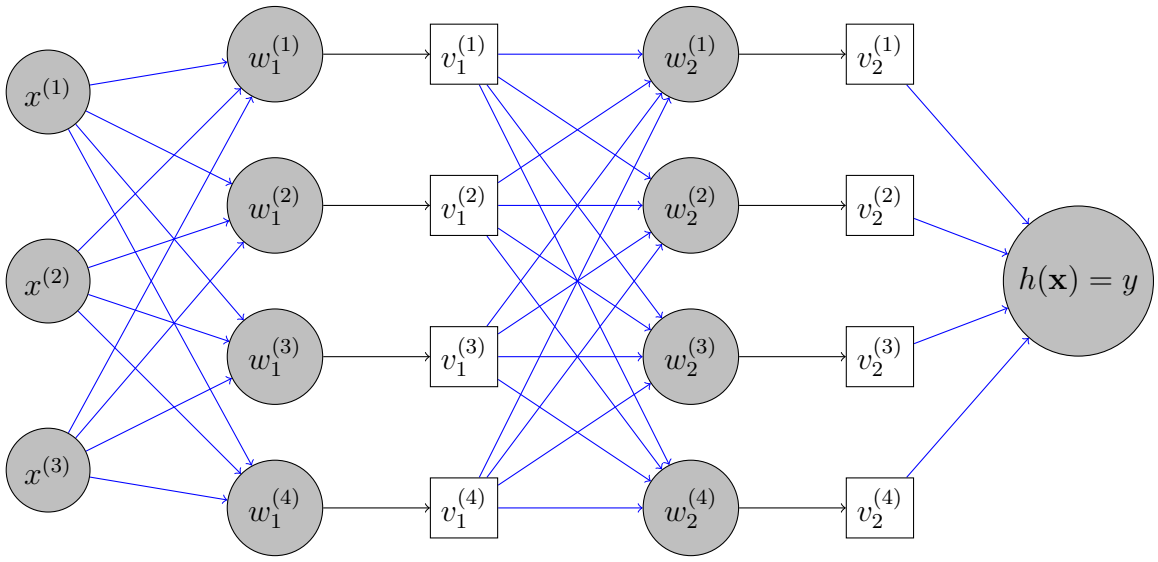

Abbildung 2.6: Der Aufbau eines künstlichen neuronalen Netzwerkes mit drei Inputvariablen und einer Outputvariablen. Diese Architektur hat zusätzlich zu Input- und Outputlayer noch zwei hidden Layer. Diese sind mit $\mathbf{v_k}$ dargestellt. $\mathbf{w_k}$ stellen die zwischenzeitlichen *linearen* Ergebnisse eines jeden Neurons dar. An den blauen Linien werden die Werte der Neuronen mit den Gewichten aus den Matrizen A_k multipliziert. An den schwarzen Linien wird die Aktivierungsfunktion angewandt.

Um die Parameter $\mathbf{w}$ des neuronalen Netzwerkes zu trainieren, wird versucht, die Summe der Fehler auf den Trainingsdaten zu minimieren. Das bedeutet, dass man versucht

$\sum_{i=1}^{n} \ell(y_i, h(\mathbf{x}_i); \mathbf{w})$ zu minimieren. Dies wird mit dem Algorithmus *gradient descent* oder *stochastic gradient descent* gemacht. Hierfür werden die Gradienten mittels *backpropagation* ermittelt [26]. Backpropagation wird auch *reverse-mode automatic differentiation* genannt, und man schafft n partielle Ableitungen mit weniger Rechenaufwand zu berechnen als mit n-maligem Aufwand einer partiellen Ableitung. Das gelingt, weil man viele Zwischenergebnisse immer weiterverwenden kann [21, Kaptitel 7.3].

2.4 Implementierung

Alle Algorithmen, Modelle und Auswertungen wurden in der Programmiersprache *Python* [23] implementiert. Die wichtigsten Bibliotheken für die allgemeine Entwicklung und Auswertung der Modelle waren *Jupyter* [15], *Pandas* [16], *Numpy* [13] und *Matplotlib* [14].

Für die Implementierung der unterschiedlichen Machine Learning Modelle wurde vor allem die Bibliothek *Scikit-Learn* [7] verwendet, welche eine einfache API für diese und ähnliche Problemstellungen bereitstellt. Weiters kann man die Algorithmen leicht auswerten und auch die Hyperparameter optimieren. Für die Implementierung der künstlichen neuronalen Netzwerke wurden *Tensorflow* [1] und die passende API von *Keras* [9] verwendet.

Da sich auf Grund der Menge und Komplexität der Daten der Rechenaufwand in Grenzen gehalten hat, sind alle Modelle auf einem herkömmlichen Computer implementiert und ausgewertet worden.

Alle verwendeten Merkmale wurden standardisiert. Das bedeutet, dass sie jeweils einen empirischen Erwartungswert von 0, und eine empirische Standardabweichung von 1 haben. Nominale Dateneinträge wurden mittels des Verfahrens *One-Hot-Encoding* auf mehrere Variablen aufgeteilt, welche nur die Werte 0 oder 1 annehmen können.

One-Hot-Encoding bedeutet, dass ein nominales Merkmal mit m Klassen in $m-1$ neue Merkmale umcodiert wird. Danach ist jenes Merkmal, welches der Klasse enspricht in der das Beispiel zuvor war, mit 1 codiert und die anderen neuen Merkmale mit 0. Sollte das Beispiel zuvor in der Klasse, für die es kein neues Merkmal gibt, gewesen sein, so entspricht es der Kodierung, dass alle neuen Merkmale 0 sind [11, Seite 67].

Für das Training der Machine Learning Algorithmen wird der von den Anpassungen

übrig gebliebene Datensatz in *Trainigsdaten*, in *Testdaten* und gegebenenfalls in *Validierungsdaten* unterteilt. Diese Unterteilung wurde so durchgeführt, dass die jeweiligen Datensätze pseudo-randomisiert zusammengestellt wurden, und auch der Anteil an prüfungsaktiven Studierenden überall gleich ist. Diesen Vorgang nennt man *Stratifizierung* [11, Seite 53]. Zusätzlich wurde bei der Auswertung des Trainings der Algorithmen Crossvalidierung (CV) verwendet.

Crossvalidation bedeuted, dass der Trainingsdatensatz in k gleich mächtige Mengen zerteilt wird, die zufällig gebildet werden. Anschließend wird das Modell k-mal trainiert, wobei immer eine Menge ausgelassen wird, auf der das Modell danach validiert wird. Somit bekommt man mehrere Validierungswerte und sieht auch deren Verteilung [11, Seiten 31 und 32].

Um die Machine Learning Modelle zu erproben sowie die besten auszuwählen und später in den übergeordneten Ansätzen zu verwenden, die dann eine Schätzung der prüfungsaktiven Studierenden ergibt, wurde wie folgt vorgegangen: Zuerst wurden die Daten auf Vollständigkeit geprüft und neue Merkmale aus bereits vorhandenen Merkmalen berechnet. Hier wurde auch entschieden, ob Merkmale verwendet oder verworfen wurden. Danach wurden die Daten, je nach Ansatz, nach Studienjahren unterteilt und anschließend wurden die unterschiedlichen Modelle ausprobiert. Hier wurde jenes Modell beibehalten, welches nach der entsprechenden Metrik am besten abgeschnitten hatte. Dieses Modell wurde danach in den übergeordneten Ansätzen weiterverwendet.

Es wurden bei allen Modellen unterschiedliche Hyperparameter ausprobiert, um für die Problemstellung den jeweiligen Algorithmus bestmöglich anzupassen. Sie sind in Tabelle 2.1 zusammengefasst. Hier handelt es sich um eine grobe Optimierung der Hyperparameter. Für das Machine Learning Modell, das bei dieser Vorauswertung die besten Werte erzielt hatte, wurden die Hyperparameter nochmals genauer angepasst. Hierzu wird die Methode des *Grid Search* verwendet.

Grid Search bedeutet, dass man einen Bereich an diskreten Werten angibt und danach mit jeder Kombination der Werte das Modell neu trainiert und auswertet. Das ist sehr zeitintensiv und kann somit nur über wenige Kombinationen durchgeführt werden [11, Seiten 76 bis 78]. In dieser Arbeit wurde zuerst Grid Search über eine grobe Einteilung vorgenommen und anschließend nur das beste Modell einer feineren Grid Search unterzogen.

Tabelle 2.1: Ausprobierte Hyperparameter

Modell	Hyperparameter	Ausprobierte Werte
Support Vector Machine	kernel	linear, rbf, polynomial
	gamma	5, 10, 15
	C	50, 100, 150
	epsilon	3, 5, 7
Random Forest	n_estimators	300, 500
	max_depths	100, 150 , 200
	max_leaf_nodes	80, 100, 120
	criterion	mse, mae
	max_samples	100, 500
Künstliches Neuronales Netzwerk	loss functions	mse, mae, huber
	epochs	30, 35
	activation	relu, selu
	num_layers	2, 3, 4
	num_neurons	50, 40, 30, 20
Multiple Linear Regression	basis functions	linear
Multiple Logistic Regression	basis functions	linear

2.5 Auswertung

Die Auswertung kann in drei Kategorien unterteilt werden. Zuerst werden die verschiedenen Machine Learning Modelle bewertet. Diese werden in Ansatz 1 für P1 zur Regression, und in Ansatz 3 für P1 zur Klassifikation beziehungsweise Schätzung von Wahrscheinlichkeiten verwendet. Als zweites werden die übergeordneten Ansätze für P1 an sich bewertet. Als drittes versucht man die beiden Ansätze für P2 zu bewerten.

Generell wurden die Machine Learning Modelle immer für Studierende, die sich in ihrem ersten Studienjahr, im zweiten oder höheren Studienjahren befinden, gebildet. Diese Unterscheidung wird durchgeführt, da man für Studierende in ihrem ersten Jahr wenig Merkmale zur Verfügung hat. Beispielsweise hat man keine Daten über bereits absolvierte ECTS. Demnach wird in der Auswertung auch immer in diese beiden Gruppen unterteilt.

2.5.1 Machine Learning Modelle

In Ansatz 1 für P1 handelt es sich um ein Regressionsproblem. Man versucht anhand von Eigenschaften einer studierenden Person am Beginn eines Studienjahres die erreichte ECTS Anzahl am Ende dieses Jahres vorherzusagen. Es wird als erstes der Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(h(\mathbf{x}_i) - y_i)^2}$$

(wobei n die Anzahl der Testbeispiele ist), als zweites der Mean Absolut Error (MAE):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|h(\mathbf{x}_i) - y_i|$$

und als drittes der *Coefficient of determination* R^2 berechnet:

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

Die ersten beiden Metriken bilden, je nach Entscheidung, ob man den bedingten Erwartungswert oder den bedingten Median approximieren möchte, eine Schätzung des empirischen Risikos. Die Modelle werden anhand dieser Metrik trainiert und anschließend auch mit Daten, die man im Training nicht verwendet hat, getestet. Der Coefficient of determination gibt an, wie gut das Modell die Variabilität der tatsächlichen Werte (in Bezug zum Mittelwert) beschreiben kann. Der Wert liegt normalerweise zwischen 0 und 1. Sollte der Wert negativ sein, bedeutet dies, dass die Schätzung des Modells schlechter die Zielwerte approximieren kann als das arithmetische Mittel der Zielwerte [12, Seiten 317 bis 319].

Bei Ansatz 3 handelt es sich um ein Klassifikationsmodell. Entweder ist eine studierende Person am Ende eines Studienjahres prüfungsaktiv oder nicht. Das soll durch Eigenschaften dieser Person, die man am Beginn des Studienjahres zur Verfügung hat, vorhergesagt werden. Jedes Modell liefert für jede studierende Person einen Wert zwischen 0 und 1, je nachdem wie wahrscheinlich es ist, dass die studierende Person prüfungsaktiv ist oder nicht. Erst anhand eines manuell festgelegten Schwellwertes klassifiziert das Modell in die jeweilige Klasse.

Man hat für die Auswertung aller Modelle eine *Confusion Matrix* gebildet, die folgende

Form besitzt:

$$\begin{bmatrix} \text{True Negatives} & \text{False Positives} \\ \text{False Negatives} & \text{True Positives} \end{bmatrix}.$$

Wobei True Negatives Studierende, die als nicht prüfungsaktiv klassifiziert werden und es tatsächlich nicht waren (TN), True Positives Studierende, die als prüfungsaktiv klassifiziert werden und es tatsächlich waren (TP), False Negatives Studierende, die als nicht prüfungsaktiv klassifiziert werden, aber es tatsächlich waren (FN) und False Positives Studierende, die als prüfungsaktiv klassifiziert werden, aber es tatsächlich nicht waren (FP) entsprechen. Diese wird wiederum anhand von Testdaten gebildet, die der Algorithmus während des Trainings nicht gesehen hat. Weiters wird die *Accuracy* anhand dieser Daten berechnet:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Es wird jedoch aufgrund der Problemstellung nur die zusammengefasste Anzahl an prüfungsaktiven Studierenden benötigt. Deswegen wird keine exakte Klassifizierung durchgeführt, sondern es wird pro studierender Person ein Wert zwischen 0 und 1 ausgegeben, je nachdem wie wahrscheinlich es ist, dass diese Person prüfungsaktiv sein wird oder nicht. Mit diesen Werten wird die Anzahl der erwarteten prüfungsaktiven Studierenden berechnet. Diese erwartete Anzahl wird mit der tatsächlichen Anzahl an prüfungsaktiven Studierenden verglichen. Vor allem diese Differenz ist für die Wahl des besten Vorhersagemodells ausschlaggebend. Je kleiner die Differenz ist, desto besser ist das Modell.

2.5.2 Ansätze für P1

Bei Ansatz 1 wird überprüft, ob es Sinn macht die erreichten ECTS der Studiereden Jahr für Jahr zu schätzen. Sollten die Schätzungen der ECTS Werte geringe RMSE und MAE im Bereich bis 5 ECTS ergeben, wird eine Vorhersage Jahr für Jahr durchgeführt. Anschließend wird im letzten Jahr, für das die ECTS geschätzt wurden, entschieden, ob die studierende Person prüfungsaktiv war oder nicht. Diese verkettete Schätzung findet auch auf einem eigenen Testdatensatz statt. Nun wird bewertet, wie weit sich die geschätzte Anzahl an prüfungsaktiven Studierenden von der tatsächlichen Anzahl in diesem Jahr unterscheidet.

Bei Ansatz 2 kann man aufgrund von mangelnden Daten keine Bewertung der Berechnungen durchführen. Weil es sich um berechnete relative Häufigkeiten handelt, macht es keinen Sinn in einen Testdatensatz und Trainingsdatensatz, welche zufällig gewählt

werden zu unterscheiden. Es werden die relativen Häufigkeiten berechnet, die man an einem Testzeitraum in der Zukunft bewerten müsste.

Bei Ansatz 3 kann, wie oben angeführt, die geschätzte Anzahl an prüfungsaktiven Studierenden mit der tatsächlichen Anzahl verglichen werden.

Schlussendlich ist es das Ziel, dass die geschätzte Anzahl an prüfungsaktiven Studierenden im Jahr $t = 3$ möglichst nahe an der tatsächlichen Anzahl liegt.

2.5.3 Ansätze für P2

Man hat bei P2 nicht ausreichend Daten, um die Studienbeginner in den kommenden Jahren vorherzusagen. Das bedeutet auch, dass es innerhalb der vorhandenen Daten nicht möglich ist, einen Testdatensatz zu bilden, an dem die beiden Ansätze für P2 verglichen werden können. Bei beiden Ansätzen hängt sehr viel von der Schätzung der Anzahl der zukünftigen Studienbeginner ab, für die man mehr Kalenderjahre im Datensatz benötigen würde, als man zur Verfügung hat.

Für Ansatz 1 wird überprüft, ob es für zwei aus den fünf vorhandenen Kalenderjahren sinnvoll gewesen wäre, diesen Ansatz zu wählen oder, ob sich die Merkmalskombinationen stark von jenen im Vorjahr unterscheiden. Dafür wird die Anzahl, welche bei einer tatsächlichen Anwendung geschätzt werden muss, als gegeben angenommen. Anschließend vergleicht man, wie stark sich eine Schätzung anhand der tatsächlichen Merkmalskombinationen mit einer Schätzung anhand von Merkmalskombinationen aus dem Vorjahr unterscheiden. Zusätzlich werden diese beiden Werte mit der tatsächlichen Anzahl an prüfungsaktiven Neuinskribierenden in diesem Kalenderjahr verglichen.

Weil Ansatz 2 noch mehr auf der Schätzung der Anzahl der neuinskribierenden Studierenden aufbaut, kann man diesen Ansatz mit den vorhandenen Daten nicht bewerten oder legitimieren. Zusätzlich zur Schätzung der Anzahl der Studierenden wird die Schätzung der Prüfungsaktivität so wie in P1 durchgeführt. Hierfür kann man die Ergebnisse aus P1 als einen Richtwert nehmen.

3 Ergebnisse

3.1 Ansätze für Problem 1

3.1.1 Ansatz 1

In Tabelle 3.1 sind die Machine Learning Modelle und die Metriken zu ihrer Auswertung zusammengefasst.

Tabelle 3.1: Auswertung der Machine Learnig Modelle in Ansatz 1 für P1.

Metrik		lineare Regression	Random Forest	SVM	KNN (ohne CV)
RMSE (Crossvalidation)	1 Jahr	18.7 ± 0.2	19.2 ± 0.3	19.7 ± 0.4	18.72
	$\geq$ 2 Jahr	16.8 ± 0.2	15.4 ± 0.2	19.2 ± 0.3	14.8
MAE (Trainingsdaten)	1 Jahr	15.6	15.9	15.9	14.5
	$\geq$ 2 Jahr	13.3	11.7	16.2	10.4
R2-Score	1 Jahr	0.06	$-.01$	-0.05	0.06
	$\geq$ 2 Jahr	0.38	0.48	0.17	0.52
% Accurancy	1 Jahr	61	61	60	63
	$\geq$ 2 Jahr	80	78	66	80

Nach dieser Auswertung ist das KNN jenes Modell, welches für diese Problemstellung am besten funktioniert. Jedoch sind auch bei diesem Modell der RMSE und der MAE in der Größenordnung von ca. 16 ECTS. Das bedeutet, der Fehler ist durchschnittlich so groß wie die Schwelle zur Prüfungsaktivität. Weiters ist vor allem die Schätzung der ECTS im ersten Studienjahr, wo man weniger Inputvariablen zur Verfügung hat, bei allen Modellen schlechter, als die Schätzung der ECTS in den darauffolgenden Studienjahren.

Man sieht, dass bei allen Modellen die jeweiligen Fehler groß sind und das bedeutet

jeweils für das erste darauffolgende Jahr. Deswegen ist, aufgrund der schlechten Vorhersagbarkeit der ECTS, und der weiteren Fehlerfortpflanzungen bei einer Vorhersage über mehrere Studienjahre, dieser Ansatz nicht brauchbar.

3.1.2 Ansatz 2

Es war, wie oben angeführt, wegen mangelnder Daten nicht möglich diesen Ansatz zu überprüfen. Aus den 72 eingeteilten Kategorien war es für 49 Kategorien möglich, Übergangsmatrizen zu berechnen. Für alle anderen Kategorien hat es in dem vorhandenen Datensatz zu wenig Daten gegeben.

Für die Beispielsklasse *Weiblich, Rechtswissenschaften, Steiermark* und *AHS Vorbildung* haben die Übergangsmatrizen der ersten fünf Jahre für den Prozess X wie folgt ausgesehen:

$$\begin{bmatrix} 0.05 & 0.19 & 0.53 & 0.23 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.02 & 0.13 & 0.79 & 0.07 \\ 0.02 & 0.49 & 0.19 & 0.29 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.01 & 0.05 & 0.82 & 0.13 \\ 0.01 & 0.46 & 0.24 & 0.29 \end{bmatrix},$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.05 & 0.01 & 0.83 & 0.11 \\ 0.00 & 0.18 & 0.25 & 0.50 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.15 & 0.02 & 0.71 & 0.12 \\ 0.03 & 0.27 & 0.28 & 0.42 \end{bmatrix}.$$

Um eine Übergangswahrscheinlichkeit abzulesen, geht man wie folgt vor: Die Zeilen der Matrizen geben den aktuellen Status der studierenden Person an. Dieser kann entweder „aktiv und weiterhin inskribiert“, „nicht aktiv und weiterhin inskribiert“, „aktiv und nicht weiter inskribiert“ oder „nicht aktiv und nicht weiter inskribiert“ sein. Die Spalten geben die Klasse an, in die eine studierende Person im nächsten Jahr kommen kann. Beispielsweise gibt das Element (3, 2) jeder Matrix die Wahrscheinlichkeit an, mit der eine studierende Person aus Klasse c (prüfungsakiv und weiterhin inskribiert) in die Klasse b (nicht prüfungsakiv und nicht weiterhin inskribiert) gelangt.

Mit diesen Übergangswahrscheinlichkeiten ist es für Studierende dieser Klasse möglich, die erwartete Anzahl an prüfungsaktiven Studierenden zu einem Zeitpunkt t in der Zukunft zu berechnen.

3.1.3 Ansatz 3

In Tabelle 3.2 sieht man die zusammengefassten Ergebnisse der Machine Learning Modelle unter der Vorgehensweise von Ansatz 3. In Zeile 1 sind die tatsächlichen Anzahlen an prüfungsaktiven Studierenden mit den geschätzten Anzahlen dargestellt. Man sieht hier, dass diese nicht sehr weit voneinander abweichen. In den Zeilen 2 und 3 sind Ergebnisse einer möglichen Klassifizierung mittels eines Schwellwertes von 0.5 dargestellt. Man sieht in der Confusion Matrix, dass bei einer durchgeführten Klassifizierung die Ergebnisse nicht so gut ausfallen würden, weil sich die False Positiv und False Negativ klassifizierten Testbeispiele nicht gut aufheben.

Tabelle 3.2: Auswertung der Machine Learnig Modelle in Ansatz 3 für P1

		log. Reg.	RF	SVM	KNN
1 Jahr	Predicted	129.39	128.17	128.84	129.29
	Real	129	129	129	129
$\geq$ 2 Jahre	Predicted	121.25	117.46	120.59	120.9
	Real	121	121	121	121
1 Jahr	Confusion Matrix	$\begin{bmatrix} 1938 & 35 \\ 1138 & 26 \end{bmatrix}$	$\begin{bmatrix} 1572 & 401 \\ 804 & 360 \end{bmatrix}$	$\begin{bmatrix} 1677 & 296 \\ 856 & 308 \end{bmatrix}$	$\begin{bmatrix} 1672 & 301 \\ 851 & 313 \end{bmatrix}$
$\geq$ 2 Jahre	Confusion Matrix	$\begin{bmatrix} 1535 & 301 \\ 564 & 519 \end{bmatrix}$	$\begin{bmatrix} 1506 & 330 \\ 468 & 615 \end{bmatrix}$	$\begin{bmatrix} 1673 & 163 \\ 783 & 300 \end{bmatrix}$	$\begin{bmatrix} 1662 & 174 \\ 788 & 292 \end{bmatrix}$
1 Jahr	CV Scores	0.63 ± 0.00	0.62 ± 0.01	0.63 ± 0.02	0.62 ± 0.01
$\geq$ 2 Jahre	CV Scores	0.70 ± 0.01	0.73 ± 0.01	0.68 ± 0.01	0.72 ± 0.01

Man erkennt in den Ergebnissen, dass alle vier Modelle auf einem Testdatensatz gut funktionieren und in ihrer Genauigkeit nicht weit voneinander abweichen. Man sieht auch im Vergleich, wie die Klassifikatoren abschneiden, wenn sie jedes Beispiel bewerten müssten. Es ist erkenntlich, dass sich die FP und FN nicht so genau aufheben, wie es mit der Ausgabe der Wahrscheinlichkeiten der Fall ist. Die unterschiedlichen Anzahlen der Ausgaben kommen zustande, weil bei der Confusion Matrix mittels einer Crossvalidation gearbeitet wurde. Das heißt, auch hier wurde auf Daten ausgewertet, auf die das Modell nicht trainiert wurde.

Das beste Modell nach dieser Auswertung ist das Künstliche Neuronale Netzwerk.Daneben liefern auch das Support Vector Machine Modell und die logistische Regression beinahe idente Ergebnisse auf dem Testdatensatz.

3.1.4 Auswahl

Aufgrund der schlechten Vorhersagbarkeit und der weiteren Fehlerfortpflanzungen muss Ansatz 1 verworfen werden. Es macht somit keinen Sinn den ECTS Wert der Studierenden Jahr für Jahr zu schätzen.

Ansatz 2 liefert Übergangswahrscheinlichkeiten, welche auf diesem Datensatz stabil sind. Dieser Ansatz kann für wenige Klassen zuverlässig verwendet werden. Der Nachteil dieses Ansatzes ist, dass man ihn erst in einer Zeitspanne in der Zukunft überprüfen kann und, dass man für die Berechnungen der Wahrscheinlichkeiten für einige Klassen zu wenig Daten hat.

Ansatz 3 hat auf dem gebildeten Testdatensatz gut abgeschnitten. Die Modelle dieses Ansatzes müssen für eine gewisse Zeitspanne gebildet werden. Der Vorteil ist, dass bei diesem Ansatz mehr Inputmerkmale pro Person verwendet werden können, um eine Wahrscheinlichkeit zu berechnen, als in Ansatz 2.

Aufgrund der oben angeführten Ergebnisse wird Ansatz 3 für P1 verwendet.

3.2 Ansätze für Problem 2

Für P2 gibt es keine Möglichkeit die beiden Ansätze sinnvoll miteinander zu vergleichen. Es sind zu wenig Daten vorhanden um einen Testdatensatz zu bilden, auf welchem beide Ansätze miteinander verglichen werden können. Weiters würde man in beiden Ansätzen eine Schätzung der Anzahl der neu inskribierenden Studierenden durchführen müssen. Dafür sind in den Daten mit fünf Kalenderjahren zu wenig Kalenderjahre vorhanden, um eine ernsthafte Schätzung zu bilden.

3.2.1 Ansatz 1

Als Legitimation für diesen Ansatz kann man folgende Ergebnisse in Tabelle 3.3 verwenden:

In dieser Tabelle sieht man in den Zeilen *1 Jahr* und *2 Jahre* die Daten für Vorhersagen über ein beziehungsweise zwei Jahre. In der Spalte *Prediction reale Daten* werden die

Tabelle 3.3: Legitimation Ansatz 1 P2

Zeitspanne der Schätzung		Prediction reale Daten	Prediction dummy Daten (Anzahl gegeben)	tatsächliche Anzahl
1 Jahr	2016	1118	1105	1092
	2017	1000	984	973
2 Jahre	2016	867	878	819
	2017	769	769	721

tatsächlichen Daten aus den Kalenderjahren 2016 und 2017 über ein beziehungsweise zwei Jahre vorhergesagt. Die Spalte *Prediction dummy Daten* zeigt eine Vorhersage von Daten, welche die Merkmalskombinationen aus dem letzten verfügbaren Jahr haben. Die letzte Spalte gibt die tatsächliche Anzahl an prüfungsaktiven Studierenden im zu schätzenden Kalenderjahr, an.

3.2.2 Ansatz 2

Wie oben beschrieben, sind für die Schätzung der Anzahl der neu inskribierenden Studierenden nicht genügend Daten vorhanden. Weil dieser Ansatz größtenteils auf dieser Schätzung beruht, konnten keine Ergebnisse berechnet werden. Für die Vorhersage, ob die Studierenden prüfungsaktiv sein werden oder nicht, sollte Ansatz 3 aus P1 verwendet werden.

3.2.3 Auswahl

Es ist hier nicht möglich eine Auswahl zu treffen. Beide Ansätze müssten davor auf einen größeren Datensatz getestet werden.

4 Diskussion

In diesem Abschnitt wird näher auf die Ergebnisse und auf entscheidende Punkte der unterschiedlichen Ansätze eingegangen. Darüber hinaus werden weitere Möglickkeiten genannt, die man verfolgen kann, um die Problemstellung noch genauer zu untersuchen.

4.1 Problem 1

Grundsätzlich war bei diesem Problem das Verständnis der Problemstellung wichtig. Dadurch, dass man nicht jeder studierenden Person exakt zuordnen muss, ob sie prüfungsaktiv sein wird oder nicht, sondern nur die Gesamtanzahl möglichst genau schätzen muss, vereinfacht sich die Problemstellung. Durch diese Eigenschaft ist es nicht entscheidend, wenige FP oder FN klassifizierte Studierende zu haben, da diese sich gegenseitig aufheben, solange sie sich in der Waage halten.

Ein weiteres Merkmal der Problemstellung war es, dass man immer zwischen erstjährigen Studierenden und Studierenden in höheren Studienjahren unterscheiden muss. Für die erstjährigen Studierenden hat man weniger Inputmerkmale zur Verfügung. Vor allem die *Anzahl der bisherigen ECTS* und die *ECTS im Jahr zuvor* sind für Studierende in höheren Studienjahren gute Prädiktoren und genau diese fehlen bei Studienanfängern. Deswegen sind die Schätzer für Studienbeginner grundsätzlich nicht so genau wie jene für Studierende aus höheren Studienjahren.

Bei P1 hat es sich als erfolgreicher herausgestellt, eine Schätzung der Anzahl an prüfungsaktiven Studierenden über eine Zeitspanne von mehreren Jahren zu machen, als Jahr für Jahr die ECTS Anzahl pro studierender Person vorherzusagen. Die jährliche Prädiktion erweist sich als sehr schwierig und bringt eine Fehlerfortpflanzung mit sich. Weiters hat dieser Ansatz für die konkrete Problemstellung keinen erkenntlichen Vorteil.

Weitere Möglichkeiten um Ansatz 1 zu verbessern, wären Folgende: Man kann hier noch an den Daten, auf denen man die unterschiedlichen Machine Learning Modelle bildet, arbeiten. Beispielsweise kann man die ECTS Werte, die vorhergesagt werden sollen, nach oben beschränken. Es ist für die zusammengefasste Anzahl nicht ausschlaggebend, wenn den wenigen Studierenden, die tatsächlich sehr viele ECTS erreicht haben, nur eine gewisse Obergrenze zugeordnet wird. Im Gegensatz bekommt man dadurch Modelle, die nicht von wenigen Ausreißern in den Trainingsdaten beeinflusst worden sind. Diesen Ansatz müsste man wiederum auf einem gesonderten Testdatensatz validieren.

Um Ansatz 2 weiter zu vertiefen, kann man überlegen, wie man die Studierenden nach den vier Merkmalen in weniger Klassen zusammenfassen kann. Grundsätzlich verliert man bei jeder Zusammenfassung von Merkmalskombinationen gewisse Informationen. Im Gegensatz verhindert man eine zu hohe Anzahl an Klassen, wobei man für viele die Übergangswahrscheinlichkeiten nicht berechnen kann. Eine konkrete Überlegung ist es, eine Distanz zwischen den Übergangsmatrizen einzelner Klassen festzulegen. Zusätzlich muss man für viele Kombinationen von Klassen berechnen, ob sie zusammengenommen eine relevante Anzahl an Studierenden darstellen oder nicht. Danach kann man in jene zwei (oder mehr) Klassen unterteilen, die jeweils eine hohe Anzahl an Studierenden beinhalten und eine große Distanz der Übergangswahrscheinlichkeiten zueinander besitzen.

Bei Ansatz 3 kann man vor allem die Testung des Ansatzes noch ausführlicher durchführen. Bisher wurde dieser Ansatz auf einem kleinen Testdatensatz evaluiert, wo er gute Ergebnisse geliefert hat. Es wäre interessant, diesen Ansatz auf einem großen Testdatensatz zu überprüfen.

Generell ist diese Vorhersage immer mit der Unsicherheit behaftet, dass Zusammenhänge, die in den vorhandenen Daten gefunden worden sind, sich in den drei Jahren grundlegend verändern könnten. Ein Beispiel dafür könnte die Coronaviruspandemie im Jahr 2020 sein. Sollte sich durch die damit verbundenen Umstellungen das Verhalten der Studierenden grundsätzlich verändern, ist es nicht möglich eine gute Vorhersagefunktion auf Daten von vor dem Jahr 2020 zu bilden. Man muss darauf achten, dass die Trainingsdaten repräsentativ für die tatsächliche Anwendung sind. Dieses Problem kann man nicht umgehen, weil es sich um eine Vorhersage über eine gewisse Zeitspanne handelt und es immer unerwartete Ereignisse geben kann.

4.2 Problem 2

Wie im Abschnitt zuvor bereits erwähnt, hat es sich als schwierig herausgestellt, die Prüfungsaktivität von Studienbeginnern vorherzusagen, selbst wenn man die Anzahl und Merkmalskombinationen kannte. Die Problemstellung von P2 ist intrinsisch schwer zu lösen, da man hier zusätzlich weder Informationen über die Anzahl noch über die Merkmalskombinationen zur Verfügung hat.

In dieser Arbeit war es nicht möglich eine ernsthafte Überprüfung der Ansätze durchzuführen, weil zu wenig Daten vorhanden waren. Vor allem für die Schätzung der Anzahl der Studienbeginner in den folgenden Jahren benötigt man mehr (und auch andere) Daten. Beispielsweise wäre es interessant, weitere demografische Daten wie die Anzahl von Maturaabschlüssen an unterschiedlichen Schulzweigen und Schulstandorten zu verwenden. Zusätzlich braucht man für eine ernstzunehmende Schätzung von zukünftigen Studienbeginnern mehr Daten als von fünf Kalenderjahren, um eine Trendanalyse durchzuführen.

Beide vorgestellten Ansätze bauen stark auf der Anzahl der zukünftigen Studienbeginner auf, die man aber nicht schätzen konnte. Aus diesem Grund konnten beide Ansätze nicht getestet werden.

Eine weitere Verbesserungsmöglichkeit beider vorgestellten Ansätze wäre es, für die Anzahl der zukünftigen Studienbeginner nicht nur einen einzelnen Wert zu schätzen, sondern ein Konfidenzintervall zu bilden. Anhand dieses Konfidenzintervalls könnte man unterschiedliche Szenarien berechnen und miteinander vergleichen.

4.3 Fazit

Zusammenfassend kann man aus der vorliegenden Arbeit folgende Schlüsse ziehen:

- Es macht Sinn die vorliegende Problemstellung in zwei separate Probleme zu unterteilen. Anhand des verwendeten Datensatzes sieht man, dass ein relevanter Anteil von derzeitigen Studierenden bereits vor drei Jahren inskribiert gewesen ist (Altbestand).

- Obwohl viele unterschiedliche Machine Learning Modelle ausprobiert worden

sind, haben sie durchwegs ähnliche Ergebnisse geliefert. Aus diesem Grund kann man davon ausgehen, dass die Daten nicht mehr Informationen beinhalten und die Wahl des Modells keinen gravierenden Unterschied macht. Man könnte zwar weitere Modelle erproben, jedoch sollte man sich eher neue Ansätze für die Problemstellung überlegen.

- Die wichtigste Erkenntnis der vorliegenden Arbeit war das Verstehen der Problemstellung. Der Übergang von genauer Klassifizierung jeder einzelnen studierenden Person auf die Schätzung der erwarteten Gesamtanzahl von prüfungsaktiven Studierenden hat die Testergebnisse verbessert.

- Bei P1 kann man mit zusätzlichen Daten noch Ansatz 2 überprüfen. Sollte auch dieser Ansatz vielversprechend sein, können noch weitere Überlegungen hinsichtlich einer Auswahl von wenigen relevanten Zusammenfassungen von mehreren Merkmalskombinationen durchgeführt werden.

- Vor allem bei P2 kann man mittels zusätzlicher Daten und Daten, die weiter in der Vergangenheit zurückliegen, bessere Schätzungen der Anzahl von neuhinzukommenden Studierenden durchführen. Weiters können die vorgestellten Ansätze anhand zukünftiger Daten bewertet werden.

- Jede Vorhersage hängt von der Stabilität der Zusammenhänge in der Zukunft ab. Grundlegende Änderungen des Verhaltens von Studierenden können nicht vorhergesagt werden.

Literatur

[1] Abadi Martin u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Dimitri P. Bertsekas und John N. Tsitsiklis. *Introduction to Probability*. Belmont, Massachussets : Athena Scientific, 2008. ISBN: 978-1-886529-23-6.

[3] Christopher Bishop. *Pattern Recognition*. New York, New York: Springer, 2006. ISBN: 0-387-31073-8.

[4] Bernhard E. Boser, Isabelle M. Guyon und Vladimir N. Vapnik. „A Training Algorithm for Optimal Margin Classifiers“. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, S. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[5] L. Breiman u. a. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN: 9780412048418. URL: https://books.google.at/books?id=JwQx-WOmSyQC.

[6] Leo Breiman. „Random Forests“. English. In: *Machine Learning* 45.1 (2001), S. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A%3A1010933404324.

[7] Lars Buitinck u. a. „API design for machine learning software: experiences from the scikit-learn project“. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, S. 108–122.

[8] Steven Busuttil. „Support vector machines“. In: *1st Computer Science Annual Workshop (CSAW'03), Msida. 34-39.* 1 (2003), S. 34–39. URL: https://www.um.edu.mt/library/oar/handle/123456789/20624.

[9] Francois Chollet u. a. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[10] Jürgen Elstrodt. *Maß- und Integrationstheorie*. Springer-Verlag Berlin Heidelberg New York, 1996. ISBN: 3-540-15307-1.

[11] Aurelion Geron. *Hands-On Machine Learning with Scikit-Learn, Kears & Tensorflow*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019. ISBN: 978-1-492-03264-9.

[12] John Guttag. *Introduction to computation and programming using Python: with application to understanding data*. The MIT Press, Cambridge, MA, 2017. ISBN: 9780262529624.

[13] Charles R. Harris u. a. „Array programming with NumPy“. In: *Nature* 585 (2020), S. 357–362. DOI: 10.1038/s41586-020-2649-2.

[14] John D. Hunter. „Matplotlib: A 2D Graphics Environment“. In: *Computing in Science Engineering* 9.3 (2007), S. 90–95. DOI: 10.1109/MCSE.2007.55.

[15] Thomas Kluyver u. a. „Jupyter Notebooks – a publishing format for reproducible computational workflows“. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Hrsg. von F. Loizides und B. Schmidt. IOS Press. 2016, S. 87–90.

[16] Wes McKinney. „Data Structures for Statistical Computing in Python“. In: *Proceedings of the 9th Python in Science Conference*. Hrsg. von Stéfan van der Walt und Jarrod Millman. 2010, S. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[17] Rahul Parhi und Robert D. Nowak. „The Role of Neural Network Activation Functions“. In: *IEEE Signal Processing Letters* 27 (2020), S. 1779–1783. DOI: 10.1109/LSP.2020.3027517.

[18] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[19] J. R. Quinlan. „Induction of Decision Trees“. In: *Mach. Learn.* 1.1 (März 1986), S. 81–106. ISSN: 0885-6125. DOI: 10.1023/A:1022643204877. URL: https://doi.org/10.1023/A:1022643204877.

[20] Shai Shalev-Shwartz und Shai Ben-David. *Understanding Machine Learning - from Theory to Algorithms*. Cambridge University Press, 2014. ISBN: 978-1-107-05713-5.

[21] Gilbert Strang. *Linear Algebra and Learning from Data*. Wellesley - Cambridge Press, 2019. ISBN: 978-0-692-19638-0.

[22] Ngoc Tran u. a. „Hyper-parameter Optimization in Classification: To-do or Not-to-do“. In: *Pattern Recognition* 103 (Juli 2020), S. 107245. DOI: 10.1016/j.patcog.2020.107245.

[23] Guido Van Rossum und Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[24] Walter Pitts Warren McCulloch. „A logical calculus of the ideas immanent in nervous activity“. In: *The bulletin of mathematical biophysic* 5 (Dez. 1943), S. 115–133. DOI: 10.1007/BF02478259.

[25] Larry Wasserman. *All of Statistics*. New York, New York: Springer, 2004. ISBN: 0-387-40272-1.

[26] P. J. Werbos. „Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences“. Diss. Harvard University, 1974.

A Quellcode

In diesem Anhang wird der Quellcode für die Erstellung der Machine Learning Modelle und die Berechnung der Ergebnisse angeführt. In der Auflistung Ordnerstruktur wird dargestellt welche Dateien erstellt worden sind. In den weiteren Auflistungen sind die Python Files dargestellt, die einerseits zum Erstellen der passenden Daten verwendet wurden und andererseits die Berechnungen der unterschiedlichen Ansätze durchgeführt haben.

A.1 Ordnerstruktur

Code A.1: `Ordnerstruktur`

```
project/
|
|---raw_data/
|   |   bwl_pad_jus.csv
|   |   adapted_data.csv
|
|---data_generating/
|   |   data.py
|
|---problem1/
|   |   P1_A1.py
|   |   P1_A2.py
|   |   P1_A3.py
|
|---problem2/
|   |   P2.py
```

A.2 Datenaufbereitung

Code A.2: data.py

```
### importing libraries ###

import os
import math

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt



### importing data files ###

# relative path to the data folder

data_path = '../raw_data'
cwd = os.path.dirname(__file__)
raw_data_folder = os.path.abspath(os.path.join(cwd, data_path))



# labels of features, which will be used later on

columns = [
    'matrikel_num', 'Studienjahr',
    'geschlecht', 'avgECTS_sem_before',
    'ects_year_before', 'year',
    'first_exam_negative', 'AHS_dummy',
    'BHS_dummy', 'ausland_vorbildung_dummy',
    'sonstige_vorbildung_dummy', 'delayed_dummy',
    'num_parallel_studies', 'jus_dummy',
    'bwl_dummy', 'years_since_matura',
    'styria_dummy', 'not_styria_dummy',
    'germany_dummy', 'other_foreign_dummy',
    'years_since_18', 'planned_duration',
    'full_duration_sem', 'full_duration_sem_before',
    'firstGen', 'cum_ects_pos_before',
    'status_key', 'ECTS_year',
    'SWS_year', 'active_dummy',
    'subject'
    ]



# helper function for importing data. Returns a "Pandas DataFrame"
    object.
def load_df(name):
```

```
47      return pd.read_csv(os.path.join(raw_data_folder, name))
48
49  df = load_df('bwl_pad_jus.csv').reset_index(drop = True)
50
51
52
53  ### helper functions for calculating new features ###
54
55  # calculating "full_duration_sem_before" as a new column. This is the
        number of semesters the student has already studied.
56  def full_duration_sem_before(df):
57      for i in range(len(df)):
58          a = math.floor(df.loc[i, "full_duration_sem"])
59          if a <= 1:
60              df.loc[i, "full_duration_sem_before"] = 0
61          else:
62              if df.loc[i, "last_semester_name"][2] == 'W':
63                  df.loc[i, "full_duration_sem_before"] = a - 1
64              else:
65                  df.loc[i, "full_duration_sem_before"] = a - 2
66      return df
67
68
69
70  # calculating "geschlecht" categorial.
71  def geschlecht(df):
72      for i in range(len(df)):
73          a = df.loc[i, "geschlecht"]
74          if a == "W":
75              df.loc[i, "geschlecht"] = 0
76          else:
77              df.loc[i,"geschlecht"] = 1
78      return df
79
80  # calculating 'cum_ects_pos_before'. Calculating the ECTS before the
        current year. If not, we would be using
81  # information, which should actually be learned afterwards.
82  def cum_ects_pos_before(df):
83      df['cum_ects_pos_before'] = df['avgECTS_sem_before']*df['
           full_duration_sem_before']
84      return df
85
86
87  # calculating "avgECTS_sem_before" as a new column. These are the
        average ECTS per semester before the current year.
88  def avgECTS_sem_before(df):
89      for i in range(len(df)):
```

```
90          cum_ects_before_pseudo = df.loc[i, 'cumulated_ects_pos'] - df.
               loc[i, 'ECTS_year']
91          if not df.loc[i, 'full_duration_sem_before'] == 0:
92              df.loc[i, 'avgECTS_sem_before'] = float(
                   cum_ects_before_pseudo) / df.loc[i, '
                   full_duration_sem_before']
93          else:
94              df.loc[i, 'avgECTS_sem_before'] = 0
95      return df
96
97
98  # calculating "Studienjahr" as a new column.
99  def Studienjahr(df):
100     for i in range(len(df)):
101         df.loc[i, "Studienjahr"] = int((df.loc[i, "
               full_duration_sem_before"]//2) + (df.loc[i, "
               full_duration_sem_before"]%2) + 1)
102     return df
103
104
105 # deleting rows with negative "years since matura".
106 def Matura(df):
107     df.drop(df[df['years_since_matura'] < 0].index, inplace = True)
108     return df
109
110
111
112 # calculating "ECTS_year_before". Only if they are available in the
       data.
113 def year(df):
114     for i in df.index:
115         num = df.loc[i, 'year'][:2]
116         df.loc[i, 'year'] = int(num)
117     return df
118
119 def ects_year_before(df):
120     list_studienjahre = list(df['year'].unique())
121     list_studienjahre.sort()
122     for i in list_studienjahre[1:]:
123         df_year = df.query('year == @i')
124         df_year0 = df.query('year == (@i-1)')
125
126         for num in df_year.index:
127             if df_year.loc[num, 'Studienjahr'] > 1:
128                 matr = df_year.loc[num, 'matrikel_num']
129                 if (num-1) in df_year0.index and df_year0.loc[(num-1),
                       'matrikel_num'] == matr:
```

```
                    df.loc[num, 'ects_year_before'] = df.loc[(num-1),
                        'ECTS_year']

    return df


# calculating one-hot-encoding for subject.
def subject(df):
    for i in range((len(df))):
        if df.loc[i, 'subject'] == 'Rechtswissenschaften':
            df.loc[i, 'jus_dummy'] = 1
            df.loc[i, 'bwl_dummy'] = 0
        elif df.loc[i, 'subject'] == 'Betriebswirtschaft':
            df.loc[i, 'jus_dummy'] = 0
            df.loc[i, 'bwl_dummy'] = 1
        else:
            df.loc[i, 'jus_dummy'] = 0
            df.loc[i, 'bwl_dummy'] = 0
    return df



### here I am actually manipulating the data with the helper functions
    . ###
# functions for saving manipulated DataFrame
def saving(df, name):
    df.to_csv(os.path.join(raw_data_folder, name))


def data_manipulation_pipeline(df, columns, name, save_or_return): #
    does manipulations to df and is able to save
    subject(df)
    geschlecht(df)
    full_duration_sem_before(df)
    avgECTS_sem_before(df)
    cum_ects_pos_before(df)
    Studienjahr(df)
    Matura(df)
    year(df)
    ects_year_before(df)

    df = df[columns] # ordering of columns

    if save_or_return:
        saving(df, name)
    else:
        return df

```

```
# calling the pipeline
data_manipulation_pipeline(df, columns, 'adapted_data.csv', True)
```

A.3 P1 Ansatz 1

Code A.3: P1_A1.py

```
### importing libraries ###

import sys
import os
import random as rd

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

import tensorflow as tf
from tensorflow import keras

scaler = StandardScaler()

### importing data files ###

# relative path to the data folder
data_path = '../raw_data'
cwd = os.path.dirname(__file__)
raw_data_folder = os.path.abspath(os.path.join(cwd, data_path))

def load_df(name):
    return pd.read_csv(os.path.join(raw_data_folder, name))
```

```
39
40
41 # loading data
42 df = load_df('adapted_data.csv').drop(['Unnamed: 0'], axis =1)
43
44
45 # used features for first study year
46 features_year1 = [
47     'styria_dummy', 'not_styria_dummy',
48     'germany_dummy', 'num_parallel_studies',
49     'cum_ects_pos_before', 'years_since_matura',
50     'firstGen', 'geschlecht',
51     'AHS_dummy', 'BHS_dummy',
52     'ausland_vorbildung_dummy', 'sonstige_vorbildung_dummy',
53     'jus_dummy', 'bwl_dummy',
54     'delayed_dummy', 'ECTS_year',
55     'active_dummy'
56     ]
57
58 # used features for highter study years
59 features_years = [
60     'Studienjahr', 'styria_dummy',
61     'not_styria_dummy', 'germany_dummy',
62     'num_parallel_studies', 'cum_ects_pos_before',
63     'avgECTS_sem_before', 'ects_year_before',
64     'full_duration_sem_before', 'geschlecht',
65     'years_since_matura', 'firstGen',
66     'AHS_dummy', 'BHS_dummy',
67     'ausland_vorbildung_dummy', 'sonstige_vorbildung_dummy',
68     'delayed_dummy', 'jus_dummy',
69     'bwl_dummy', 'ECTS_year',
70     'active_dummy'
71     ]
72
73
74 ### helper functions and training functions ###
75
76 def active(df_, prediction):
77     df = df_.copy(deep = True)
78     df.reset_index(drop = True)
79     if len(df) == len(prediction):
80         new_column = []
81         for i in range(len(prediction)):
82             if prediction[i] >= 16:
83                 new_column.append(1)
84             else:
85                 new_column.append(0)
```

```
86          df.insert(len(df.columns), 'active_predicted', new_column,
                allow_duplicates = True)
87          df.insert(len(df.columns), 'ECTS_predicted', prediction,
                allow_duplicates = True)
88      else:
89          print('ERROR')
90      return df
91
92
93  def ratio(df):
94      ratio = 0
95      for i in df.index:
96          if df.loc[i, 'active_dummy'] == df.loc[i, 'active_predicted']:
97              ratio +=1
98      ratio = ratio/len(df)
99      return ratio
100
101
102 def display_scores(scores):
103     print('Scores: ', scores)
104     print('Mean: ', scores.mean())
105     print('Standard deviation: ', scores.std())
106
107
108
109 ### cerating proper data DataFrames ###
110 def create_data1(df):
111     df = df.query('Studienjahr == 1')[features_year1].dropna().
            reset_index(drop = True).copy(deep = True)
112     split = StratifiedShuffleSplit(n_splits = 1, test_size = 0.1,
            random_state = 42)
113     for train_index, test_index in split.split(df, df['active_dummy'])
            :
114         df_train = df.loc[train_index, :]
115         df_test = df.loc[test_index, :]
116     df_train_copy = df_train.copy(deep = True)
117     y_train = list(df_train['ECTS_year'])
118     y_test = list(df_test['ECTS_year'])
119     df_train = scaler.fit_transform(df_train.drop(['ECTS_year', '
            active_dummy'], axis = 1))
120     df_test = scaler.fit_transform(df_test.drop(['ECTS_year', '
            active_dummy'], axis = 1))
121     return df_train, y_train, df_test, y_test, df_train_copy
122
123 def create_data2(df):
124     df = df.query('Studienjahr > 1')[features_years].dropna().
            reset_index(drop = True).copy(deep = True)
```

```
125     split = StratifiedShuffleSplit(n_splits = 1, test_size = 0.1,
            random_state = 42)
126     for train_index, test_index in split.split(df, df['active_dummy'])
            :
127         df_train = df.loc[train_index, :]
128         df_test = df.loc[test_index, :]
129     df_train_copy = df_train.copy(deep = True)
130     y_train = list(df_train['ECTS_year'])
131     y_test = list(df_test['ECTS_year'])
132     df_train = scaler.fit_transform(df_train.drop(['ECTS_year', '
            active_dummy'], axis = 1))
133     df_test = scaler.fit_transform(df_test.drop(['ECTS_year', '
            active_dummy'], axis = 1))
134     return df_train, y_train, df_test, y_test, df_train_copy
135
136
137
138
139 ### training of various models ###
140 def training_regression(df_train, y_train, df_train_copy, model): #
        df_train is already scaled.
141     model = sklearn.base.clone(model)
142     model.fit(df_train, y_train)
143
144     predictions = cross_val_predict(model, df_train, y_train, cv=3)
145     rmse = np.sqrt(mean_squared_error(y_train, predictions))
146     mae = mean_absolute_error(y_train, predictions)
147     R_squared = r2_score(y_train, predictions)
148
149     print('RMSE = ' + str(rmse))
150     print('MAE = ' + str(mae))
151     print('R2_score = ' + str(R_squared))
152     print('----------')
153     df_new = active(df_train_copy, predictions)
154     print('Ratio = ' + str(ratio(df_new)))
155     scores = cross_val_score(model, df_train, y_train, scoring = '
            neg_mean_squared_error', cv = 5)
156     rmse_scores = np.sqrt(-scores)
157     display_scores(rmse_scores)
158     print('###########')
159
160     return model
161
162
163 ### generating data ###
164
165 # Year 1 :
```

```
df1_train, y1_train, df1_test, y1_test, df1_train_copy = create_data1(df)

# Year >= 2:
df2_train, y2_train, df2_test, y2_test, df2_train_copy = create_data2(df)




### linear Regression ###
print('Linear Regression')
lin_reg = LinearRegression()

# Year 1:

reg1 = training_regression(df1_train, y1_train, df1_train_copy, lin_reg)

# Year >= 2:

reg2 = training_regression(df2_train, y2_train, df2_train_copy, lin_reg)

print('')



### Support Vector Machine ###
# Hyperparameters:
# kernel: linear, polynomial, gaussian or sigmoid. Uses the kernel trick to compute additional features.

# gamma: if the model ist underfitting I should increase it. If the model is overfitting I should decrease it.

# C: is a parameter which controls the margin violations. Higher more violations, but generalizes better.

# epsilion (only for regression): controls the width of the "street". Similar to C. However, I don't get the difference.

print('SVM')
svm_reg = SVR(kernel = 'rbf', gamma = 10, C = 100, epsilon = 5)

# Year 1:
svm1 = training_regression(df1_train, y1_train, df1_train_copy, svm_reg)
```

```
205
206 # Year >=2:
207 svm2 = training_regression(df2_train, y2_train, df2_train_copy,
        svm_reg)
208 print('')
209
210
211
212 ### Random Forest ###
213 # Hyperparameters:
214 # n_estimators: number of Decsiontrees.
215
216 # max_depths: maximal depth of each tree.
217
218 # max_leaf_nodes: controls the depths of the trees.
219
220 # criterion: mse or msa. function to measure the quality of the split.
221
222 # max_samples: (bootstrap = True) How many samples from df_train are
        drawn to train each regressor.
223
224 print('Random Forest')
225 forest_reg = RandomForestRegressor(
226 #             n_estimators = 500, max_depth = 150, max_leaf_nodes =
        100, criterion = 'mae',
227 #             max_samples = 500
228         )
229 # Year 1:
230 forest1 = training_regression(df1_train, y1_train, df1_train_copy,
        forest_reg)
231
232 # Year >= 2:
233 forest2 = training_regression(df2_train, y2_train, df2_train_copy,
        forest_reg)
234 print('')
235
236
237
238 ### training of ANN ###
239
240 print('KNN')
241 def training_ann_regression(df_train, y_train, df_train_copy):
242     model = keras.models.Sequential([
243         keras.layers.Dense(60, activation = 'relu', input_shape =
            df_train.shape[1:]),
244         keras.layers.Dense(40, activation = 'relu'),
245         keras.layers.Dense(20, activation = 'relu'),
246         keras.layers.Dense(1, activation = 'relu')
```

```
    ])
    model.compile(loss = 'huber', optimizer = 'sgd')
    history = model.fit(df_train, np.array(y_train), epochs = 35,
        validation_split = 0.1)

    predictions = model.predict(df_train)
    rmse = np.sqrt(mean_squared_error(y_train, predictions))
    mae = mean_absolute_error(y_train, predictions)
    R_squared = r2_score(y_train, predictions)

    print('RMSE = ' + str(rmse))
    print('MAE = ' + str(mae))
    print('R2_score = ' + str(R_squared))
    print('----------')
    df_new = active(df_train_copy, predictions)
    print('Ratio = ' + str(ratio(df_new)))
    print('##########')

    return model, history

### Year 1 ###

ann1, history1 = training_ann_regression(df1_train, y1_train,
    df1_train_copy)

pd.DataFrame(history1.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 25)
plt.show()


### Year 2 ###

ann2, history2 = training_ann_regression(df2_train, y2_train,
    df2_train_copy)

pd.DataFrame(history2.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 25)
plt.show()
```

A.4 P1 Ansatz 2

Code A.4: P1_A2.py

```
1
2 ### importing libraries ###
3
4 import sys
5 import os
6 import random as rd
7
8 import numpy as np
9 import pandas as pd
10 import matplotlib.pyplot as plt
11
12
13 data_path = '../raw_data'
14 cwd = os.path.dirname(__file__)
15 raw_data_folder = os.path.abspath(os.path.join(cwd, data_path))
16
17
18 def load_df(name):
19     return pd.read_csv(os.path.join(raw_data_folder, name))
20
21 df = load_df('adapted_data.csv').drop(['Unnamed: 0'], axis =1)
22
23 # copy, for doubled usage afterwards.
24 df1 = df.copy(deep = True)
25
26
27 features = [
28     'year','Studienjahr', 'active_year_before',
29     'geschlecht', 'status_year_before',
30     'country', 'school',
31     'subject', 'active_dummy',
32     'status_key'
33     ]
34
35
36
37 ### helper Functions ###
38
39 def active_year_before(df):
40     list_studienjahre = list(df['year'].unique())
41     list_studienjahre.sort()
42     for i in list_studienjahre[1:]:
43         df_year = df.query('year == @i')
44         df_year0 = df.query('year == (@i-1)')
45         for num in df_year.index:
46             if df_year.loc[num, 'Studienjahr'] > 1:
47                 matrikel = df_year.loc[num, 'matrikel_num']
```

```
                if (num-1) in df_year0.index and df_year0.loc[(num-1),
                    'matrikel_num'] == matrikel:
                    df.loc[num, 'active_year_before'] = df.loc[(num-1)
                        , 'active_dummy']
                    df.loc[num, 'status_year_before'] = df.loc[(num-1)
                        , 'status_key']

    return df

def country(df):
    for i in range(len(df)):
        if df.loc[i, 'styria_dummy'] == 1:
            df.loc[i, 'country'] = 0
        elif df.loc[i, 'not_styria_dummy'] == 1:
            df.loc[i, 'country'] = 1
        elif df.loc[i, 'germany_dummy'] == 1:
            df.loc[i, 'country'] = 2
        else:
            df.loc[i, 'country'] = 3
    return df

def school(df):
    for i in range(len(df)):
        if df.loc[i, 'AHS_dummy'] == 1:
            df.loc[i, 'school'] = 0
        elif df.loc[i, 'BHS_dummy'] == 1:
            df.loc[i, 'school'] = 1
        else:
            df.loc[i, 'school'] = 2
    return df

def subject(df):
    for i  in range(len(df)):
        if df.loc[i, 'jus_dummy'] == 1:
            df.loc[i, 'subject'] = 0
        elif df.loc[i, 'bwl_dummy'] == 1:
            df.loc[i, 'subject'] = 1
        else:
            df.loc[i, 'subject'] = 2
    return df


### applying them ###
active_year_before(df)
country(df)
school(df)
subject(df)

```

```
df_working = df[features]

### helper Function for creating all combinations ###

def create_combinations():
    return_list = []
    for a in range(2):
        for b in range(3):
            for c in range(4):
                for d in range(3):
                    return_list.append([a,b,c,d])
    return return_list

numerical_combinations = create_combinations()
print(len(numerical_combinations))

def split(df, combinations):
    return_list = []
    for i in combinations:
        geschlecht = i[0]
        subject = i[1]
        country = i[2]
        school = i[3]

        df_append = df.query('geschlecht == @geschlecht and subject ==
            @subject and country == @country and school == @school')
        return_list.append((i, df_append))
    return return_list



# setting concrete year of study: in this case 2 #
df_list = split(df_working.query('Studienjahr == 2'),
   numerical_combinations)


### helper Functions for calculating probabilities ###

def prob_year(df): # returns a list with probabilities
    a = len(df.query("active_dummy == 1 and status_key != 'I'"))
    b = len(df.query("active_dummy == 0 and status_key != 'I'"))
    c = len(df.query("active_dummy == 1 and status_key == 'I'"))
    d = len(df.query("active_dummy == 0 and status_key == 'I'"))

    denominator = len(df)
    if denominator > 0:
        return [round(a/denominator, 2), round(b/denominator,2), round
           (c/denominator,2),  round(d/denominator, 2)]
```

```
    else:
        return "Zero"

def print_matrix(df):
    df_active = df.query('active_year_before == 1')
    df_inactive = df.query('active_year_before == 0')

    print(prob_year(df_active))
    print(prob_year(df_inactive))
    print('---------')



### calculating probability for 1st Year for the first class (0,0,0,0)
# This is: weiblich, Steiermark, JUS, AHS Vorbildung.

print('--------')
print(prob_year(df_working.query('Studienjahr == 1 and subject == 0
    and country == 0 and geschlecht == 0 and school == 0')))



### calculating all probabilities for all classes for year 2 ###

for i in df_list:
    print(i[0]) # combination
    print(len(i[1]))
    print_matrix(i[1])
```

A.5 P1 Ansatz 3

Code A.5: P1_A3.py

```


### importing libraries ###


import os
import random as rd

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
12
13 import sklearn
14 from sklearn.metrics import mean_squared_error
15 from sklearn.model_selection import StratifiedShuffleSplit
16
17 from sklearn.preprocessing import StandardScaler
18 from sklearn.model_selection import cross_val_predict
19 from sklearn.model_selection import cross_val_score
20 from sklearn.metrics import confusion_matrix, precision_score,
       recall_score, f1_score, precision_recall_curve
21 scaler = StandardScaler()
22
23 import tensorflow as tf
24 from tensorflow import keras
25
26
27 data_path = '../raw_data'
28 cwd = os.path.dirname(__file__)
29 raw_data_folder = os.path.abspath(os.path.join(cwd, data_path))
30
31
32 def load_df(name):
33     return pd.read_csv(os.path.join(raw_data_folder, name))
34
35
36
37 # loaging data
38 df_working = load_df('adapted_data.csv').drop(['Unnamed: 0'], axis =1)
39
40 df_test = df_working.copy(deep = True)
41
42 # features for Year = 1
43 features_year1 = [
44     'styria_dummy', 'not_styria_dummy',
45     'germany_dummy', 'num_parallel_studies',
46     'years_since_matura', 'firstGen',
47     'geschlecht', 'AHS_dummy',
48     'BHS_dummy', 'ausland_vorbildung_dummy',
49     'sonstige_vorbildung_dummy', 'jus_dummy',
50     'bwl_dummy', 'delayed_dummy',
51     'active_3years', 'ECTS_year',
52     'active_dummy'
53     ]
54
55
56 # features for Year >= 2
57 features_years = [
58     'Studienjahr', 'styria_dummy',
```

```
    'not_styria_dummy', 'germany_dummy',
    'num_parallel_studies', 'cum_ects_pos_before',
    'avgECTS_sem_before', 'ects_year_before',
    'full_duration_sem_before', 'geschlecht',
    'years_since_matura', 'firstGen',
    'AHS_dummy', 'BHS_dummy',
    'ausland_vorbildung_dummy', 'sonstige_vorbildung_dummy',
    'delayed_dummy', 'jus_dummy',
    'bwl_dummy', 'active_3years',
    'ECTS_year', 'active_dummy'
    ]



### helper Function, for getting labels of 3 Years in the future ###

def active_3years(df):
    count = 0

    for i in range(len(df)):

        matrikel_number = df.loc[i, 'matrikel_num']
        studienjahr = df.loc[i, 'Studienjahr'] + 2
        jus_dummy = df.loc[i, 'jus_dummy']
        bwl_dummy = df.loc[i, 'bwl_dummy']

        student_in_3_years = df.query('matrikel_num ==
           @matrikel_number and Studienjahr == @studienjahr and
           jus_dummy == @jus_dummy and bwl_dummy == @bwl_dummy').
           reset_index(drop = True)

        if len(student_in_3_years) != 0:
            count += 1
            active = student_in_3_years.loc[0, 'active_dummy']
            df.loc[i, 'active_3years'] = active
        else:
            df.loc[i, 'active_3years'] = 0


    # print(count)
    return df


df_new = active_3years(df_test).query('year <= 17')



### helper Functions for creating appropriate data ###
```

```
104
105 # Year 1
106 def create_data1(df):
107     df = df.query('Studienjahr == 1')[features_year1].copy(deep = True
        )
108     df = df.dropna().reset_index(drop = True)
109     split = StratifiedShuffleSplit(n_splits = 1, test_size = 0.1,
        random_state = 42)
110     for train_index, test_index in split.split(df, df['active_3years'
        ]):
111         df_train = df.loc[train_index, :]
112         df_test = df.loc[test_index, :]
113     df_train_copy = df_train.copy(deep = True)
114     y_train = list(df_train['active_3years'])
115     y_test = list(df_test['active_3years'])
116     df_train = scaler.fit_transform(df_train.drop(['active_3years', '
        ECTS_year', 'active_dummy'], axis = 1))
117     df_test = scaler.fit_transform(df_test.drop(['active_3years', '
        ECTS_year', 'active_dummy'], axis = 1))
118     return df_train, y_train, df_test, y_test, df_train_copy
119
120
121 # Years >= 2
122 def create_data2(df):
123     df = df.query('Studienjahr > 1')[features_years].copy(deep = True)
124     df = df.dropna().reset_index(drop = True)
125     split = StratifiedShuffleSplit(n_splits = 1, test_size = 0.1,
        random_state = 42)
126     for train_index, test_index in split.split(df, df['active_3years'
        ]):
127
128         df_train = df.loc[train_index, :]
129         df_test = df.loc[test_index, :]
130     df_train_copy = df_train.copy(deep = True)
131     y_train = list(df_train['active_3years'])
132     y_test = list(df_test['active_3years'])
133     df_train = scaler.fit_transform(df_train.drop(['active_3years', '
        ECTS_year', 'active_dummy'], axis = 1))
134     df_test = scaler.fit_transform(df_test.drop(['active_3years', '
        ECTS_year', 'active_dummy'], axis = 1))
135     return df_train, y_train, df_test, y_test, df_train_copy
136
137
138 # concrete creation of data:
139 df1_train, y1_train, df1_test, y1_test, df1_train_copy = create_data1(
    df_new)
140 df2_train, y2_train, df2_test, y2_test, df2_train_copy = create_data2(
    df_new)
```

```
141
142
143
144 ### helper Functions for training and creating results ###
145
146 def display_scores(scores):
147     print('Scores: ', scores)
148     print('Mean: ', scores.mean())
149     print('Standard deviation: ', scores.std())
150
151
152 def perform_classification(classifier, df, labels):
153     df = df.copy()
154     print('#####')
155     scores = cross_val_score(classifier, df, labels, cv=5, scoring='
            accuracy')
156     display_scores(scores)
157
158     predicted = cross_val_predict(classifier, df, labels, cv=5)
159     matrix = confusion_matrix(labels, predicted)
160     print(matrix)
161
162     plt.show()
163     row_sums = matrix.sum(axis=1, keepdims=True)
164     norm_matrix = matrix / row_sums
165     np.fill_diagonal(norm_matrix, 0)
166
167     plt.show()
168     print('#####')
169     return
170
171
172 def probability_classification(classifier, df_train, y_train, df_test,
        y_test):
173     classifier = sklearn.base.clone(classifier)
174     classifier.fit(df_train, y_train)
175     probabilities = classifier.predict_proba(df_test)[:,1]
176     prediction = np.sum(probabilities)
177     real_value = np.sum(y_test)
178     print('Prediction: ' + str(prediction) + ' vs. Real value: ' + str
            (real_value))
179
180
181
182 ### Logistic Regression ###
183 from sklearn.linear_model import LogisticRegression
184
185 print('Logistic Regression')
```

```
186 log_clf1 = LogisticRegression(random_state=0, multi_class = '
    multinomial')
187 log_clf2 = LogisticRegression(random_state=0, multi_class = '
    multinomial')
188
189
190 probability_classification(log_clf1, df1_train, y1_train, df1_test,
    y1_test)
191 probability_classification(log_clf2, df2_train, y2_train, df2_test,
    y2_test)
192
193 perform_classification(log_clf1, df1_train, y1_train)
194 perform_classification(log_clf2, df2_train, y2_train)
195 print('')
196
197
198
199 ### Support Vector Machines ###
200
201 from sklearn.svm import SVC
202
203 svm_clf1 = SVC(kernel = 'rbf', gamma = 10, C = 100, probability = True
    )
204 svm_clf2 = SVC(kernel = 'rbf', gamma = 10, C = 100, probability = True
    )
205 print('')
206 print('SVM')
207 probability_classification(svm_clf1, df1_train, y1_train, df1_test,
    y1_test)
208 probability_classification(svm_clf2, df2_train, y2_train, df2_test,
    y2_test)
209
210 perform_classification(svm_clf1, df1_train, y1_train)
211 perform_classification(svm_clf2, df2_train, y2_train)
212 print('')
213
214
215
216 ### Random Forest ###
217
218 from sklearn.ensemble import RandomForestClassifier
219
220 forest_clf1 = RandomForestClassifier()
221 forest_clf2 = RandomForestClassifier()
222
223 print('')
224 print('Random Forest')
```

```
probability_classification(forest_clf1, df1_train, y1_train, df1_test, y1_test)
probability_classification(forest_clf2, df2_train, y2_train, df2_test, y2_test)

perform_classification(forest_clf1, df1_train, y1_train)
perform_classification(forest_clf2, df2_train, y2_train)
print('')



### KNN ###

print('KNN')
model1 = keras.models.Sequential([
        keras.layers.Dense(60, activation = 'relu', input_shape = df1_train.shape[1:]),
        keras.layers.Dense(40, activation = 'relu'),
        keras.layers.Dense(20, activation = 'relu'),
        keras.layers.Dense(1, activation = 'sigmoid')
    ])

model1.compile(loss = 'binary_crossentropy', optimizer = 'sgd', metrics = ['accuracy'])
history1 = model1.fit(df1_train, np.array(y1_train), epochs = 35, validation_split = 0.1)

predictions1 = model1.predict(df1_test)
predict1 = np.sum(predictions1)
real1 = np.sum(y1_test)
print('Prediction: ' + str(predict1) + ' vs. Real value: ' + str(real1))


model2 = keras.models.Sequential([
        keras.layers.Dense(60, activation = 'relu', input_shape = df2_train.shape[1:]),
        keras.layers.Dense(40, activation = 'relu'),
        keras.layers.Dense(20, activation = 'relu'),
        keras.layers.Dense(1, activation = 'sigmoid')
    ])
model2.compile(loss = 'binary_crossentropy', optimizer = 'sgd', metrics = ['accuracy'])
history2 = model2.fit(df2_train, np.array(y2_train), epochs = 35, validation_split = 0.1)

predictions2 = model2.predict(df2_test)
predict2 = np.sum(predictions2)
```

```
real2 = np.sum(y2_test)
print('Prediction: ' + str(predict2) + ' vs. Real value: ' + str(real2))
```

A.6 P2 Legitimierung

Code A.6: P2.py

```


### importing libraries ###


import os
import random as rd

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import StratifiedShuffleSplit

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, precision_recall_curve
scaler = StandardScaler()

### loading data ###
data_path = '../raw_data'
cwd = os.path.dirname(__file__)
raw_data_folder = os.path.abspath(os.path.join(cwd, data_path))


def load_df(name):
    return pd.read_csv(os.path.join(raw_data_folder, name))


features_year1 = [
    'styria_dummy', 'not_styria_dummy',
    'germany_dummy', 'num_parallel_studies',
    'years_since_matura', 'firstGen',
```

```
    'geschlecht', 'AHS_dummy',
    'BHS_dummy', 'ausland_vorbildung_dummy',
    'sonstige_vorbildung_dummy', 'jus_dummy',
    'bwl_dummy', 'delayed_dummy',
    'active_3years', 'ECTS_year',
    'active_dummy'
    ]

features_years = [
    'Studienjahr', 'styria_dummy',
    'not_styria_dummy', 'germany_dummy',
    'num_parallel_studies', 'cum_ects_pos_before',
    'avgECTS_sem_before', 'ects_year_before',
    'full_duration_sem_before', 'geschlecht',
    'years_since_matura', 'firstGen',
    'AHS_dummy', 'BHS_dummy',
    'ausland_vorbildung_dummy', 'sonstige_vorbildung_dummy',
    'delayed_dummy', 'jus_dummy',
    'bwl_dummy', 'active_3years',
    'ECTS_year', 'active_dummy'
    ]


df = load_df('adapted_data.csv').drop(['Unnamed: 0'], axis =1)



### helper Function for creating new labels ###

def active_2years(df):
    df['active_3years'] = 0
    for i in range(len(df)):

        matrikel_number = df.loc[i, 'matrikel_num']
        studienjahr = df.loc[i, 'Studienjahr'] + 1
        jus_dummy = df.loc[i, 'jus_dummy']
        bwl_dummy = df.loc[i, 'bwl_dummy']

        student2 = df.query('matrikel_num == @matrikel_number and
           Studienjahr == @studienjahr and jus_dummy == @jus_dummy
           and bwl_dummy == @bwl_dummy').reset_index(drop = True)

        if len(student2) != 0:
            active = student2.loc[0, 'active_dummy']
            df.loc[i, 'active_2years'] = active

    return df

```

```
### making new DataFrames ###

df_2years = active_2years(df)
df_2years['active_2years'].fillna(0)


df = load_df('adapted_data.csv').drop(['Unnamed: 0'], axis =1)
df_1year = df.copy(deep = True)
df_1year['active_dummy'].fillna(0)



### New Training Data ###

df2years = df_2years[features2years].query('Studienjahr == 1').
    reset_index(drop = True).dropna()
df1year = df_1year[features1year].query('Studienjahr == 1').
    reset_index(drop = True).dropna()


df_train2 = df2years.drop(['active_2years'], axis = 1)
y_train2 = df2years['active_2years']

df_train1 = df1year.drop(['active_dummy'], axis = 1)
y_train1 = df1year['active_dummy']


### choosing SVM Classifier ###
from sklearn.svm import SVC
svm_clf1year = SVC(kernel = 'rbf', gamma = 10, C = 100, probability =
    True)
svm_clf2years = SVC(kernel = 'rbf', gamma = 10, C = 100, probability =
     True)

svm_clf1year.fit(df_train1, y_train1)
svm_clf2years.fit(df_train2, y_train2)


### making real and dummy DataFrames ###

df1_15 = df_1year.query('Studienjahr == 1 and year == 15')[
    features1year].drop(['active_dummy'], axis = 1).fillna(0)
df1_16 = df_1year.query('Studienjahr == 1 and year == 16')[
    features1year].drop(['active_dummy'], axis = 1).fillna(0)
df1_17 = df_1year.query('Studienjahr == 1 and year == 17')[
    features1year].drop(['active_dummy'], axis = 1).fillna(0)

df1_dummy16 = df1_15.sample(n = 2048)
```

```
df1_dummy17 = df1_16.sample(n = 1899)


### helper Function for making Prediction ###

def predict_sum(classifier, df):
    probabilities = classifier.predict_proba(df)[:,1]
    for i in range(len(probabilities)):
        num = rd.random()
        if probabilities[i] >= num:
            probabilities[i] = 1
        else:
            probabilities[i] = 0
    print(sum(probabilities))
    print('######')



### Making actually Predictions ###

print('Schaetzung echte Daten: ')
predict_sum(svm_clf1year, df1_16)
print('Schaetzung dummy Daten: ')
predict_sum(svm_clf1year, df1_dummy16)

print('Tatsaechliche Anzahl: ')
print(len(df.query('Studienjahr == 1 and year == 16 and active_dummy
    == 1')))
print('')

print('Schaetzung echte Daten: ')
predict_sum(svm_clf1year, df1_17)
print('Schaetzung dummy Daten: ')
predict_sum(svm_clf1year, df1_dummy17)

print('Tatsaechliche Anzahl: ')
print(len(df.query('Studienjahr == 1 and year == 17 and active_dummy
    == 1')))
print('')




### data for estimation over 2 years ###

df2_15 = df_2years.query('Studienjahr == 1 and year == 15 and
    active_2years >= 0')[features2years].drop(['active_2years'], axis
    = 1).fillna(0)
```

```
168 df2_16 = df_2years.query('Studienjahr == 1 and year == 16 and
        active_2years >= 0')[features2years].drop(['active_2years'], axis
        = 1).fillna(0)
169 df2_17 = df_2years.query('Studienjahr == 1 and year == 17 and
        active_2years >= 0')[features2years].drop(['active_2years'], axis
        = 1).fillna(0)
170
171 df2_dummy16 = df2_15.sample(n = 1421)
172 df2_dummy17 = df2_16.sample(n = 1249)
173
174
175 ### Making actually Predicitions ###
176
177 print('Schaetzung echte Daten: ')
178 predict_sum(svm_clf2years, df2_16)
179 print('Schaetzung dummy Daten: ')
180 predict_sum(svm_clf2years, df2_dummy16)
181
182 print('Tatsaechliche Anzahl: ')
183 print(len(df.query('Studienjahr == 2 and year == 17 and active_dummy
        == 1')))
184 print('')
185
186 print('Schaetzung echte Daten: ')
187 predict_sum(svm_clf2years, df2_17)
188 print('Schaetzung dummy Daten: ')
189 predict_sum(svm_clf2years, df2_dummy17)
190
191 print('Tatsaechliche Anzahl: ')
192 print(len(df.query('Studienjahr == 2 and year == 18 and active_dummy
        == 1')))
193 print('')
```