HAW HAMBURG

BACHELORTHESIS
Quang Huy Nguyen

Künstliche Intelligenz in der Kompetenzentwicklung zur Steigerung des Studienerfolgs

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Quang Huy Nguyen

Künstliche Intelligenz in der Kompetenzentwicklung zur Steigerung des Studienerfolgs

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Marina Tropmann-Frick
Zweitgutachter: Prof. Dr. Martin Schultz

Eingereicht am: 24.05.2022

Quang Huy Nguyen

Thema der Arbeit

Künstliche Intelligenz in der Kompetenzentwicklung zur Steigerung des Studienerfolgs

Stichworte

Kompetenzerfassung, Kompetenz, Klassifikationsalgorithmen, Kompetenzorientierung, Informatikstudiengang, maschinellen Lernens

Kurzzusammenfassung

Die Vorhersagefähigkeit eines Studiengangs auf der Grundlage der damit verbundenen Kompetenzen ist eine entscheidende Komponente eines KI-Systems, das weitere Empfehlungen für Studierende und fundierte Entscheidungen für Lehrkräfte und Tutoren liefern kann. In dieser Arbeit wird ein intelligentes Vorhersagemodell entwickelt, das den Informatikstudiengang eines immatrikulierten Studenten auf der Grundlage von Kompetenzdaten dieses Studenten prognostizieren kann. Die verwendeten Daten stammen aus den Modulhandbüchern des Departments Informatik der HAW. Es werden verschiedene Klassifikationsalgorithmen des maschinellen Lernens eingesetzt und evaluiert.

Quang Huy Nguyen

Title of Thesis

Artificial intelligence in competency development to improve student success

Keywords

Competence assessment, competence, classification algorithms, competence orientation, Computer Science Program, machine learning

Abstract

The predictive capability of a course of study based on its associated competencies is a critical component of an AI system that can provide further recommendations for students and informed decisions for teachers and tutors. In this work, an intelligent prediction model is developed that can predict the computer science course of an enrolled

student based on competency data of that student. The data used is taken from the module handbooks in Department of Computer Science from HAW. Different classification algorithms in machine learning are used and evaluated.

Inhaltsverzeichnis

1 Einleitung

Online-Studierendenbefragungen der HAW Hamburg und bundesweite Studien (Marczuk et al. 2021, Zimmer et.al. 2021) haben gezeigt, dass die Digitalisierung von Studium und Lehre ebenso als Katalysator für die Verstärkung von Bildungsungerechtigkeit/Benachteiligungen wirken kann, wie auch als Chance, KI-basierte Techniken zur effektiven und effizienten Förderung des Studienerfolgs zu nutzen. Um technische Verfahren der künstlichen Intelligenz nachhaltig einsetzen zu können, um sowohl den Studienerfolg als auch die Qualität der Hochschulbildung zu steigern, sollen drei Themenbereiche abgedeckt werden: Bildungswissenschaften, Datenwissenschaften und Kognitionspsychologie. Methoden der Kompetenzerfassung werden mit technischen Verfahren der Künstlichen Intelligenz kombiniert. Ziel ist die Entwicklung einer intelligenten dynamischen adaptiven KI-Lernplattform mit E-Portfolios als formativem lernprozessintegriertem Bewertungsformat. Dieses selbstlernende technische System verarbeitet die Daten der Kompetenzfeststellung und verknüpft sie mit dem individuellen Bildungsprozess des Schülers, um weitere Empfehlungen für die Schüler und fundierte Entscheidungen für Lehrer und Tutoren anzubieten. Der erste Baustein dieses intelligenten Systems ist die Fähigkeit, einen Studiengang auf der Grundlage vordefinierter Kompetenzen korrekt zu identifizieren oder genauer zu klassifizieren, d.h. die Herausforderung dieser Arbeit besteht in Textklassifikationsproblemen.

Textklassifizierungsprobleme wurden in den letzten Jahrzehnten umfassend untersucht und in vielen realen Anwendungen behandelt. Insbesondere mit den jüngsten Durchbrüchen in Natural Language Processing (NLP) und Text Mining sind viele Forscher nun daran interessiert, Anwendungen zu entwickeln, die Textklassifizierungsmethoden nutzen. Das in dieser Arbeit entwickelte System zur Dokumentenkategorisierung lässt sich in die folgenden vier Phasen unterteilen: Datenbereinigung, Merkmalsextraktion, Klassifikatorauswahl und Auswertung.

Die vorliegende Arbeit ist in fünf Kapitel unterteilt. In Kapitel 2 wird zunächst eine Beschreibung der vorliegenden Problemstellung gegeben. Weiterhin wird ein Einblick in die Grundlagen der Aufbereitung, Verarbeitung und Auswertung von Textdaten gegeben.

Kapitel 3 ist der praktischen Umsetzung der genannten Techniken und Algorithmen gewidmet. Besonderes Augenmerk wird dabei auf die Effizienz des jeweiligen Modells bei der korrekten Erkennung aller Domains gelegt.
Kapitel 4 wertet die Ergebnisse und Beobachtungen aus den zuvor beschriebenen Experimenten aus. Darüber hinaus werden Probleme genannt, die in dieser Arbeit identifiziert und beseitigt wurden.
Kapitel 5 schließt die Arbeit mit einem Fazit ab. Zusätzlich wird ein Ausblick auf mögliche Weiterentwicklungen des Zielsystems gegeben, die im Anschluss an diese Arbeit vorgenommen werden könnten.

2 Grundlage

Algorithmen des maschinellen Lernens wurden in der Vergangenheit häufig zur Textklassifizierung eingesetzt. Das folgende Kapitel befasst sich mit der Einführung der entsprechenden Technologie.
Das Unterkapitel 2.1 enthält eine detaillierte Beschreibung der Problemstellung, die in dieser Arbeit gelöst werden soll. Es folgt eine Vorstellung des aktuellen Forschungsstandes im Bereich des Maschinellen Lernens und Natural Language Processings in den Abschnitten 2.2 und 2.3. Schließlich wird in Abschnitt 2.4 die Konfusionsmatrix, die Methode zur späteren Darstellung des Ergebnisses eines Modells, spezifiziert.

2.1 Problemstellung

2.1.1 Qualifikationsrahmens für Deutsche Hochschulabschlüsse (HQR)

Ein Qualifikationsrahmen ist eine systematische Beschreibung der Qualifikationen, die das Bildungssystem eines Landes hervorbringt. Diese Beschreibung umfasst:

- eine allgemeine Darstellung des Qualifikationsprofils eines Absolventen, der den zugeordneten Abschluss besitzt,
- eine Auflistung der angestrebten Lernergebnisse,
- eine Beschreibung der Kompetenzen und Fertigkeiten, über die der Absolvent verfügen sollte,
- eine Beschreibung der formalen Aspekte eines Ausbildungslevels (Arbeitsumfang in ECTS Credits, Zulassungskriterien, Bezeichnung der Abschlüsse, formale Berechtigungen).

Der Qualifikationsrahmen für deutsche Hochschulabschlüsse ist ein Transparenzinstrument und dient der Verbesserung der nationalen und internationalen Vergleichbarkeit von Bildungsangeboten. Er bietet Informationen für Studieninteressierte und Arbeitgeber, unterstützt die Evaluation und Akkreditierung und erleichtert die Lehrplanentwicklung [24].

2.1.2 Kompetenz

Der HQR folgt einem weit verbreiteten Konzept der Handlungskompetenz, das von Weinert 2001 formuliert wurde. Demnach werden Kompetenzen verstanden als

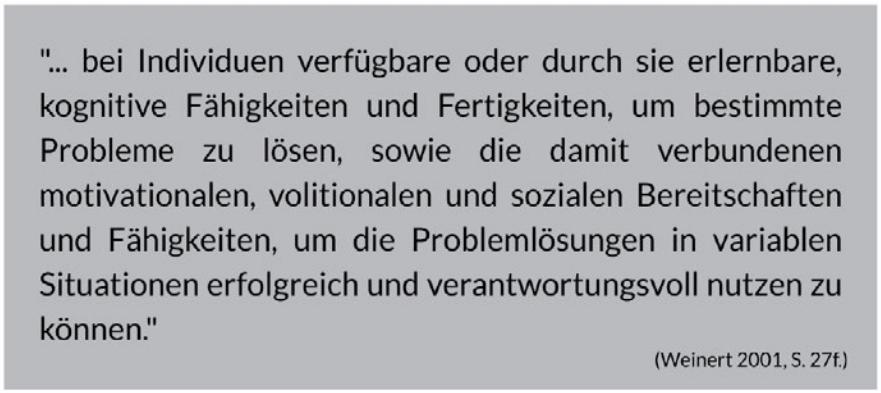


Abbildung 2.1: Definition von Kompetenz durch Weinert [43]

Auf diese Weise werden die Ursprünge des Potenzials einer Person (und ggf. einer Organisation) in zwei Perspektiven benannt. Kognitive und affektive Aspekte spielen ineinander. Erfolgreiches Handeln basiert auf einem mehrdimensionalen Set von Fähigkeiten, die in konkreten Situationen zum Tragen kommen können. Unsere Kompetenz erlaubt es uns, unsichere Herausforderungen mit Zuversicht anzugehen und andere erwarten von uns erfolgreiches Handeln, wenn sie uns Kompetenz zuschreiben.
Kompetenz beschreibt ein Potenzial für die vermutete Fähigkeit einer Person, intentional, zielgerichtet und erfolgreich zu handeln. Kompetenz ist ein Konstrukt, das sich - differenziert nach plausiblen und beobachtbaren Elementen - als multidimensionales Gebilde darstellt. Erst die Definition von Kompetenz im Rahmen eines Kompetenzmodells ermöglicht die Entwicklung von Kriterien für die Beobachtung und Messung von Kompetenz. Vereinfacht ausgedrückt: Was die Kompetenz eines Hochschulabsolventen ausmacht,

kann nur beschrieben und beobachtet werden, wenn vorher definiert ist, was diese Kompetenz ausmachen soll. Die Operationalisierung des Konstrukts Kompetenz im Rahmen eines Modells legt also fest, was gemeinhin unter der Kompetenz dieser Personengruppe verstanden wird. Für alle Vergleichsprozesse mit anderen Personengruppen ist es daher entscheidend, dass die spezifischen Aspekte der Hochschulbildung im Kompetenzmodell für Absolventen zum Ausdruck kommen. Die Abbildung 2.2 bringt einen Überblick über die Beziehung zwischen Kompetenzmodell HQR, Deskriptoren eines Fachqualifikationsrahmen-profil (FQRFQP) und den Lernergebnissen eines Modulhandbuches.

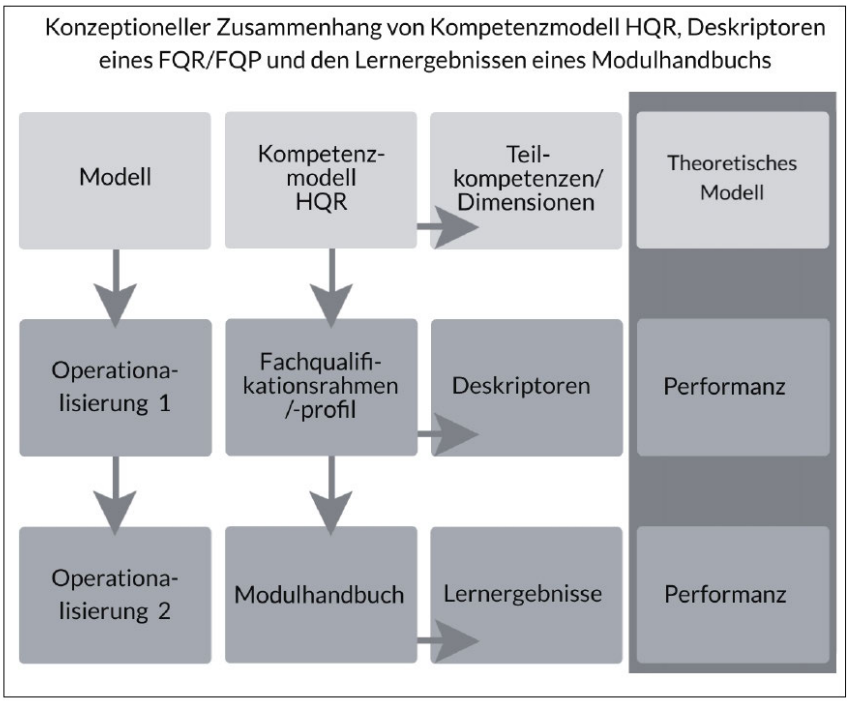


Abbildung 2.2: Konzeptioneller Zusammenhang von Kompetenzmodell HQR, Deskriptoren eines FQR/FQP und den Lernergebnissen eines Modulhandbuches [3]

Als Weiterentwicklung der alten Version, die 2005 unter dem Namen QR_DH veröffentlicht wurde, formuliert der HQR (2017) sein Kompetenzmodell in vier Dimensionen: (1) Wissen und Verstehen; (2) Einsatz, Anwendung und Erzeugung von Wissen; (3) Kommunikation und Kooperation; (4) Wissenschaftliches Selbstverständnis oder Professionalität (siehe Abbildung 2.3).

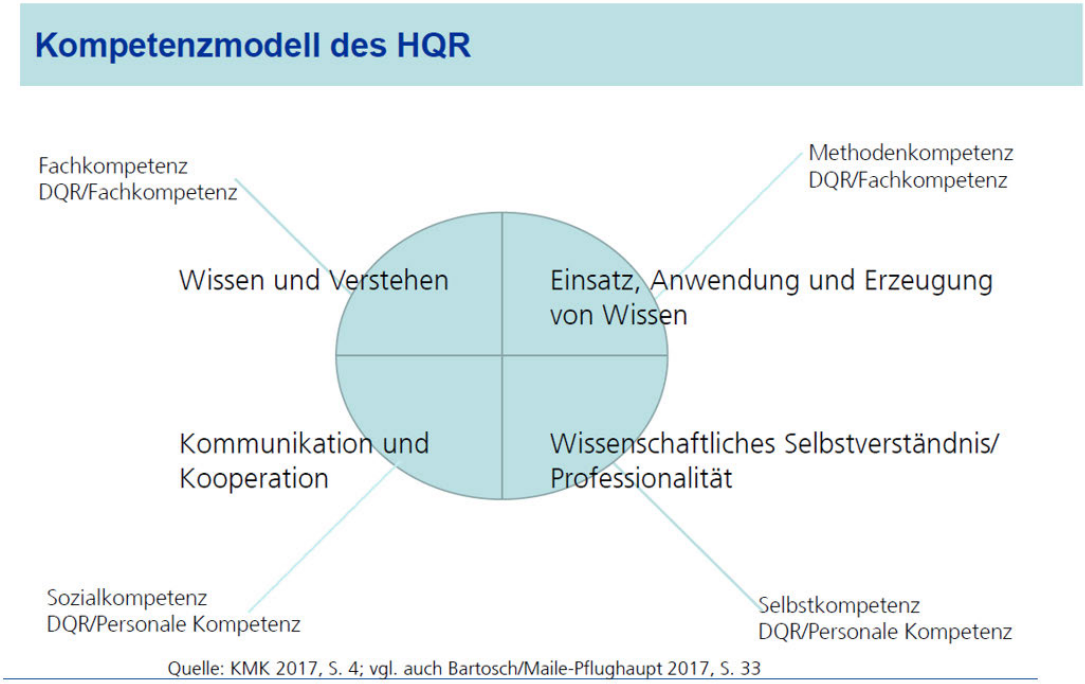


Abbildung 2.3: Kompetenzmodell des HQR. URL

Es geht um die Beschreibung der erwarteten, vorausgesetzten Fähigkeiten einer akademisch gebildeten Person, die sich in vier Fragerichtungen entwickelt:
Was kennzeichnet ihre kognitiven Operationen? Welcher Ansatz zur Problemlösung ist zu erwarten? Wie sind ihre Interaktionen und ihre Teilhabe an sozialen Kontexten gerahmt? Welche Haltung bildet den Rahmen für das berufliche Handeln?

2.1.3 Konstruktion von Studiengängen auf der Grundlage der Qualifikationsprofile

Ein Studiengang kann als ein konkretisierter, exemplarischer Lern-/Lehrraum zur Realisierung der Qualifikationsversprechen des Qualifikationsrahmens verstanden werden. Die Hochschule interpretiert und gestaltet die Deskriptoren (siehe Abbildung 2.2) des HQR in spezifischer Weise und beschreibt dies durch Lernergebnisse in ihren eigenen Dokumenten (z.B. Modulhandbuch). In der konkreten Umsetzung kann der HQR durch einen fachlichen Qualifikationsrahmen im internen Diskurs einer Disziplin präzisiert werden. Dabei werden die Dimensionen des HQR-Kompetenzmodells von der Beschreibung des

Potenzials in eine Beschreibung der Leistung überführt. Es empfiehlt sich, ein Qualifikationsprofil (QP) für den Studiengang zu verfassen, das die gemeinsame Zielperspektive der Hochschule und der beteiligten Lehrenden zum Ausdruck bringt. Es kann als Konkretisierung eines bestehenden Leitbildes für das Lehren und Lernen gestaltet werden. Im QP wird das spezifische Qualifikationsversprechen dieses Studiengangs offengelegt und durch die Zuordnung zu den einzelnen Modulen transparent gemacht.
Mit der differenzierten Gesamtdarstellung des Studiengangs wird ein transparenter Einblick in dessen innere Logik ermöglicht. Gleichwohl bleiben Qualifikationsbeschreibungen Annäherungen, die auf der Grundlage einer abgestimmten Plausibilitätsprüfung erfolgen. Die Abstimmung muss zwischen den verschiedenen beteiligten Akteuren erfolgen. Sie können und dürfen nicht im Sinne technischer oder psychometrischer Größen gelesen werden. Die transparente Darstellung des Studienverlaufs gibt allen Beteiligten vielfältige Möglichkeiten der Unterstützung, aber keine fachliche Anleitung.
Durch die Zuordnung des QPs zu den einzelnen Modulen ergibt sich eine zeitliche Verteilung der erreichten Lernergebnisse über den Studienverlauf und daraus abgeleitet eine Darstellung der schrittweisen Akkumulation des QPs. Das vollständige QP wird als Zielmatrix verwendet.

Qualifikationsprofil für einen Studiengang:

	A Wissen/ Verstehen	B Bewerten/ Analysieren	C Konzipieren	D Forschen/ Recherchieren	E Organisieren/ Durchführen/ Evaluieren
Level Bachelor	Lernergebnis 1 2 3 4	Lernergebnis 1 5 2 6 3 4	Lernergebnis 1 5 2 3 4	Lernergebnis 1 5 2 6 3 7 4	Lernergebnis 1 2 3 4
Level Master	Lernergebnis 1 5 2 6 3 4	Lernergebnis 1 5 2 6 3 7 4	Lernergebnis 1 2 3 4	Lernergebnis 1 5 2 6 3 4	Lernergebnis 1 5 2 6 3 7 4

Abbildung 2.4: Qualifikationsrahmen/-profile als handlungsorientierte Matrix

Das QP kann als eine Kompetenzmatrix symbolisiert werden, in die seine Elemente eingetragen werden (siehe Abbildung 2.4). Jedes Feld sammelt typische Qualifikationsbeschreibungen,

- die als Disposition zur Handlungsfähigkeit ausgedrückt werden (Kompetenzorientierung),
- in der situativen Bewältigung von Anforderungen beobachtbar und überprüfbar sind (Performanz),
- das Qualifikationsniveau in Lernergebnissen angeben (Deskriptor) und
- als Ergebnis der vorangegangenen Lernarbeit angesehen werden können (Lernergebnis).

Diese einzelnen Elemente (Teilkompetenzen) sind Ziel- und Bezugspunkte für jedes Modul. Die Deskriptoren der Lernergebnisse auf Modulebene stützen sich auf das konzeptionelle Vokabular von Wissen und Fertigkeiten, Fähigkeiten und Fertigkeiten, Einstellungen und persönlichen Qualitäten, Kompetenzen und Qualifikationen, seien es Fach- oder Schlüsselqualifikationen oder Kompetenzen. Die Deskriptoren werden durch einzelne, feinkörnige Lernergebnisse innerhalb der Module operationalisiert. Die Lernergebnisse der Module beziehen sich auf die übergeordneten Lernergebnisse (Deskriptoren) des Qualifikationsprofils und sind somit miteinander verknüpft (siehe Abbildung 2.5). Die einzelnen Module werden eine spezifische Auswahl davon als ihre eigenen Zielpunkte bestimmen:

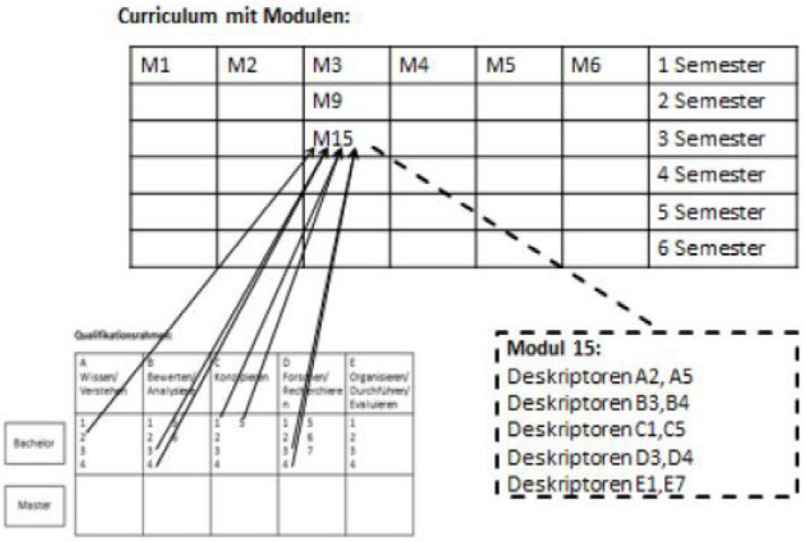


Abbildung 2.5: Module im Zusammenhang mit Deskriptoren

In der Summe werden also alle Deskriptoren des QPs in der Gesamtheit der Module abgebildet. Sie werden dort differenzierter als die Lernergebnisse beschrieben, die dem Deskriptor im konkreten Modul entsprechen (siehe Abbildung 2.6).

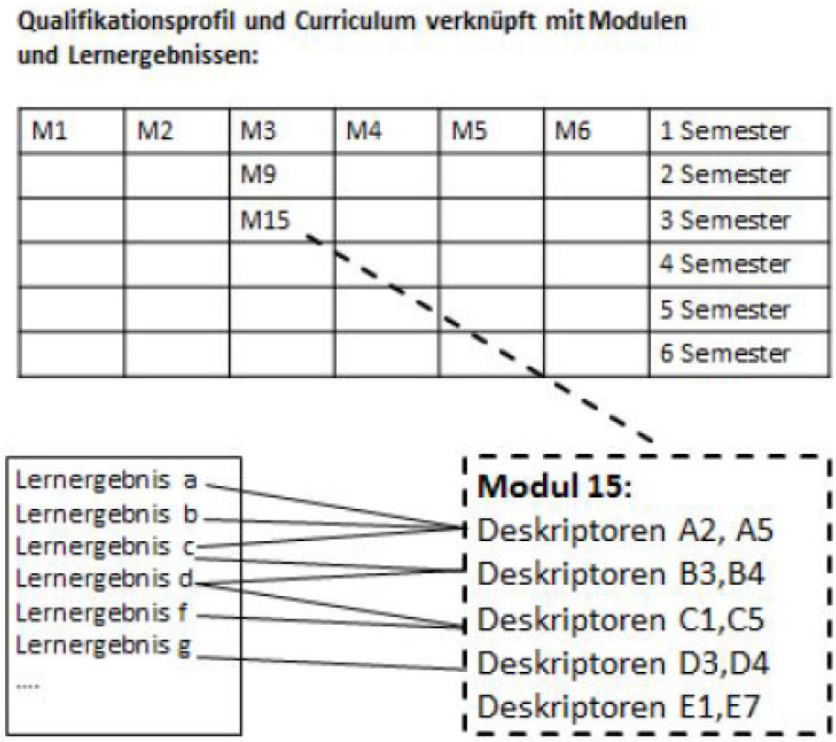


Abbildung 2.6: Lernergebnis, Modul, Deskriptor in Beziehung zueinander gesetzt

Die Deskriptoren werden also in den einzelnen Modulen differenziert. Diese Feinstrukturierung setzt sich praktisch in den einzelnen Lernschritten fort, wird aber nicht theoretisch formuliert. Damit wird auch die relative Bedeutung der einzelnen Module für die Bildung des Gesamt-QPs deutlich. Wir sind also in der Lage, die angestrebte Fähigkeit vor dem (individuellen) Beginn des Studiums hinreichend genau zu beschreiben. Diese Beschreibung bezieht sich nicht auf die Inhalte der Module, sondern auf die Kompetenz, die der Lernende nach erfolgreicher Absolvierung des Moduls nachweisen wird [3].

2.1.4 Zielsetzung

Das Ziel dieser Arbeit ist insofern begrenzt, dass nur ein Modell zur Klassifizierung von textbasierten Daten (Kompetenzen von Studiengängen) entwickelt wird. Genauer gesagt, dass dieses Modell zuerst alle Kompetenzen eines Moduls lernt und danach dieses Modul korrekt vorhersagen kann.
Die Daten, die in dieser Arbeit verwendet werden, stammen aus den Modulhandbüchern

von drei Studiengängen mit Schwerpunkt Informatik an der HAW. Sie werden daraus manuell gefiltert und in eine csv-Datei eingefügt. CSV steht für "comma-separated-values" und ist ein Dateiformat. Es ist ein einfaches Textformat, das in vielen Zusammenhängen verwendet wird, wenn große Datenmengen zusammengeführt werden sollen, ohne dass sie direkt miteinander verknüpft werden. Dies ist z. B. erforderlich, um Daten aus einer Datenbank in eine Tabellenkalkulation einzubinden.
Die Integration von Textdaten ist meist mit zusätzlichen Schwierigkeiten verbunden, da diese zunächst umformuliert und bereinigt werden müssen. So dient diese Arbeit auch dazu, den Datensatz auf seine Qualität und Eignung im Rahmen eines maschinellen Lernverfahrens zu untersuchen. Zu diesem Zweck wurden verschiedene Algorithmen verwendet, um ein Vorhersagemodell zu entwickeln. Die entwickelten Modelle werden am Ende miteinander verglichen, um das beste Modell für den Zweck dieser Arbeit herauszufinden. Das ausgewählte Modell kann in Zukunft für die Weiterentwicklung der KI verwendet werden.
Bei der automatischen Textklassifizierung werden maschinelles Lernen, die Verarbeitung natürlicher Sprache und andere KI-gestützte Verfahren eingesetzt, um Texte automatisch schneller, kostengünstiger und genauer zu klassifizieren als andere herkömmliche Methoden, die eine Reihe handgefertigter Daten verwenden.

2.2 Natural Language Processing

Natural Language Processing (NLP) entstand in den 1950er Jahren als Schnittpunkt von künstlicher Intelligenz und Linguistik. Ursprünglich unterschied sich NLP vom *Information Retrieval* (IR), das hochskalierbare, auf Statistiken basierende Techniken einsetzt, um große Textmengen effizient zu indizieren und zu durchsuchen. Im Laufe der Zeit haben sich NLP und IR jedoch etwas angenähert. Gegenwärtig nimmt NLP Anleihen aus mehreren, sehr unterschiedlichen Bereichen, was von den heutigen NLP-Forschern und -Entwicklern verlangt, ihre mentale Wissensbasis erheblich zu erweitern. [30]
Die NLP ist ein Teilgebiet der Informatik, der Informationstechnik und der künstlichen Intelligenz, das sich mit der Interaktion zwischen Computern und menschlichen (natürlichen) Sprachen befasst, insbesondere mit der Frage, wie Computer programmiert werden können, um große Mengen natürlichsprachlicher Daten zu verarbeiten und zu analysieren. Zu den Aufgaben der NLP gehören häufig die Spracherkennung, das Verstehen natürlicher Sprache und die Erzeugung natürlicher Sprache.
Einige interessante Anwendungsfälle von NLP sind:

- **Maschinelle Übersetzung**: Maschinelle Übersetzung ist die Aufgabe, eine natürliche Sprache automatisch in eine andere umzuwandeln, wobei die Bedeutung des Eingabetextes erhalten bleibt und ein flüssiger Text in der Ausgabesprache erzeugt wird.

- **Textklassifizierung**: Textklassifizierung ist der Prozess der Zuweisung von Tags oder Kategorien zu Text entsprechend seinem Inhalt. Dies ist ein grundlegendes Problem in der NLP und kann entweder manuell (mühsam, zeitaufwendig und anfällig für menschliche Fehler) oder mit Hilfe von ML-Techniken durchgeführt werden. Das ist es, was in dieser Arbeit betrachtet werden soll.

- **Stimmungsanalyse**: Die Sentiment-Analyse ist die kontextbezogene Auswertung von Text, die subjektive Informationen im Ausgangstext identifiziert und extrahiert, z. B. die Erkennung von Polarität (positiv, negativ, neutral), die Identifizierung von Emotionen usw. Ein typisches Beispiel ist die E-Commerce-Branche, in der die Auswertung und Analyse von Bewertungen wichtig ist, um Erkenntnisse über die Kundenzufriedenheit und -erfahrung zu gewinnen und potenzielle Verbesserungsbereiche zu ermitteln.

Das Verstehen natürlicher Sprache ist jedoch mit den folgenden Herausforderungen verbunden:

- **Mehrdeutigkeit**: Mehrdeutigkeit ist ein wesentliches Merkmal menschlicher Unterhaltungen und stellt eine besondere Herausforderung beim Verstehen natürlicher Sprache dar, da es unterschiedliche Formen geben kann, die in der natürlichen Sprache und in dem von uns programmierten KI-System relevant sind. In der KI-Theorie wird der Prozess des Umgangs mit Mehrdeutigkeit als Disambiguierung bezeichnet.

- **Synonymität**: Synonymität ergibt sich aus der Tatsache, dass man die gleiche Idee mit verschiedenen Begriffen ausdrücken kann (die auch vom jeweiligen Kontext abhängen); zum Beispiel haben “big” und “large” eine ähnliche Bedeutung, wenn sie sich auf Größen beziehen, während “large” keinen Sinn ergibt, wenn es als Qualifizierer für das Wort “sister” verwendet wird.

- **Co-Referenz**: Unter Co-Referenz versteht man das Auffinden aller Ausdrücke, die sich auf dieselbe Entität in einem Text beziehen. Die Auflösung von Co-Referenzen

ist ein wichtiger Schritt für viele übergeordnete NLP-Aufgaben, die das Verstehen natürlicher Sprache beinhalten, und trägt oft zur Verbesserung der Leistung neuronaler Architekturen wie RNN und LSTM bei.

- **Syntaktische Regeln**: Kenntnisse über die Struktur und Syntax der Sprache sind oft hilfreich. Beispiele: POS Tagging, Shallow Parsing/Chunking, ...

Die Abbildung 2.7 zeigt einen typischen NLP-Workflow mit maschinellem Lernen. Die detaillierten Schritte der Entwicklung werden im anschließenden Kapitel demonstriert.

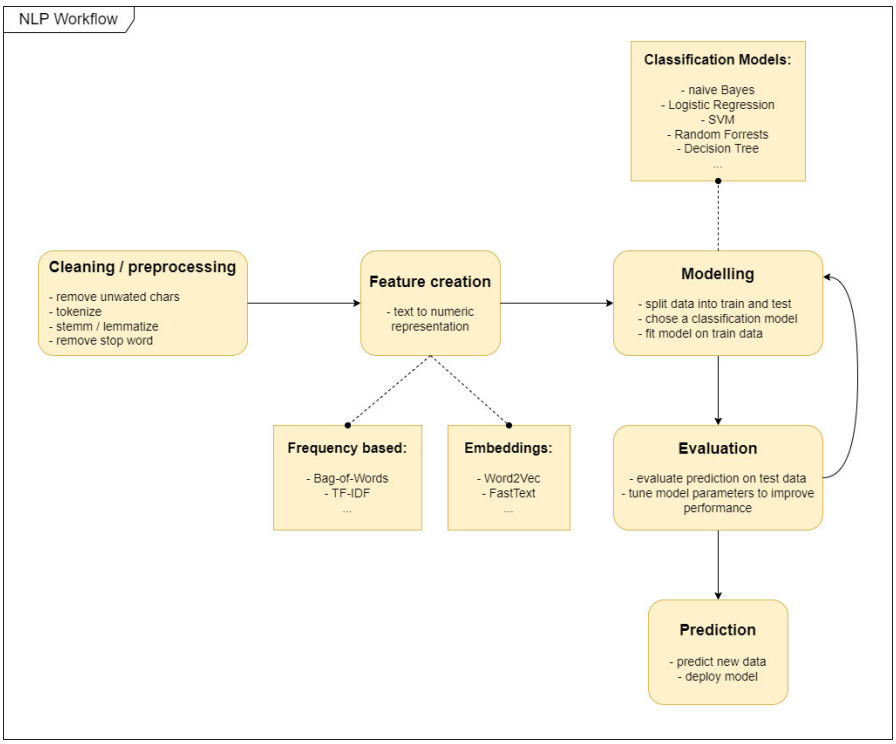


Abbildung 2.7: Standard NLP Arbeitsablauf

2.3 Machine Learning

In den letzten Jahren sind das maschinelle Lernen und die künstliche Intelligenz zu sehr populären Themen geworden. Heutzutage sind ihre Methoden und Ansätze Teil einer großen Anzahl von Produkten, außerdem sind sie in den meisten Anwendungen und Geräten unverzichtbar. Ein Beispiel für den Einsatz von ML (Machine Learning) ist die automatische Bestimmung wichtiger E-Mails und schneller Antworten in Gmail [16].

Heutzutage kann man getrost sagen, dass künstliche Intelligenz mit maschinellem Lernen den Menschen aus vielen technologischen Prozessen verdrängen kann [27].
Maschinelles Lernen ist die wissenschaftliche Untersuchung von Algorithmen und statistischen Methoden, die von Computersystemen verwendet werden, um eine bestimmte Aufgabe effektiv zu erfüllen, ohne explizite Anweisungen zu verwenden und sich stattdessen auf Muster und Schlussfolgerungen zu verlassen. Es wird als ein Teilbereich der künstlichen Intelligenz betrachtet. Algorithmen des maschinellen Lernens erstellen ein mathematisches Modell von Beispieldaten, die als "Trainingsdaten" bezeichnet werden, um Vorhersagen oder Entscheidungen zu treffen, ohne dass sie explizit für die Durchführung der Aufgabe programmiert werden. Es gibt fünf Arten von Algorithmen für maschinelles Lernen: überwachtes, halbüberwachtes, aktives Lernen, Verstärkung und unüberwachtes Lernen [20].
Die Aufgabe, einen Klassifikator für Dokumente zu konstruieren unterscheidet sich nicht wesentlich von anderen Aufgaben des Maschinellen Lernens. Anstatt sich auf manuell erstellte Regeln zu verlassen, lernt die maschinelle Textklassifizierung, Klassifizierungen auf der Grundlage früherer Beobachtungen vorzunehmen. Durch die Verwendung von vorbeschrifteten Beispielen als Trainingsdaten können Algorithmen des maschinellen Lernens die verschiedenen Assoziationen zwischen Textstücken erlernen und erkennen, dass eine bestimmte Ausgabe (Tags) für eine bestimmte Eingabe (Text) erwartet wird. Ein "Tag" ist die vorher festgelegte Klassifizierung oder Kategorie, in die ein bestimmter Text fallen könnte.
Der erste Schritt zum Training eines NLP-Klassifikators mit maschinellem Lernen ist die Merkmalsextraktion: Jeder Text wird mit einer Methode in eine numerische Darstellung in Form eines Vektors umgewandelt. Einer der am häufigsten verwendeten Ansätze ist *Bag of Words*, bei dem ein Vektor die Häufigkeit eines Wortes in einem vordefinierten Wörterbuch darstellt.
Dann wird der Algorithmus für maschinelles Lernen mit Trainingsdaten gefüttert, die aus Paaren von Merkmalen (Vektoren für jedes Textbeispiel) und Tags (z. B. WI, AI, ...) bestehen, um ein Klassifikationsmodell zu erstellen (siehe Abbildung 2.8).

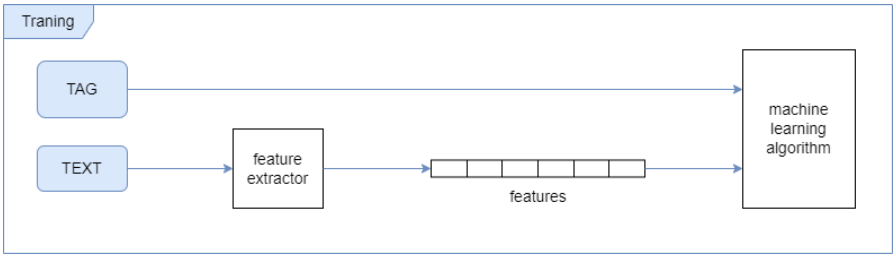


Abbildung 2.8: Trainingsphase des maschinellen Lernens

Sobald es mit genügend Trainingsbeispielen trainiert wurde, kann das maschinelle Lernmodell genaue Vorhersagen machen. Derselbe Merkmalsextraktor wird verwendet, um ungesehenen Text in Merkmalssätze umzuwandeln, die in das Klassifizierungsmodell eingespeist werden können, um Vorhersagen zu Tags zu erhalten (siehe Abbildung 2.9).

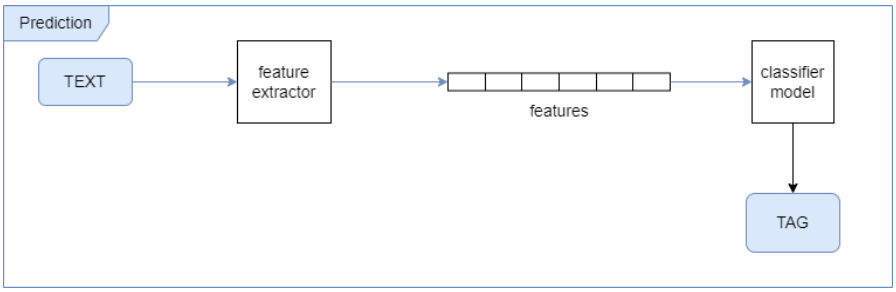


Abbildung 2.9: Vorhersagephase des maschinellen Lernens

Die Textklassifizierung mit maschinellem Lernen ist in der Regel viel genauer als von Menschen erstellte Regelsysteme, insbesondere bei komplexen NLP-Klassifizierungsaufgaben. Außerdem sind Klassifikatoren mit maschinellem Lernen einfacher zu pflegen und man kann immer neue Beispiele markieren, um neue Aufgaben zu lernen.
Zu den in dieser Arbeit implementierten Algorithmen zur Textklassifizierung gehören:

- Decision Tree
- Gradient Boosting
- K Nearest Neighbor
- Logistic Regression
- Multinomial Naive Bayes

- Random Forest
- Support Vector Machine

2.4 Konfusionsmatrix

Im Bereich des maschinellen Lernens und insbesondere des Problems der statistischen Klassifizierung ist eine **Konfusionsmatrix**, auch Fehlermatrix genannt, ein spezielles Tabellenlayout, das die Visualisierung der Leistung eines Algorithmus, in der Regel eines überwachten Lernverfahrens, ermöglicht (beim unüberwachten Lernen wird sie gewöhnlich als Matching-Matrix bezeichnet). Jede Zeile der Matrix stellt die Instanzen in einer tatsächlichen Klasse dar, während jede Spalte die Instanzen in einer vorhergesagten Klasse darstellt, oder umgekehrt.
Im Folgenden wird das Vorgehen zur Berechnung einer Konfusionsmatrix beschrieben:

1. Ein Testdatensatz oder ein Validierungsdatensatz mit erwarteten Ergebniswerten sollte vorhanden sein.
2. Mach eine Vorhersage für jede Zeile im Testdatensatz.
3. Von den erwarteten Ergebnissen und Vorhersagen zählen:
    - Anzahl der richtigen Vorhersagen für jede Klasse.
    - Anzahl der falschen Vorhersagen für jede Klasse, geordnet nach der vorhergesagten Klasse.

Diese Zahlen werden dann wie folgt in einer Tabelle oder einer Matrix angeordnet:

- Jede Zeile der Matrix entspricht einer vorhergesagten Klasse.
- Jede Spalte der Matrix entspricht einer tatsächlichen Klasse.

Die Anzahl der richtigen und falschen Klassifizierungen wird dann in die Tabelle eingetragen. Die Gesamtzahl der richtigen Vorhersagen für eine Klasse wird in die erwartete Zeile für diesen Klassenwert und die vorhergesagte Spalte für diesen Klassenwert übertragen. Ebenso wird die Gesamtzahl der falschen Vorhersagen für eine Klasse in die erwartete Zeile für diesen Klassenwert und die vorhergesagte Spalte für diesen Klassenwert eingefügt.

In der Praxis kann ein binärer Klassifikator wie dieser zwei Arten von Fehlern machen: Er kann eine Person, die standardmäßig in die Kategorie “kein Standard” fällt, falsch zuordnen, oder er kann eine Person, die nicht standardmäßig in die Kategorie “Standard” fällt, falsch zuordnen. Es ist oft von Interesse, welche dieser beiden Arten von Fehlern gemacht werden. Eine Konfusionsmatrix ist eine praktische Möglichkeit, diese Informationen anzuzeigen [18].
Diese Matrix kann für 2-Klassen-Probleme verwendet werden, wo sie sehr leicht zu verstehen ist, kann aber auch leicht auf Probleme mit 3 oder mehr Klassenwerten angewendet werden, indem weitere Zeilen und Spalten zur Konfusionsmatrix hinzugefügt werden.

3 Praktische Durchführung

Das folgende Kapitel befasst sich mit der Entwicklung eines Vorhersagemodells, das Studienbewerbern an der HAW bei ihrer Studiengangsentscheidung helfen soll. Zunächst erfolgen eine detaillierte Beschreibung des vorliegenden Datenbestandes und eine erste Beurteilung bzgl. der Umsetzbarkeit der NLP-Technik. Anschließend wird in Abschnitt 3.2 die Vorauswahl der zu berücksichtigenden Datenpunkte beschrieben, die in Abschnitt 3.3 zur Identifikation auffälliger Merkmale ausgewertet werden. Daran schließt sich in Abschnitt 3.4 die Identifikation von abstrakten Klassen an. Damit das System erkennen kann, welche Kompetenz zu welchem Studiengang gehört, werden in Abschnitt 3.5 das Training und der entsprechende Test durchgeführt. Abschließend erfolgt in Abschnitt 3.6 eine Beschreibung der verwendeten Modelle und der Maßnahmen zur Optimierung dieser Modelle.

3.1 Eingabedaten

Das Modulhandbuch bietet den Studierenden des jeweiligen Studiengangs eine Orientierung über die gesamte Studienzeit. Darüber hinaus regelt es die Inhalte des Studiengangs und stellt sie in eine zeitliche Abfolge und beschreibt die zu erreichenden Kompetenzen. In Anlehnung an den Deutschen Qualifikationsrahmen für Lebenslanges Lernen [8] werden die Kompetenzen der einzelnen Fächer ausgewiesen, die in 2 Unterkategorien unterteilt sind:

1. Wissenverbreitung und Wissenvertiefung: Die Studierenden verfügen über ein breites und integriertes Wissen und Verständnis der wissenschaftlichen Grundlagen ihres Fachgebiets. Sie verfügen über ein kritisches Verständnis der wichtigsten Methoden, Grundsätze und Theorien ihres Fachs und sind in der Lage, ihr Wissen vertikal, horizontal und lateral zu vertiefen und zu erweitern.

2. Fertigkeiten: Diese Fähigkeiten beschreiben eine instrumentelle und systemische Kompetenz sowie die Fähigkeit, den Bereich der Wissensentwicklung zu beurteilen und zuzuordnen. Die Studierenden sind in der Lage, ihre Wissensgebiete selbstständig zu entwickeln und zu erweitern.

In dieser Studie werden nur die drei Studiengänge des Fakultät Technik und Informatik an der Hochschule für Angewandte Wissenschaften Hamburg betrachtet. Der in diesem Projekt verwendete Datensatz beschreibt die für Informatikstudiengänge definierten Kompetenzen, die aus 3 aktuellen Modulhandbüchern der HAW gesammelt wurden. Für jedes Modul wird eine Liste von Kompetenzen angegeben, die ein Student nach erfolgreichem Abschluss der Prüfung beherrschen sollte.
Der Datensatz besteht aus 165 Kompetenzen, die ein Student nach den entsprechenden Modulen beherrschen sollte und ist in 4 Bereiche unterteilt. Diese Bereiche sind:

1. **fachübergreifend** bezeichnet die allgemeine Kompetenzen von Informatiker.

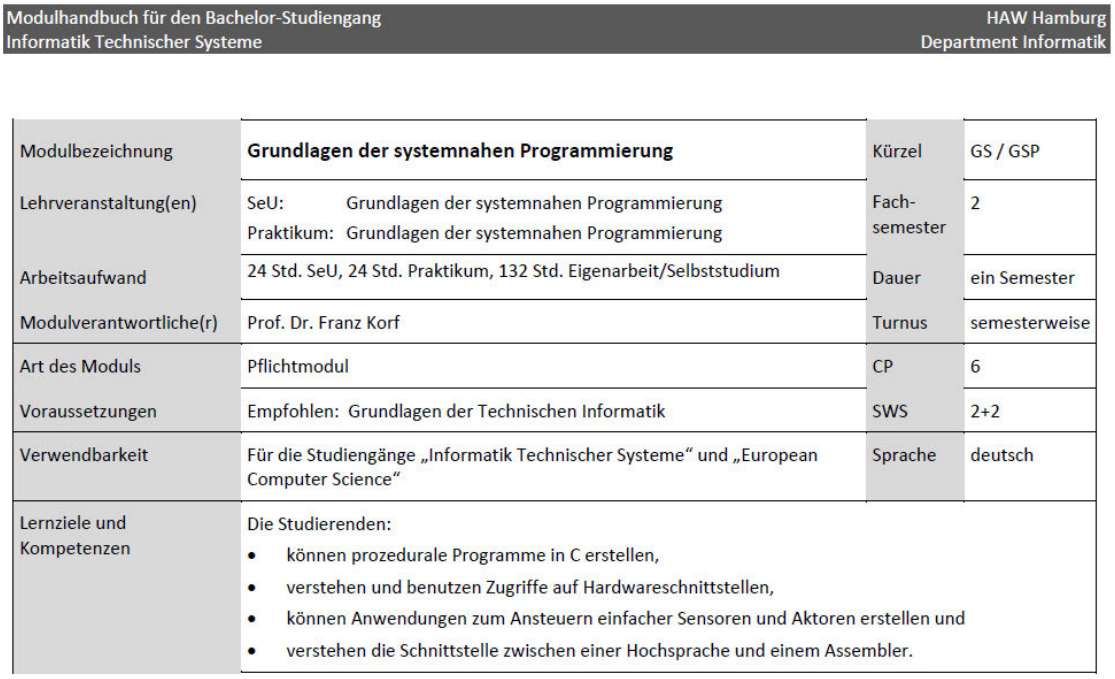
2. **Angewandte Informatik** ist die Wissenschaft, die sich mit der Entwicklung und Verbesserung computergestützter Lösungen für Probleme im Alltag und in der Wissenschaft beschäftigt. Sie beschäftigt sich einerseits mit der Softwareentwicklung und der Entwicklung von Systemen, die Daten optimal verarbeiten (z.B. Datenbanken), andererseits aber auch mit der Entwicklung automatisierter Prozesse. Die Angewandte Informatik untersucht auch Computerarchitekturen und Computerstrukturen und versucht, Wege zu finden, beides so optimal wie möglich zu gestalten.

3. **Wirtschaftsinformatik** beschäftigt sich mit der Entwicklung, Steuerung und Pflege von Informations- und Kommunikationssystemen in Wirtschaftsunternehmen. Sie verbindet wirtschaftlichen Sachverstand mit angewandter Informatik und ist daher eine interdisziplinäre Wissenschaft.

4. **Informatik Technischer Systeme** nutzt und steuert technische Systeme in einer intelligenten Umgebung, um autonome Geräte zu steuern, beispielsweise im Rahmen der Hausautomatisierung, Smart City oder postindustrieller Fertigungsanlagen. Im Kern steht die Entwicklung von Software, insbesondere für den Einsatz bei Cyber-physischen Systemen und bei der Nutzung und Bereitstellung von Schnittstellen zu Hardware- und Infrastrukturkomponenten.

3.2 Erstellung des Ausgangsdatensatzes

Im Anschluss an eine erste Analyse der Daten erfolgte die Auswahl geeigneter Datenpunkte aus dem Gesamtdatenbestand. Da es sich bei den Fächern um Kompetenzerfassung der Studierenden handelt, wurde nur das angegebene Sachverständnis der einzelnen Module berücksichtigt. Ein Großteil der im Modulhandbuch definierten Fachkompetenzen ist einzigartig.



Modulhandbuch für den Bachelor-Studiengang Informatik Technischer Systeme — HAW Hamburg Department Informatik

Modulbezeichnung	**Grundlagen der systemnahen Programmierung**	Kürzel	GS / GSP
Lehrveranstaltung(en)	SeU: Grundlagen der systemnahen Programmierung Praktikum: Grundlagen der systemnahen Programmierung	Fach-semester	2
Arbeitsaufwand	24 Std. SeU, 24 Std. Praktikum, 132 Std. Eigenarbeit/Selbststudium	Dauer	ein Semester
Modulverantwortliche(r)	Prof. Dr. Franz Korf	Turnus	semesterweise
Art des Moduls	Pflichtmodul	CP	6
Voraussetzungen	Empfohlen: Grundlagen der Technischen Informatik	SWS	2+2
Verwendbarkeit	Für die Studiengänge „Informatik Technischer Systeme“ und „European Computer Science“	Sprache	deutsch
Lernziele und Kompetenzen	Die Studierenden: • können prozedurale Programme in C erstellen, • verstehen und benutzen Zugriffe auf Hardwareschnittstellen, • können Anwendungen zum Ansteuern einfacher Sensoren und Aktoren erstellen und • verstehen die Schnittstelle zwischen einer Hochsprache und einem Assembler.		

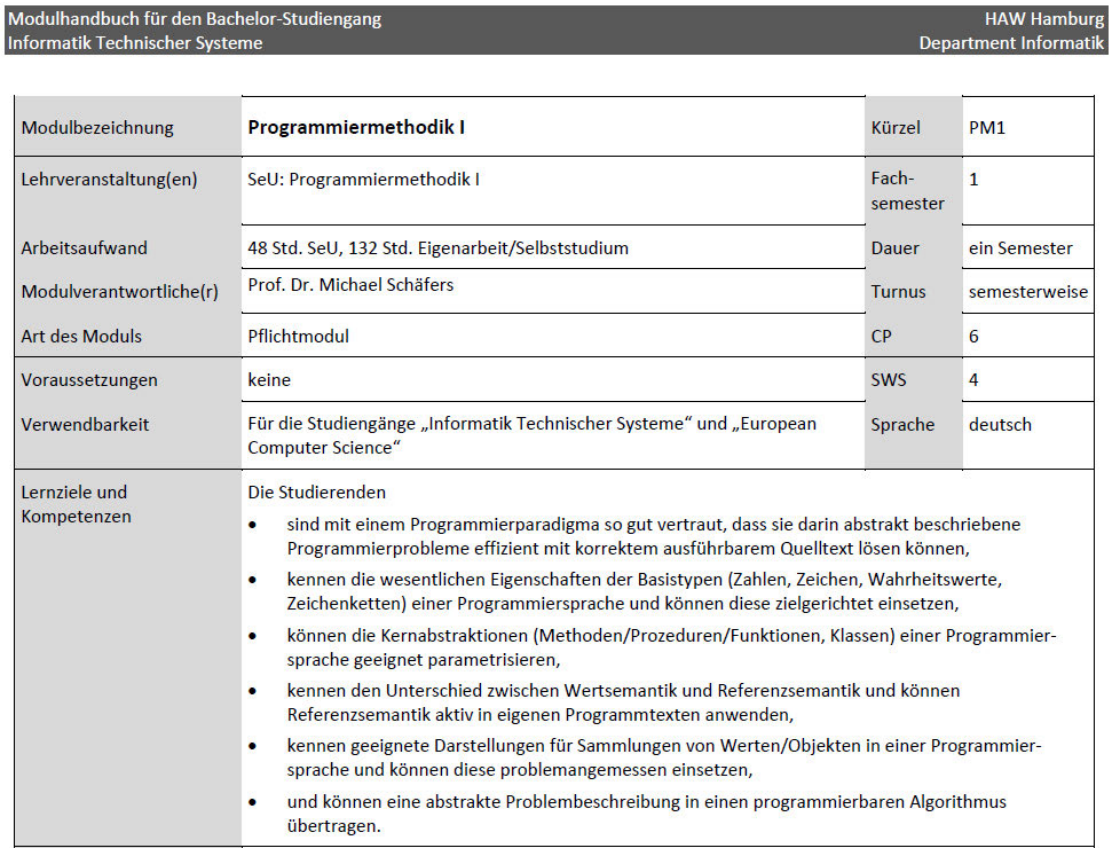
Abbildung 3.1: Modul Grundlagen der systemnahen Programmierung vom Modulhandbuch des Informatik Technischer Systeme. Stand: 21.01.2021

Modulhandbuch für den Bachelor-Studiengang Wirtschaftsinformatik — HAW Hamburg Department Informatik

Modulbezeichnung	**Betriebswirtschaftslehre II (Betriebliches Rechnungswesen)**	Kürzel	BWL2/BWLP2
Lehrveranstaltung(en)	Vorlesung: Betriebswirtschaftslehre II (Betriebliches Rechnungswesen) Praktikum: Betriebswirtschaftslehre II (Betriebliches Rechnungswesen)	Semester	2
Arbeitsaufwand	36 Std. Vorlesung, 12 Std. Praktikum, 132 Std. Eigenarbeit/Selbststudium	CP	6
Modulverantwortliche(r)	Prof. Dr. Jens-Eric von Düsterlho	SWS	3+1
Dozenten	Prof. Dr. Jens-Eric von Düsterlho	Sprache	deutsch
Voraussetzungen	keine	Häufigkeit	jährlich
Lernziele und Kompetenzen	Die Studierenden • besitzen fundierte Kenntnisse über das betriebliche Rechnungswesen (internes und externes Rechnungswesen) • können Kalkulations-, Buchhaltungs- und Bilanzierungsfragen lösen • können Jahresabschlüsse lesen und verstehen		

Abbildung 3.2: Modul Betriebswirtschaftslehre II vom Modulhandbuch des Wirtschaftsinformatik. Stand: 2016

Solche Datenpunkte bezeichnen in der Regel explizit die Merkmale des jeweiligen Studiengangs. Das in der Abbildung 3.1 dargestellte Modul steht für die fachspezifischen Kompetenzen des zugehörigen Studiengangs. In der **Informatik Technischer Systeme** ist es entscheidend, dass die Studierenden selbstständig mit Sensoren und Aktoren arbeiten können, um nach dem Studium eingebettete Software als Teilsystem eines intelligenten Geräts entwickeln zu können. Ein entsprechendes Modul in **Wirtschaftsinformatik** ist in der Abbildung 3.2 dargestellt. Es ist also zu erkennen, dass in der **Wirtschaftsinformatik** das technische Wissen aus dem Teil „Wirtschaft“ eher im Vordergrund steht.



Modulhandbuch für den Bachelor-Studiengang Informatik Technischer Systeme — HAW Hamburg Department Informatik

Modulbezeichnung	**Programmiermethodik I**	Kürzel	PM1
Lehrveranstaltung(en)	SeU: Programmiermethodik I	Fach-semester	1
Arbeitsaufwand	48 Std. SeU, 132 Std. Eigenarbeit/Selbststudium	Dauer	ein Semester
Modulverantwortliche(r)	Prof. Dr. Michael Schäfers	Turnus	semesterweise
Art des Moduls	Pflichtmodul	CP	6
Voraussetzungen	keine	SWS	4
Verwendbarkeit	Für die Studiengänge „Informatik Technischer Systeme“ und „European Computer Science“	Sprache	deutsch
Lernziele und Kompetenzen	Die Studierenden • sind mit einem Programmierparadigma so gut vertraut, dass sie darin abstrakt beschriebene Programmierprobleme effizient mit korrektem ausführbarem Quelltext lösen können, • kennen die wesentlichen Eigenschaften der Basistypen (Zahlen, Zeichen, Wahrheitswerte, Zeichenketten) einer Programmiersprache und können diese zielgerichtet einsetzen, • können die Kernabstraktionen (Methoden/Prozeduren/Funktionen, Klassen) einer Programmiersprache geeignet parametrisieren, • kennen den Unterschied zwischen Wertsemantik und Referenzsemantik und können Referenzsemantik aktiv in eigenen Programmtexten anwenden, • kennen geeignete Darstellungen für Sammlungen von Werten/Objekten in einer Programmiersprache und können diese problemangemessen einsetzen, • und können eine abstrakte Problembeschreibung in einen programmierbaren Algorithmus übertragen.		

Abbildung 3.3: Modul Programmiermethodik vom Modulhandbuch des Informatik Technischer Systeme. Stand: 21.01.2021

Darüber hinaus sind einige Datenpunkte aus verschiedenen Studiengängen identisch, die meist in den früheren Semestern beigebracht werden und elementare Fähigkeiten des allgemeinen Informatikers identifizieren. Die Abbildung 3.3 zeigt die Kompetenzen, die ein Softwareentwickler als Basiswissen beherrschen muss. Diese Datenpunkte werden als **fachübergreifende** Kompetenz oder allgemeine Kompetenz von Informatik zugeordnet. Schließlich wird ein Datensatz wie in der Tabelle 3.1 strukturiert:

Kompetenz	Studiengang
Kompetenz 1	Studiengang 1
Kompetenz 2	Studiengang 2
Kompetenz 3	Studiengang 3
Kompetenz ...	Studiengang ...

Tabelle 3.1: Struktur des Datensatzes

3.3 Explorative Datenanalyse

Das erste, was man bei jedem Datensatz tun sollte, ist, sich mit ihm vertraut zu machen. Dies geschieht nicht nur, um sich mit allen gesammelten Daten vertraut zu machen, sondern auch, um die Arbeitsbelastung bei der Analyse zu verringern. [6] Bei einer explorativen Analyse [28] werden die Daten aus möglichst vielen Blickwinkeln betrachtet, immer auf der Suche nach einem interessanten Merkmal. Der Datenanalyst ist daran interessiert, Fakten über die Daten aufzudecken und kann zu diesem Zweck jedes beliebige Verfahren anwenden. Die einzigen Grenzen, die einer solchen Analyse gesetzt sind, sind die zeitlichen Beschränkungen und die Kreativität des Datenanalysten. In diesem Fall können aufgrund der Art der textbasierten Merkmale nicht viele Erkenntnisse gewonnen werden. Aus diesem Grund wird in diesem Schritt nur eine oberflächliche Analyse durchgeführt.

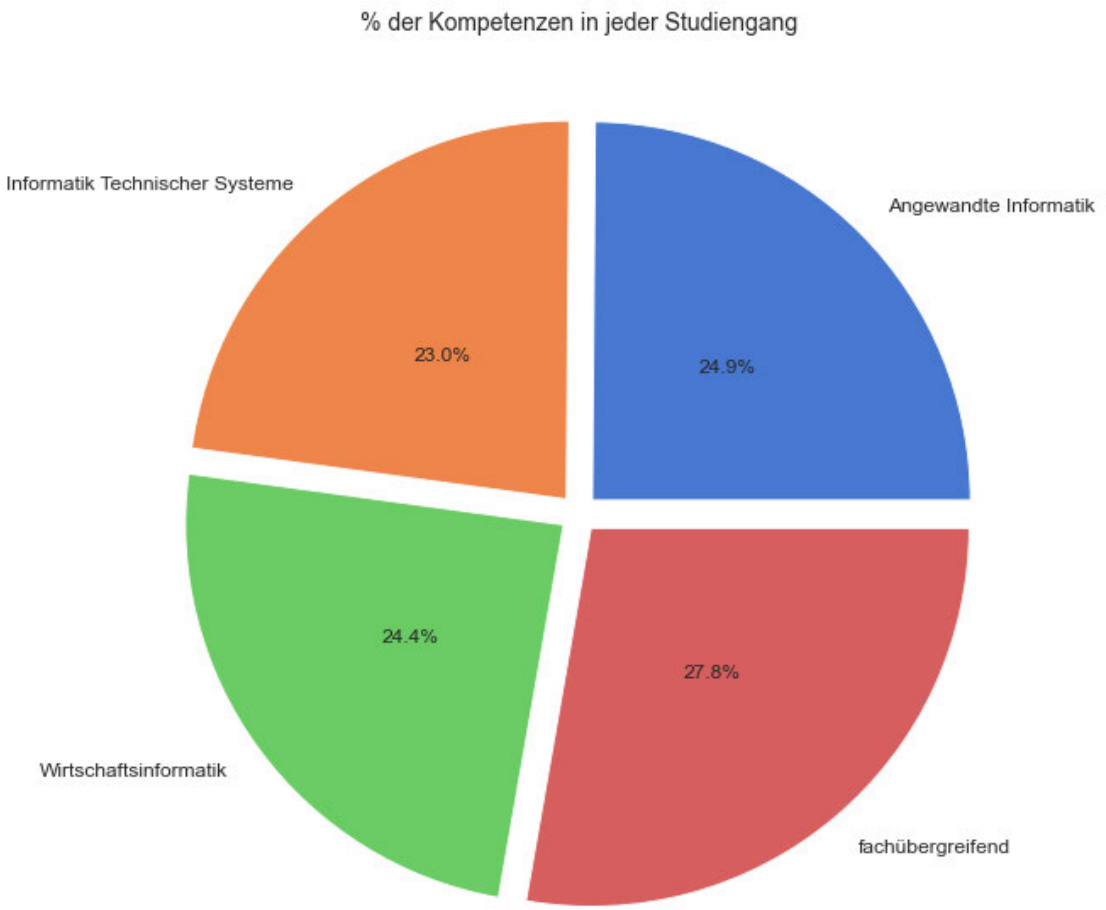


Abbildung 3.4: Prozent der Kompetenzen in jedem Studiengang

In der Abbildung 3.4 wird der prozentualen Anteil der Gesamtkompetenzen in jedem Studiengang dargestellt. Es fällt auf, dass **Informatik Technische Systeme** den kleinsten Anteil hat, etwa ein Fünftel der Gesamtzahl. Die verbleibenden Studiengänge sind recht ausgewogen und weisen nur 1 bis 2 Prozent mehr Kompetenzen auf als die ersten. Dies ist jedoch eines der Hauptanliegen bei der Entwicklung eines Klassifizierungsmodells, dass die verschiedenen Klassen ausgewogen sind. Es gibt mehrere Möglichkeiten, mit balancierten Datensätzen umzugehen.
Anschließend ist es wichtig, einen Blick auf die Länge des Textes zu werfen, denn es ist eine einfache Berechnung, die viele Erkenntnisse liefern kann. Wenn eine Kategorie systematisch länger ist als eine andere, dann wäre die Länge das einzige Merkmal, das

für die Erstellung des Modells benötigt wird. In Python gibt es mehrere Längenmaße für Textdaten:

- **word count**: zählt die Anzahl der Token im Text (getrennt durch ein Leerzeichen)
- **character count**: die Anzahl der Zeichen eines jeden Tokens summieren
- **sentence count**: die Anzahl der Sätze zählen (getrennt durch einen Punkt)
- **average word length**: Summe der Wortlänge geteilt durch die Anzahl der Wörter (Zeichenanzahl/Wortanzahl)
- **average sentence length**: Summe der Satzlänge geteilt durch die Anzahl der Sätze (Wortanzahl/Satzanzahl)

Da eine Kompetenz eher einen String als einen vollständigen Satz dargestellt wird, entspricht die Methode **character count** im Prinzip dem Ziel dieser Arbeit. Die Abbildung 3.5 gibt zunächst Informationen über die Verteilung der Länge der Kompetenzerfassungen. Interessant ist, die Mehrheit der Kompetenz von 80 bis 120 Charaktere abgebildet wird. Außerdem werden nur 4 Kompetenzen mit mehr als 300 Zeichen beschrieben.

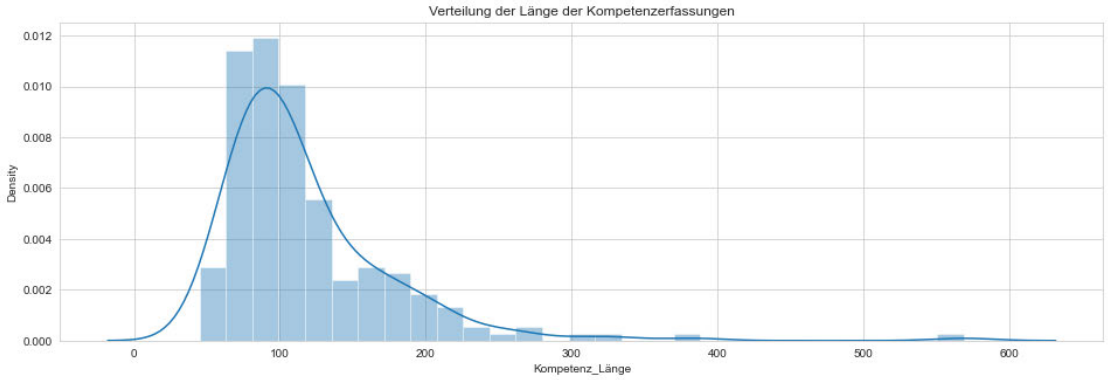


Abbildung 3.5: Verteilung der Länge der Kompetenzerfassungen im Gesamtkontext

Bei der explorativen Datenanalyse werden statistische Verfahren eingesetzt, um Muster zu erkennen, die in einer Gruppe von Zahlen verborgen sein könnten. Eine dieser Techniken ist das *Boxplot* [44], das dazu dient, Datengruppen visuell zusammenzufassen und zu vergleichen. Das *Boxplot* verwendet den Median, die ungefähren Quartile sowie die niedrigsten und höchsten Datenpunkte, um das Niveau, die Streuung und die Symmetrie

einer Verteilung von Datenwerten zu vermitteln. Die Abbildung 3.6 zeigt die wesentlichen Teile eines *Boxplot*, die folgendermaßen definiert sind:

- Median: Der Median (mittleres Quartil) markiert den Mittelpunkt der Daten und wird durch die Linie dargestellt, die das Feld in zwei Teile teilt. Die Hälfte der Werte ist größer oder gleich diesem Wert, die andere Hälfte ist kleiner.
- Inter-Quartil-Bereich: Die mittlere "Box" stellt die mittleren 50% der Werte für die Gruppe dar. Der Bereich zwischen dem unteren und dem oberen Quartil wird als Interquartilsbereich bezeichnet. Die mittleren 50% der Werte liegen innerhalb des Interquartilsbereichs.
- Oberes Quartil: Fünfundsiebzig Prozent der Ergebnisse liegen unterhalb des oberen Quartils.
- Unteres Quartil: Fünfundzwanzig Prozent der Ergebnisse liegen unterhalb des unteren Quartils.
- Whisker: Der obere und der untere Whisker stellen die Werte dar, die außerhalb der mittleren 50% liegen. Die Whiskers erstrecken sich oft (aber nicht immer) über einen größeren Wertebereich als die mittleren Quartilsgruppen.

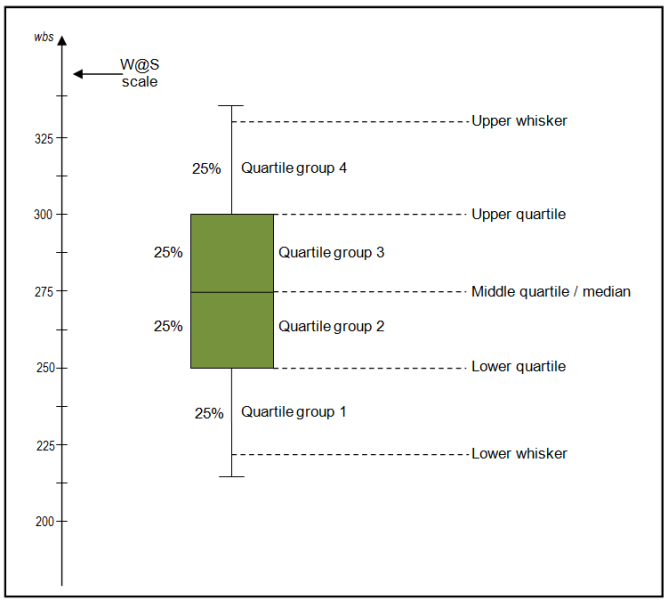


Abbildung 3.6: Boxplot-Labels. URL

Die Abbildung 3.7 stellt in Form eines *Boxplot* dar, wie die Länge der Kompetenz in jedem Studiengang verteilt ist. Es lässt sich eine deutliche Tendenz dadurch erkennen, dass Kompetenzen in **Informatik Technische Systeme** länger sind, allerdings nicht in signifikantem Ausmaß. Darüber hinaus wird im nächsten Abschnitt die Länge der Texte berücksichtigt und durch die Methode, die zur Erstellung der Merkmale verwendet wird, korrigiert.

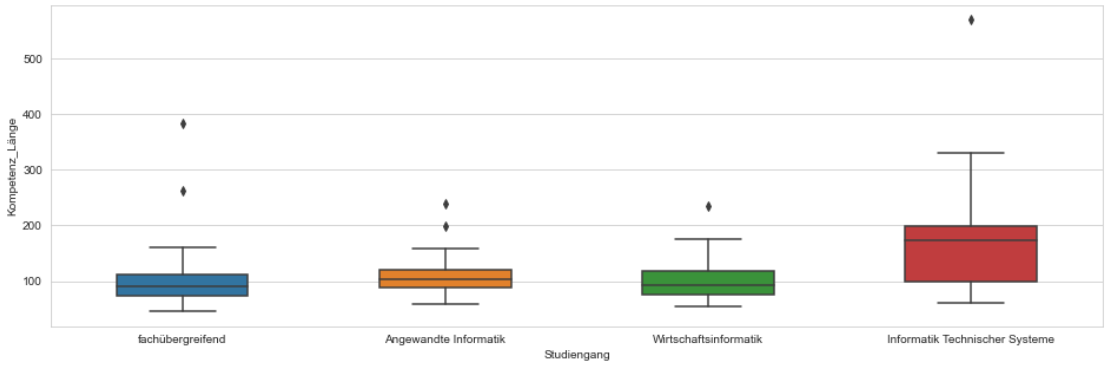


Abbildung 3.7: Verteilung der Länge der Kompetenzerfassungen in jedem Studiengang durch Boxplot

3.4 Merkmalstechnik

Im Anschluss an eine erste Analyse der Daten erfolgte die Auswahl geeigneter Merkmale aus dem Gesamtdatenbestand. Die Merkmalstechnik (engl. feature engineering) [19] ist ein entscheidender Schritt im Prozess der prädiktiven Modellierung. Es umfasst die Transformation eines gegebenen Merkmalsraums, in der Regel unter Verwendung mathematischer Funktionen, mit dem Ziel, den Modellierungsfehler für ein bestimmtes Ziel zu reduzieren. Beim Umgang mit Textdaten gibt es mehrere Möglichkeiten, um Merkmale zu erhalten, die die Daten repräsentieren. Einige der gebräuchlichsten Methoden werden im nächsten Unterabschnitt vorgestellt, dann wird die für diese Fallstudie am besten geeignete Methode ausgewählt.

3.4.1 Darstellung des Textes

Yoav Goldberg hat festgestellt, dass bei der Sprachverarbeitung Vektoren x aus Textdaten abgeleitet werden, um verschiedene linguistische Eigenschaften des Textes wiederzu-

geben [13]. Dies führt jedoch zu einem Problem bei der Modellierung von Text, dass er unübersichtlich ist, und Techniken wie Algorithmen für maschinelles Lernen bevorzugen klar definierte Eingaben und Ausgaben mit fester Länge. Algorithmen für maschinelles Lernen können nicht direkt mit Rohtext arbeiten. Der Text muss in Zahlen oder in Zahlenvektoren umgewandelt werden.

3.4.1.1 Bag-of-Words-Modell

Eine beliebte und einfache Methode der Merkmalsextraktion aus Textdaten ist das Bag-of-Words-Modell [48] für Text, welches in der *Natural Language Processing* und im Information Retrieval weit verbreitet ist. In diesem Modell wird jedes Dokument als Tasche betrachtet, die seine Wörter enthält, wobei die Wortvielfalt beibehalten und die Grammatik und die Wortreihenfolge ignoriert werden. Seltene und häufige Wörter werden oft aus dem Originaltext herausgefiltert, und manchmal werden Wörter, die in einer Liste von Stoppwörtern [38] auftauchen, entfernt (Stoppwörter sind funktionale oder verbindende Wörter, von denen angenommen wird, dass sie keinen Informationsgehalt haben). Neben der Entfernung von häufigen, seltenen und Stopp-Wörtern wird oft versucht, die Merkmale statistisch unabhängiger zu machen. Am häufigsten wird dies durch die Entfernung von Suffixen aus Wörtern mithilfe eines Stemming-Algorithmus erreicht. Das Stemming bewirkt, dass mehrere morphologische Formen von Wörtern auf ein gemeinsames Merkmal abgebildet werden. Beispielsweise würden die Wörter “learner”, “learning” und “learned” würden alle dem gemeinsamen Stamm gemeinsamen Wortstamm “learn” abgebildet, und diese letztere Zeichenfolge würde in die Merkmalsmenge aufgenommen werden, anstatt die drei ersten.
Es gibt verschiedene Möglichkeiten, Text in Vektoren umzuwandeln:

- Zählen, wie oft jedes Wort in einem Dokument vorkommt.
- Berechnen der Häufigkeit, mit der jedes Wort in einem Dokument angezeigt wird, aus allen Wörtern im Dokument

Diese beiden folgenden Methoden werden oft als Bag of Words-Methoden bezeichnet, da die Reihenfolge der Wörter in einem Satz ignoriert wird.

1. Word Count Vectors: Die Methode arbeitet mit der Häufigkeit von Begriffen, d. h. sie zählt das Auftreten von Token und erstellt eine dünnbesetzte Matrix [2] aus Dokumenten und Token.

2. *TF–IDF* Vectors: *TF-IDF* steht für Term Frequency-Inverse Document Frequency und ist ein Maß, das in den Bereichen Information Retrieval (IR) und maschinelles Lernen verwendet wird und mit dem die Bedeutung oder Relevanz von Zeichenfolgen (Wörtern, Phrasen, Lemmata usw.) in einem Dokument oder in einer Sammlung von Dokumenten (auch als Korpus bezeichnet) quantifiziert werden kann. *TF-IDF* wird wie folgt berechnet:

$$TFIDF(t,d) = TF(t,d) \times log(\frac{N}{DF(t)})$$

In dem:

- t: Begriff (d.h. ein Wort in einem Dokument)
- d: Dokument
- TF(t): Termfrequenz (d. h. wie oft der Begriff t in dem Dokument d vorkommt)
- N: Anzahl der Dokumente im Korpus
- DF(t): Anzahl der Dokumente im Korpus, die Begriff Term t enthalten

Die TF-IDF-Gewichtung ist ein statistisches Maß, mit dem bewertet wird, wie wichtig ein Wort für ein Dokument in einer Sammlung oder einem Korpus ist. Die Bedeutung steigt proportional zur Häufigkeit, mit der ein Wort im Dokument erscheint, wird aber durch die Häufigkeit des Wortes im Korpus ausgeglichen.

3.4.1.2 Worteinbettungen

Worteinbettungen (engl. word embeddings) [12] sind im Allgemeinen Projektionen auf einen kontinuierlichen Raum von Wörtern, also die semantischen und syntaktischen Ähnlichkeiten zwischen ihnen. Im Einzelnen handelt es sich um eine Klasse von Techniken, bei denen einzelne Wörter als echtwertige Vektoren in einem vordefinierten Vektorraum dargestellt werden. Jedes Wort wird einem Vektor zugeordnet, und die Vektorwerte werden auf eine Art und Weise erlernt, die einem neuronalen Netz ähnelt, weshalb diese Technik oft in den Bereich des Deep Learning eingeordnet wird.
Der Schlüssel zu diesem Ansatz ist die Idee der Verwendung einer dicht verteilten Darstellung für jedes Wort. Jedes Wort wird durch einen realwertigen Vektor mit oft zehn oder hundert Dimensionen dargestellt. Dies steht im Gegensatz zu den Tausenden oder

Millionen von Dimensionen, die für spärliche Wortdarstellungen erforderlich sind, wie z. B. bei einer One-Hot Encoding [36]. Die verteilte Darstellung wird auf der Grundlage der Verwendung von Wörtern gelernt. Dies ermöglicht es, dass Wörter, die auf ähnliche Weise verwendet werden, ähnliche Repräsentationen ergeben, die ihre Bedeutung auf natürliche Weise erfassen. Dies steht im Gegensatz zu der klaren, aber fragilen Darstellung in einem Bag-of-Words-Modell, bei dem - sofern nicht explizit verwaltet - verschiedene Wörter unabhängig von ihrer Verwendung unterschiedliche Darstellungen haben.

3.4.1.3 Text-/NLP-basierte Merkmale

Es kann auch eine Reihe von zusätzlichen textbasierten Merkmalen erstellt werden, die manchmal zur Verbesserung von Textklassifizierungsmodellen hilfreich sind. Einige Beispiele sind:

- Wortanzahl der Dokumente: Gesamtzahl der Wörter in den Dokumenten
- Zeichenanzahl der Dokumente: Gesamtzahl der Zeichen in den Dokumenten
- Anzahl der Satzzeichen im gesamten Aufsatz: Gesamtzahl der Satzzeichen in den Dokumenten
- Anzahl der Großbuchstaben im gesamten Aufsatz: Gesamtzahl der Wörter mit Großbuchstaben in den Dokumenten
- Anzahl der Wörter für den Titel im gesamten Aufsatz: Gesamtzahl der Wörter für den richtigen Fall (Titel) in den Dokumenten

Darüber hinaus können NLP-basierte Merkmale von Part-of-Speech-Modellen verwendet werden, mit denen z. B. festgestellt werden kann, ob ein Wort ein Substantiv oder ein Verb ist. Anschließend wird die Häufigkeitsverteilung anhand des Part-of-Speech-Tags (Anzahl der Substantive, Verben, Adjektive, Adverbien und Pronomen) ermittelt.

3.4.1.4 Themenmodelle

Unter Themenmodelle (eng. topic models) [45] versteht man ein bekanntes und wichtiges Verfahren des modernen maschinellen Lernens. Sie sind ein häufig verwendetes Instrument zur Textklassifizierung, um die zugrunde liegende Semantik in einem Textkörper zu entdecken. Themenmodelle beruhen auf der Annahme der Bag of Words (BOW), bei der

die Informationen über die Reihenfolge der Wörter ignoriert werden. Bei der Erstellung eines neuen Dokuments wird also eine Verteilung über die Themen gewählt. Danach kann jedes Wort nach dem Zufallsprinzip ausgewählt werden, abhängig von einer Verteilung über die Wörter jedes Themas.
Um die Merkmale der Themenmodellierung zu generieren, gibt es eine bekannte Methode, die *Latent Dirichlet Allocation* (LDA). LDA ist ein iteratives Modell, das von einer festen Anzahl von Themen ausgeht. Jedes Thema wird als eine Verteilung über Wörter dargestellt, und jedes Dokument wird dann als eine Verteilung über Themen dargestellt. Obwohl die Token selbst bedeutungslos sind, geben die Wahrscheinlichkeitsverteilungen über Wörter, die von den Themen geliefert werden, einen Eindruck von den verschiedenen Ideen, die in den Dokumenten enthalten sind.

3.4.1.5 Auswahl der besten Methode

Nach der vorläufigen Analyse beinhaltet jeder Datenpunkt ausschließlich einen vollständigen Satz, der einen einzigen Kompetenz beschreibt. Aus diesem Grund wird *TF-IDF-Vektoren* für diese Arbeit gewählt, um die Dokumente im Korpus darzustellen. Die Vorteile dieses Ansatzes im Vergleich zu anderen sind:

- *TF-IDF* ist ein einfaches Modell, das in diesem speziellen Bereich hervorragende Ergebnisse liefert, wie in Abschnitt 3.6 gesehen werden.
- Die Erstellung von *TF-IDF-Merkmalen* ist ein schneller Prozess, was für kleine Datensätze wie in dieser Arbeit wichtig ist.
- Um Probleme wie Overfitting [7] [47] zu vermeiden, sollte der Prozess der Merkmalserstellung optimiert werden.

3.4.1.6 Abstimmung der Parameters

Bei der Erstellung von Merkmalen mit dieser Methode ist es wichtig, einige Parameter festzulegen, um das beste Ergebnis zu erzielen:

- N-gram range: Ein n-Gramm ist eine zusammenhängende Folge von n Elementen aus einer gegebenen Text- oder Sprachprobe. Bei den Elementen kann es sich je nach Anwendung um Phoneme, Silben, Buchstaben, Wörter oder Basenpaare handeln. Die n-Gramme werden normalerweise aus einem Text- oder Sprachkorpus

gesammelt. Ein n-Gramm der Größe 1 wird als "Unigramm" bezeichnet, Größe 2 ist ein "Bigramm", Größe 3 ist ein "Trigramm".

- Maximum/Minimum Document Frequency: Bei der Erstellung des Vokabulars werden die Begriffe, deren Dokumenthäufigkeit strikt über/unter dem angegebenen Schwellenwert liegt, ignoriert.
- Maximum features: Die besten N Merkmale, geordnet nach der Häufigkeit der Begriffe im Korpus, können ausgewählt werden.

Man kann davon ausgehen, dass Bigramme dazu beitragen, die Leistung des Modells zu verbessern, da sie Wörter berücksichtigen, die in den Dokumenten häufig zusammen vorkommen. Als Mindest-DF wird ein Wert von 2 gewählt, um extrem seltene Wörter loszuwerden, die in nicht mehr als 2 Dokumenten vorkommen, und eine maximale DF von 100%, um keine anderen Wörter zu ignorieren. Die Wahl von 250 als maximale Anzahl von Merkmalen wurde getroffen, um eine mögliche *Overfitting* zu vermeiden, die häufig aus einer großen Anzahl von Merkmalen im Vergleich zur Anzahl der Trainingsbeobachtungen resultiert. Die Tabelle 3.2 gibt einen Überblick über die ausgewählten Parameter.

Parameter	**Wert**
N-gram range	(1,2)
Minimum DF	2
Maximum DF	1.0
Maximum features	250

Tabelle 3.2: Ausgewählte Parameter

Bevor dieser Abschnitt abgeschlossen wird, muss noch ein wichtiger Aspekt erwähnt werden. Für die Berechnung der TF-IDF-Werte wird ein Dokumentenkorpus benötigt, um den Begriff "Inverse Document Frequency" zu berechnen. Wenn also eine einzelne Kompetenz vorhergesagt werden soll, muss dieser Korpus zunächst definiert werden.
In diesem Fall ist dieser Korpus die Menge der Trainingsdokumente. Wenn also TF-IDF-Merkmale aus einer neuen Kompetenz gewonnen werden, werden nur die Merkmale, die im Trainingskorpus vorhanden waren, für diese neue Kompetenz erstellt. Daraus lässt sich schließen, dass die Genauigkeit umso höher ist, je ähnlicher der Trainingskorpus dem Kompetenzbereich ist, der bei der Anwendung des Modells ausgelesen wird.

3.4.2 Reinigung und Vorbereitung des Textes

„Garbage in, garbage out“ [4] ist eine treffende Aussage, die auch auf die Analyse von Texten angewendet werden kann, denn wenn die Vorverarbeitung nicht richtig durchgeführt wird, haben die verwendeten Algorithmen später Probleme, den Inhalt korrekt zu verarbeiten. Die Textvorverarbeitung [9] kann als eine Reihe von Operationen verstanden werden, die notwendig sind, damit eine Maschine Texte automatisch verarbeiten kann. Der Hauptzweck der Textbereinigung besteht also darin, die Textdaten zu systematisieren. Unbereinigte Daten können viele potenzielle Probleme enthalten, wie z. B. falsch geschriebene Wörter, falsche Zeichensetzung, falsche Abstände usw., und die Verwendung ungereinigter Daten kann sogar die Linguistik des Dokuments verzerren und Prozesse zur Informationsextraktion behindern.
Außerdem wird bei NLP-Methoden jedes Wort als eine Variable (Dimension) betrachtet. Einerseits versucht man, den Wortschatz und damit die Dimensionen so klein wie möglich zu halten, andererseits aber auch groß genug, damit keine wichtigen Informationen verloren gehen. Die Entfernung von Rauschen aus dem Dokument kann die Rechenkosten senken und die Leistung von NLP-Modellen erhöhen.
Es ist zu beachten, dass die Textbereinigung ein intensiver Prozess ist. Es wird geschätzt, dass die Textvorverarbeitung etwa 80% der Zeit in Anspruch nimmt, während für die Datenanalyse nur 20% des Aufwands erforderlich sind. Während der Textbereinigung trifft der Forscher subjektive Entscheidungen, die zu verzerrten Analyseergebnissen führen können; daher hat die Textbereinigung einen entscheidenden Einfluss auf die Textanalyse und die Endergebnisse.
Im Rahmen dieser Arbeit sind die Datensätze nicht so *dirty* (Duplizierte Datensätze, unvollständige Sätze, Rechtschreibfehler, usw...), da jeder Datenpunkt bereits bei der Erstellung der csv-Datei sorgfältig angepasst wurde. Daher ist die Arbeit der Textbereinigung viel weniger aufwändig. Die verschiedenen Schritte, die für die Vorverarbeitung von Textdaten erforderlich sind, werden jedoch im Folgenden ausführlich beschrieben.

1. **Tokenisation**: Für die Durchführung von NLP-Aufgaben ist ein robuster Wortschatz erforderlich. Jeder Korpus besteht aus einer Anzahl von Dokumenten [9], die jeweils eine Instanz darstellen und als eindeutiger Bezeichner dienen. Das bedeutet, dass jede Dokumentensammlung (Korpus) aus einzelnen Dokumenten besteht, die durch einzelne Token gebildet werden. Um das erforderliche Vokabular zu erhalten, muss ein Dokument in einzelne Bedeutungs-Token zerlegt werden. Die deutsche

Sprache ist *space-delimited*, was bedeutet, dass die meisten Begriffe durch Leerzeichen getrennt sind.

2. **Reinigung von Sonderzeichen**: Sonderzeichen wie Anführungszeichen müssen aus dem Text entfernt werden, da von ihnen keine Vorhersagekraft erwartet werden kann.

3. **Kleinschreibung und Entfernung von Interpunktion**: In der Regel werden die Umwandlung von Text in Kleinbuchstaben und die Entfernung von Satzzeichen als erste Vorverarbeitungsschritte [9] betrachtet. Textdaten in Kleinbuchstaben sind besonders wichtig, um Wortredundanzen zu vermeiden. Bei der Zählung von Wörtern würden beispielsweise die Begriffe „Kompetenz“ und „kompetenz“ als zwei getrennte Wörter gezählt, was zu einer unerwünschten Vergrößerung der Datenmenge führen würde. Gleichzeitig verursachen Interpunktionszeichen Rauschen in den Daten und tragen nicht zur Analyse bei, weshalb sie entfernt werden sollten. In bestimmten Situationen ist es jedoch sinnvoll, einzelne Sätze zu analysieren; in solchen Fällen sollte das Korpus in einzelne Sätze zerlegt werden, bevor Satzzeichen entfernt werden.

4. **Stemming**: Unter **Stemming** im NLP versteht man ein Prozess, der Varianten eines Stammes/Grundwortes erzeugt. Im Grunde wird ein Grundwort auf seinen Stamm reduziert. So hat „Wolken“ beispielsweise den Wortstamm „Wolk“ und „Fenster“ „Fenst“. Stemming wird verwendet, um die Suche zu verkürzen und Sätze zum besseren Verständnis zu normalisieren.

5. **Entfernung von Stopp-Wörtern**: Die Wörter wie “allerdings” oder “aber” haben keine Vorhersagekraft, da sie wahrscheinlich in vielen Dokumenten vorkommen. Aus diesem Grund können sie Rauschen darstellen, das eliminiert werden kann.

	Studiengang	Kompetenz	geparste_Kompetenz
0	fachübergreifend	Die Studierenden können wichtige mathematische...	studier wichtig mathemat struktur sich verwend
1	fachübergreifend	Die Studierenden beherrschen die logischen und...	studier beherrsch logisch algebra grundlag the...
2	Angewandte Informatik	Die Studierenden können eine präzise und abstr...	studier prazis abstrakt denkweis sowi formal d...
3	fachübergreifend	Die Studierenden können Definitionsprinzipien ...	studier definitionsprinzipi beweistechn unters...
4	Wirtschaftsinformatik	Die Studierenden können Graphen in Maximalflus...	studier graph maximalflussproblem anwend
5	Informatik Technischer Systeme	Die Studierenden können die Methoden der linea...	studier method linear algebra anwend
6	fachübergreifend	Die Studierenden verstehen die Grundlagen, Pri...	studier versteh grundlag prinzipi grenz informat
7	Wirtschaftsinformatik	Die Studierenden besitzen einen Überblick über...	studier besitz uberblick inhalt fragestell met...
8	Wirtschaftsinformatik	Die Studierenden verstehen grundlegende Begrif...	studier versteh grundleg begriff konzept wirts...
9	Wirtschaftsinformatik	Die Studierenden können ein realistisches Bild...	studier realist bild potentiell zukunft berufs...

Abbildung 3.8: Datensatz nach der Reinigung

Die Abbildung 3.8 zeigt den bereinigten Datensatz. Die bereinigten Kompetenzbeschreibungen sind viel prägnanter, da die ursprüngliche Satzstruktur und die Wörter stark verändert wurden. Obwohl die Bedeutung immer noch nachvollziehbar ist, wird es den Menschen wahrscheinlich schwerer fallen, diese Sätze zu verstehen. Im Gegensatz dazu profitieren viele Klassifizierungsansätze stark von dieser Vereinfachung. Einige Gründe dafür wurden bereits oben genannt. Im Allgemeinen werden nur informative Teile des Textes beibehalten, die dem spezifischen Modell helfen, später auf die Aufgabe angewendet zu werden. In diesem Fall sind einfache Modelle nützlicher, da sie von einer geringeren Komplexität profitieren. Fortgeschrittenere Modelle sind jedoch in der Lage, Informationen aus komplexeren Merkmalen zu extrahieren. Daher können sie mit einer geringeren Vereinfachung der Eingabetexte bessere Ergebnisse erzielen. Weitere Einzelheiten werden in Abschnitt 3.6 vorgestellt.

3.4.3 Kodierung von Label

Das Problem der Vorhersage [14] ist in jedem Bereich sehr wichtig, um die Preise und Präferenzen der Menschen zu bewerten. Dieses Problem ist bei verschiedenen Arten von Daten unterschiedlich. Die Daten können nominal oder ordinal sein, sie können mehr oder weniger Kategorien umfassen. Damit eine Kategorie von einem Algorithmus für maschinelles Lernen berücksichtigt werden kann, muss sie kodiert werden, bevor andere Operationen durchgeführt werden können. Bei der Kodierung des Labels [15] wird einfach jedem möglichen Wert einer kategorialen Variablen ein ganzzahliger Wert zuge-

wiesen. Das folgende Schema wurde erstellt, um jedem Studiengang eine numerische ID zuzuordnen:

Studiengang	Code
fachübergreifend	0
Angewandte Informatik	1
Wirtschaftsinformatik	2
Informatik Technischer Systeme	3

Tabelle 3.3: Labelcoding für Studiengänge

3.5 Training und Test

Der nächste entscheidende Schritt bei der Erstellung eines maschinellen Lernmodells ist das Trainieren und Testen des Datensatzes. Ohne Training des Algorithmus konvergieren die Modellparameter nicht auf brauchbare Werte, und das Modell ist nutzlos. Wenn der Algorithmus nach dem Training der Modellparameter nicht getestet wird, gibt es keine Möglichkeit festzustellen, wie gut das Modell tatsächlich ist, was ebenso nutzlos ist.

3.5.1 Train-Test-Split-Verfahren

Der Prozess umfasst das Train-Test-Split-Verfahren, um die Leistung von Algorithmen des maschinellen Lernens abzuschätzen. Bei diesem Verfahren wird ein Datensatz in zwei Teilmengen aufgeteilt. Die erste Teilmenge wird für die Anpassung des Modells verwendet und als *Trainingsdatensatz* bezeichnet. Der zweite Teilsatz wird nicht zum Trainieren des Modells verwendet, stattdessen wird das Eingabeelement des Datensatzes dem Modell zur Verfügung gestellt, dann werden Vorhersagen gemacht und mit den erwarteten Werten verglichen. Dieser zweite Datensatz wird als *Testdatensatz* bezeichnet. Der wesentliche Konfigurationsparameter bei diesem Vorgehen ist dabei die Größe der Trainings- und Testdatensätze. Dieser wird in der Regel als Prozentsatz zwischen 0 und 1 für die Trainings- und Testdatensätze ausgedrückt. Ein *Trainingsdatensatz* mit einer Größe von 0,75 (75%) bedeutet beispielsweise, dass der verbleibende Prozentsatz von 0,25 (25%) dem *Testdatensatz* zugewiesen wird. Die Größe der Datensätze und das Verhältnis zwischen Training und Test [33] können jedoch das Ergebnis der Modelle und damit die Klassifizierungsleistung selbst stark beeinflussen. In der Realität gibt es keinen optimalen Aufteilungsprozentsatz, die Wahl des Prozentsatzes hängt von Faktoren wie

dem Anwendungsfall, der Struktur des Modells, der Dimension der Daten usw. ab.
In dieser Studie wird eine zufällige Aufteilung gewählt, wobei 80% der Beobachtungen den *Trainingsdatensatz* und 20% der Beobachtungen den *Testdatensatz* bilden. Die meisten modernen Algorithmen für maschinelles Lernen haben Parameter, die vor der Ausführung festgelegt werden müssen. Solche Parameter werden als *Hyperparameter* bezeichnet. [42] Beim maschinellen Lernen bzw. Tiefen-Lernen werden *Hyperparameter* als Parameter definiert, deren Werte den Lernprozess kontrollieren und die Werte der Modellparameter bestimmen, die ein Lernalgorithmus schließlich lernt. Das Präfix "hyper_" deutet darauf hin, dass es sich um "Top-Level"-Parameter handelt, die den Lernprozess und die daraus resultierenden Modellparameter steuern.
Die Abstimmung der Hyperparameter erfolgt dann durch Kreuzvalidierung in den Trainingsdaten, Anpassung des endgültigen Modells an diese Daten und anschließende Bewertung mit völlig ungesehenen Daten, um eine möglichst unverzerrte Bewertungsmetrik zu erhalten. Um die Verzerrung zwischen dem gesamten Datensatz und dem Trainingssatz (oder Testsatz) zu verringern, gilt die K-fache Kreuzvalidierung, die im nächsten Unterabschnitt vorgestellt wird, als wissenschaftliche Bewertungsmethode für die Klassifikationsmodellierung.

3.5.2 *K*-fachen Kreuzvalidierung

Die Kreuzvalidierung [34] ist eine statistische Methode zur Bewertung und zum Vergleich von Lernalgorithmen, bei der die Daten in zwei Segmente aufgeteilt werden: eines zum Lernen oder Trainieren eines Modells und das andere zur Validierung des Modells. Bei einer typischen Kreuzvalidierung müssen sich die Trainings- und Validierungssätze in aufeinanderfolgenden Runden überkreuzen, so dass jeder Datenpunkt eine Chance hat, validiert zu werden. Die Grundform der Kreuzvalidierung ist die K-fache Kreuzvalidierung. Andere Formen der Kreuzvalidierung sind Spezialfälle der K-fachen Kreuzvalidierung oder beinhalten wiederholte Runden der K-fachen Kreuzvalidierung.
Bei diesem Verfahren wird der Rohdatensatz in K Teilmengen aufgeteilt. Eine der Teilmengen wird als Testmenge ausgewählt, und die verbleibenden K - 1 Teilmengen werden bei jeder Iteration als Trainingsmenge betrachtet, wie in Abbildung 3.9 [35] dargestellt. Die Klassifizierungsgenauigkeit jedes Klassifizierers wird dann als Durchschnittswert der K Modelle ausgedrückt, die aus der Trainingszahl K erhalten wurden. Daher kann die K-fache Kreuzvalidierungsmethode die Zufälligkeit der Datensatzauswahl wirksam reduzieren und die Zuverlässigkeit des Klassifizierungsmodells wissenschaftlich bewerten.

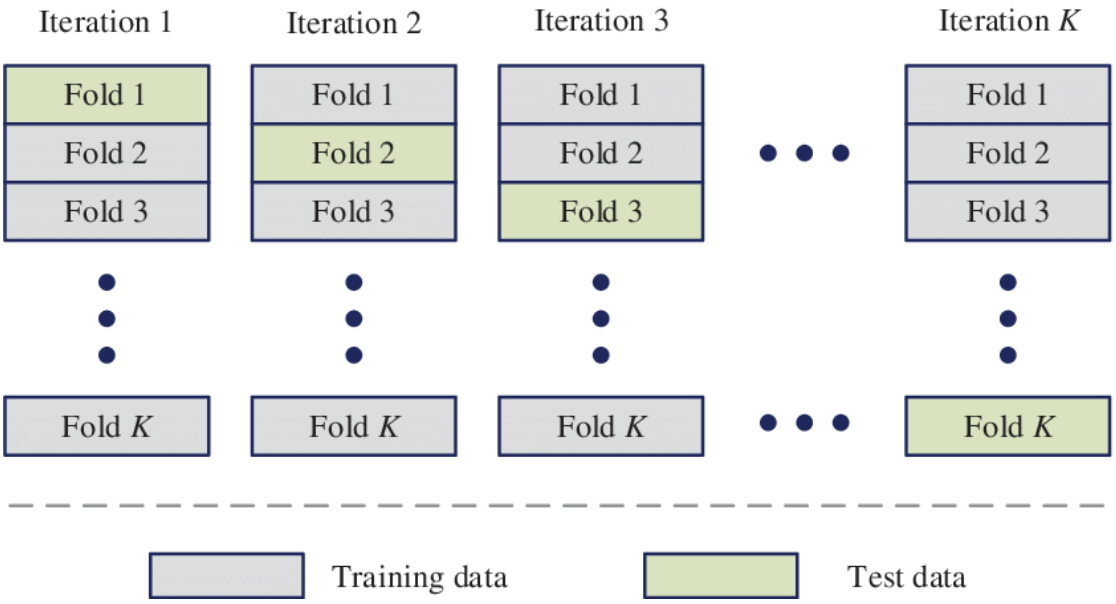


Abbildung 3.9: K-fache Kreuzvalidierungsmethode

3.6 Vorhersagemodelle

Sobald die Daten für das Training bereit sind, werden verschiedene Klassifizierungsmodelle des maschinellen Lernens ausprobiert, um festzustellen, welches Modell am besten für die Daten geeignet ist und die Beziehungen zwischen Punkten und ihren Labels richtig erfasst. Der Grund dafür, dass in dieser Studie nur klassische Modelle des maschinellen Lernens anstelle von Deep-Learning-Modellen verwendet werden, ist die zu geringe Datenmenge, die wahrscheinlich zu **Overfitting** führen würde, die bei ungesehenen Daten nicht gut verallgemeinert werden können.

Die Vorgehensweise zum Trainieren der einzelnen Modelle sieht wie folgt aus:

1. Zunächst werden die Hyperparameter bestimmt, die einen größeren Einfluss auf das Modellverhalten haben könnten. Es ist jedoch zu beachten, dass eine große Anzahl von Parametern viel Rechenzeit erfordern würde.

2. Zweitens wird die Metrik festgelegt, die bei der Messung der Leistung eines Modells zu verwenden ist. In diesem Fall wird die Genauigkeit (**accuracy**) verwendet.

3. Danach wird ein **Randomized Search Cross Validation** [5] Prozess mit *3*-facher Kreuzvalidierung (mit 50 Iterationen) durchgeführt, um die Region der Hyperparameter zu finden, in der die höchsten Werte für die Genauigkeit erzielt werden.

4. Sobald diese Region bestimmt ist, wird ein **Grid Search Cross Validation** [5] Prozess mit *3*-facher Kreuzvalidierung angewendet, um die beste Kombination von Hyperparametern erschöpfend zu finden.

5. Anschließend wird die Genauigkeit für die Trainingsdaten und die Testdaten berechnet. Daraufhin werden der Klassifizierungsbericht und die Konfusionsmatrix ausgedruckt.

6. Schließlich wird die Genauigkeit eines Modells mit Standard-Hyperparametern und abgestimmten Parametern verglichen, um festzustellen, welches Modell bessere Ergebnisse liefert.

Ein Vorteil dieses Ansatzes ist, dass mit der **Randomized Search** ein viel größerer Wertebereich für jeden Hyperparameter abgedeckt werden kann, ohne dass die Ausführungszeit sehr hoch ist. Sobald der Bereich für jeden Parameter eingegrenzt ist, wird sichtbar, worauf sich die Suche konzentrieren muss, und jede Kombination von Einstellungen, die ausprobiert werden soll, wird explizit angegeben. Der Grund für die Wahl von $K = 3$ als Anzahl der Faltungen und 50 Iterationen bei der randomisierten Suche liegt in der Überlegung, einen Kompromiss zwischen kürzerer Ausführungszeit und dem Testen einer großen Anzahl von Kombinationen zu finden.
Die in dieser Arbeit implementierten Modelle werden in Unterabschnitten genauer erklärt.

3.6.1 Decision Tree

Zunächst wird einer der beliebtesten und am weitesten verbreiteten Algorithmen des maschinellen Lernens, der **Decision Tree** (DT), vorgestellt. In ihrer einfachsten Beschreibung ist DT [29] ein Ansatz zur Klassifizierung durch Divide-and-Conquer. Mit Hilfe von DT können Merkmale entdeckt und Muster in großen Datenbanken extrahiert werden, die für die Unterscheidung und Vorhersagemodellierung wichtig sind. Diese Eigenschaften in Verbindung mit ihrer intuitiven Interpretation sind einige der Gründe,

warum DT seit mehr als zwei Jahrzehnten in großem Umfang sowohl für die explorative Datenanalyse als auch für die prädiktive Modellierung von Anwendungen eingesetzt werden.

3.6.1.1 Klassifizierungsalgorithmus

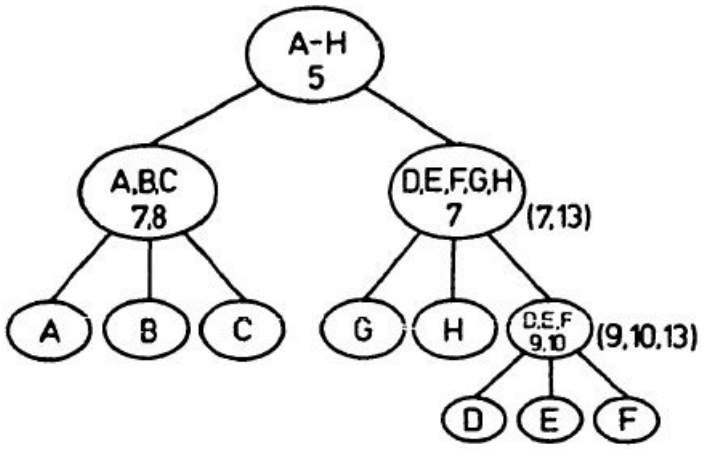


Abbildung 3.10: Ein einfacher Decision Tree, der die Terminologie veranschaulicht [29]

Der DT-Klassifikator zeichnet sich dadurch aus, dass eine unbekannte Probe mit Hilfe einer oder mehrerer Entscheidungsfunktionen sukzessive in eine Klasse eingeordnet wird. Diese Klassifizierungsstrategie kann durch ein Baumdiagramm beschrieben werden (siehe Abbildung 3.10). Zur Verarbeitung wird der Baum als Zeichenkette kodiert, so dass eine eindeutige Beziehung zwischen Zeichenkette und DT besteht. Die Zeichenfolge wird im Computer dekodiert, und es werden Zeiger eingerichtet, um den entsprechenden Klassifizierungspfad für jedes Datenmuster zu definieren.
Im Allgemeinen besteht ein DT aus einem Wurzelknoten, einer Reihe von inneren Knoten und einer Reihe von Endknoten. Der Wurzelknoten und die inneren Knoten, die zusammen als nichtterminale Knoten bezeichnet werden, sind mit Entscheidungsstufen verbunden; die terminalen Knoten stellen die endgültigen Klassifizierungen dar (siehe Abbildung 3.10). Mit dem Wurzelknoten ist die gesamte Menge der Klassen verbunden, in die eine Probe klassifiziert werden kann. Die Menge der Knoten auf einer bestimmten Ebene des Baums, d. h., die alle den gleichen “Abstand” von der Wurzel haben,

wird als Ebene bezeichnet. Jeder Knoten besteht aus einem Satz von unterschiedlichen Klassen, einer Menge zu verwendender Merkmale und einer Entscheidungsregel für die Klassifizierung.

3.6.1.2 Einstellung der Hyperparameter

Als Klassifikatoren werden DTs durch Regeln dargestellt, die als Baum strukturiert sind. Sie werden vor allem wegen ihrer verständlichen Natur, die dem menschlichen Denken ähnelt, häufig verwendet. DT-Induktionsalgorithmen haben gegenüber vielen anderen Algorithmen des maschinellen Lernens mehrere Vorteile, z. B. Robustheit gegenüber Rauschen (fehlende Werte, unausgewogene Klassen), geringe Rechenkosten und die Fähigkeit, mit redundanten Attributen umzugehen. Die für die Hyperparameter [25] des Algorithmus für maschinelles Lernen gewählten Werte wirken sich direkt auf die Vorhersageleistung der durch sie induzierten Modelle aus.
Die folgenden Hyperparameter werden abgestimmt:

1. **criterion: {“gini”, “entropy”}, default=”gini”**: Die Funktion zur Messung der Qualität einer Aufteilung. Unterstützte Kriterien sind „gini“ für die Gini-Unreinheit und „Entropie“ für den Informationsgewinn. Der Entscheidungsbaum verwendet die Unreinheit zur Aufteilung seiner Knoten. Die Unreinheit ist ein Maß für die Homogenität der Beschriftungen an einem Knoten. Beim Informationsgewinn wird das Entropiemaß als Maß für die Unreinheit verwendet und ein Knoten wird so aufgeteilt, dass er den größten Informationsgewinn bietet. Die Gini-Unreinheit hingegen misst die Divergenzen zwischen den Wahrscheinlichkeitsverteilungen der Werte des Zielattributs und teilt einen Knoten so auf, dass er die geringste Unreinheit ergibt. Der einzige Unterschied besteht darin, dass die Entropie etwas langsamer berechnet werden kann, weil eine logarithmische Funktion berechnet werden muss:

$$Gini : Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

$$Entropy : H(E) = -\sum_{j=1}^{c} p_j log p_j$$

2. **max_depth: int, default=None**: Die maximale Tiefe des Baums. Wenn *None*, dann werden die Knoten so lange erweitert, bis alle Blätter **pure** sind (wenn alle

Daten zu einer einzigen Klasse gehören) oder bis alle Blätter weniger als *min_samples_split* samples enthalten. Die theoretische maximale Tiefe, die ein DT erreichen kann, ist um eins geringer als die Anzahl der Trainingsstichproben. Dabei ist zu beachten, dass es sich um die Anzahl der Trainingsstichproben und nicht um die Anzahl der Merkmale handelt, da die Daten mehrfach auf dasselbe Merkmal aufgeteilt werden können. Im Fall von *None* expandiert scikit-learn (eine Softwarebibliothek für maschinelles Lernen in der Programmiersprache Python) die Knoten so lange, bis alle Blätter rein sind, d. h. die Blätter haben nur dann Labels, wenn *min_samples_leaf* auf default gesetzt ist, wobei der Standardwert eins ist. Sollte *min_samples_split* angegeben werden, werden die Knoten so lange expandiert, bis alle Blätter weniger als die Mindestanzahl an Stichproben enthalten. Scikit-learn entscheidet sich für die eine oder die andere Methode, je nachdem, welche die maximale Tiefe für den Baum ergibt.
Im Wesentlichen wird das Modell umso komplexer, je tiefer der Baum wachsen kann, da mehr Splits vorhanden sind und mehr Informationen über die Daten erfasst werden. Dies ist eine der Hauptursachen für das *Overfitting* von DT, da das Modell perfekt an die Trainingsdaten angepasst ist und nicht in der Lage ist, die Testdaten gut zu verallgemeinern.

3. **min_samples_split: int or float, default=2**: Die Mindestanzahl von Stichproben, die erforderlich ist, um einen internen Knoten zu teilen. Wenn *int*, dann gilt *min_samples_split* als Mindestanzahl. Wenn *float*, dann ist *min_samples_split* ein Bruch und $ceil(min\_samples\_split * n\_samples)$ ist die Mindestanzahl von Stichproben für jeden Split. Die idealen *min_samples_split-Werte* liegen für den Klassifizierungs- und Regressionsbäume (CART) Algorithmus, der in scikit-learn implementiert ist, zwischen 1 und 40. min_samples_split wird zur Kontrolle der *overfitting* verwendet [26]. Höhere Werte verhindern, dass ein Modell Beziehungen lernt, die für einen Baum ausgewählte Stichprobe sehr spezifisch sein könnten. Zu hohe Werte können auch zu einer *underfitting* führen.

4. **min_samples_leaf: int or float, default=1**: Die Mindestanzahl von Stichproben, die für einen Blattknoten erforderlich sind. Ein Splitpunkt in beliebiger Tiefe wird nur berücksichtigt, wenn er mindestens *min_samples_leaf* Trainingsstichproben in jedem der linken und rechten Zweige hinterlässt. Dies kann zu einer Glättung des Modells führen, insbesondere bei Regressionen. Wenn *float*, dann ist *min_samples_leaf* ein Bruch und $ceil(min\_samples\_leaf * n\_samples)$ ist die Mindestanzahl von Stichproben für jeden Split. Ähnlich wie *min_samples_*-

split wird auch *min_samples_leaf* zur Kontrolle der *overfitting* verwendet, indem festgelegt wird, dass jedes Blatt mehr als ein Element hat. Auf diese Weise wird sichergestellt, dass der Baum den Trainingsdatensatz nicht *overfit* kann, indem er eine Reihe kleiner Zweige ausschließlich für jeweils eine Probe erzeugt. Die idealen *min_samples_leaf-Werte* liegen für den CART-Algorithmus zwischen 1 und 20. *min_samples_split* und *min_samples_leaf* sind am meisten für die Leistung der endgültigen Bäume aus der relativen Wichtigkeitsanalyse verantwortlich [26].

5. **max_features: int, float or {"auto", "sqrt", "log2"}, default=None**: Die Anzahl der Merkmale, die bei der Suche nach dem besten Split zu berücksichtigen sind, wenn

   - *int*, dann werden bei jedem Split *max_features* features berücksichtigt.
   - *float*, dann ist *max_features* ein Bruch und $int(max\_features * n\_features)$ features werden bei jedem Split berücksichtigt.
   - *"auto"*, dann $max\_features = sqrt(n\_features)$.
   - *"sqrt"*, dann $max\_features = sqrt(n\_features)$.
   - *"log2"*, dann $max\_features = log2(n\_features)$.
   - *None*, dann $max\_features = n\_features$.

   Bei jeder Aufteilung untersucht der Algorithmus eine Reihe von Merkmalen und wählt das Merkmal mit der optimalen Metrik unter Verwendung von Gini-Unreinheit oder Entropie aus und erstellt zwei Zweige entsprechend diesem Merkmal. Da es sehr rechenintensiv ist, jedes Mal alle Merkmale zu prüfen, kann man mit den verschiedenen *max_features-Optionen* nur einige von ihnen überprüfen. Eine weitere Verwendung von *max_features* besteht darin, die *overfitting* zu begrenzen. Durch die Wahl einer geringeren Anzahl von Merkmalen können die Stabilität des Baums erhöhen lassen und die Varianz und *overfitting* verringern.

Nach der Kombination zwischen **Randomized Search Cross Validation** und **Grid Search Cross Validation** werden den Parametern die Werte in Tabelle 3.4 zugewiesen:

Parameter	Wert
criterion	gini
max_depth	20
max_features	2
min_samples_leaf	1
min_samples_split	2

Tabelle 3.4: Die besten Hyperparameter für DT

3.6.1.3 Konfusionsmatrix

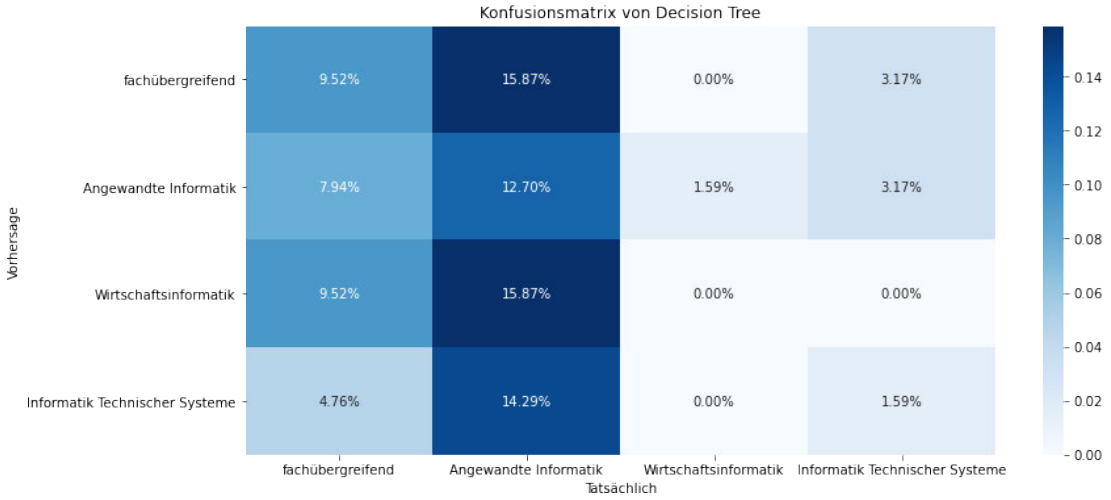


Abbildung 3.11: Konfusionsmatrix von Decision Tree

Die Abbildung 3.11 beschreibt die Vorhersageergebnisse des ersten Klassifizierungsmodells, **Decision Tree**. Es wird deutlich, dass die Effizienz des Modells gering ist. Es kann den Studiengang **Angewandte Informatik** am besten vorhersagen, aber die Genauigkeit beträgt nur 12,70%. Für **Wirtschaftsinformatik** hat es sogar gar nichts richtig vorhergesagt. Daraus lässt sich schließen, dass dieses Modell für die Klassifizierungsaufgabe überhaupt nicht geeignet ist.

3.6.2 Random Forest

Da ein Entscheidungsbaum für die Klassifizierungsaufgabe nicht ausreicht, wird anschließend eine erweiterte Variante des Entscheidungsbaums ausprobiert. Aufgrund der Ein-

fachheit des Algorithmus und der herausragenden Klassifizierungsleistung bei hochdimensionalen Daten ist **Random Forest** (RF) zu einer vielversprechenden Methode für die Textkategorisierung geworden. Der RF ist eine beliebte Klassifizierungsmethode, die aus einer Reihe von Klassifizierungsbäumen besteht. Eines der populärsten Verfahren zur Erstellung von RF, das von Breiman vorgeschlagen wurde, besteht darin, an jedem Knoten einen Unterraum von Merkmalen zufällig auszuwählen, um Zweige von Entscheidungsbäumen zu erzeugen, dann die Bagging-Methode zu verwenden, um Trainingsdatensätze für die Erstellung einzelner Bäume zu erzeugen, und schließlich alle einzelnen Bäume zu kombinieren, um ein Zufallsbaummodell zu bilden [46].

3.6.2.1 Klassifizierungsalgorithmus

Ein wesentlicher Begriff in diesem Modell ist die *ensemble*-Technik, die die Kombination mehrerer Modelle bezeichnet. RF arbeitet nach dem Bagging-Prinzip (auch bekannt als *Bootstrap-Aggregation*), d.h. es wird eine andere Trainingsuntermenge aus den Trainingsdaten der Stichprobe mit Ersetzung erstellt und die endgültige Ausgabe basiert auf einer Mehrheitsentscheidung. Das Konzept der *Bootstrap-Aggregation* wird in der Abbildung 3.12 ausführlicher beschrieben.

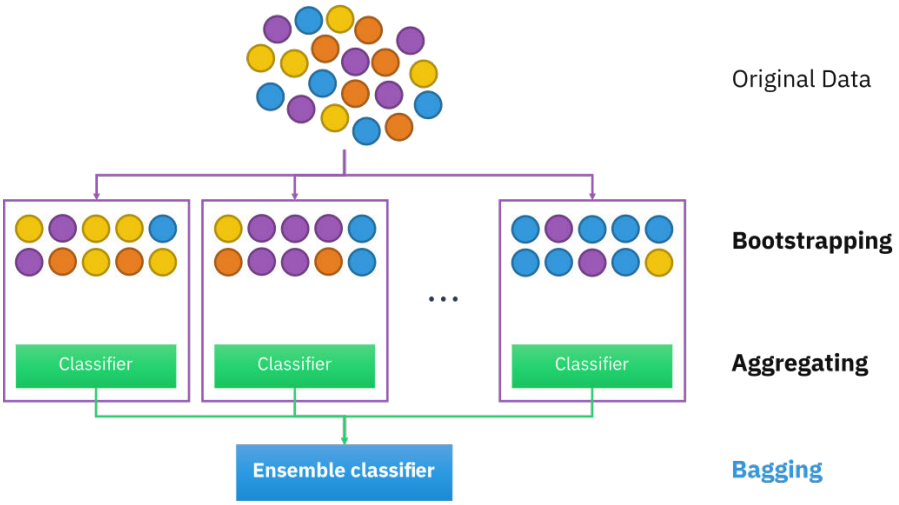


Abbildung 3.12: Konzept der Bootstrap-Aggregation

Beim *Bagging* wird eine Zufallsstichprobe aus dem Datensatz ausgewählt. Daher wird jedes Modell aus den Stichproben (Bootstrap-Samples) generiert, die durch die Originaldaten mit Ersetzung bereitgestellt werden, was als *Row Samplings* bekannt ist. Dieser

Schritt des *Row Samplings* mit Ersetzung wird *Bootstrap* genannt. Nun wird jedes Modell unabhängig voneinander trainiert, was zu den Ergebnissen der einzelnen Modelle führt. Die endgültige Ausgabe basiert auf einer Mehrheitsentscheidung, nachdem die Ergebnisse aller Modelle kombiniert wurden. Dieser Schritt, bei dem alle Ergebnisse kombiniert werden und die Ausgabe auf der Grundlage der Mehrheitsabstimmung erfolgt, wird als *Aggregation* bezeichnet.
Im Allgemeinen sieht der RF-Algorithmus [23] zur Klassifizierung wie folgt aus:

1. Ziehe n_{tree} Bootstrap-Stichproben aus den ursprünglichen Daten.
2. Für jede der Bootstrap-Stichproben wird ein ungekürzter Klassifikationsbaum mit der folgenden Änderung erstellt: An jedem Knoten wird nicht die beste Aufteilung unter allen Prädiktoren, sondern eine zufällige Auswahl m_{try} der Prädiktoren getroffen und dann die beste Aufteilung unter diesen Variablen vorgenommen. (Bagging kann als Spezialfall von RF betrachtet werden, der auftritt, wenn $m_{try} = p$, die Anzahl der Prädiktoren, ist).
3. Vorhersage neuer Daten durch Aggregation der Vorhersagen der n_{tree}-Bäume (d.h. Mehrheitsentscheidungen für Klassifizierung oder Durchschnitt für Regression).

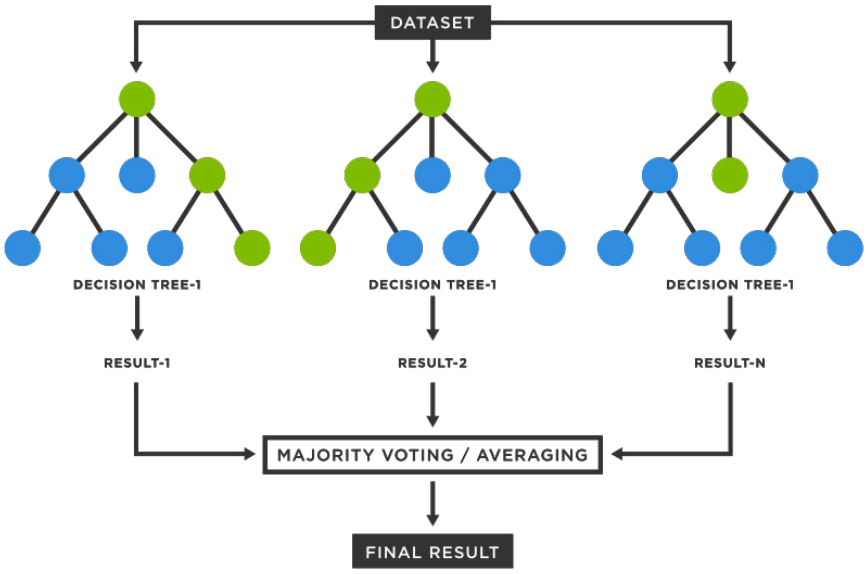


Abbildung 3.13: Diagramm eines Random-Forest. URL

Die Abbildung 3.13 gibt einen Überblick darüber, wie der RF-Algorithmus vom Datensatz bis zum Endergebnis vorgeht.

3.6.2.2 Einstellung der Hyperparamete

Der Random-Forest-Algorithmus hat mehrere Hyperparameter, die vom Benutzer eingestellt werden müssen, z. B. die Anzahl der Beobachtungen, die für jeden Baum zufällig gezogen werden und ob sie mit oder ohne Ersetzung gezogen werden, die Anzahl der Variablen, die für jeden Split zufällig gezogen werden, die Splitting-Regel, die Mindestanzahl von Stichproben, die ein Knoten enthalten muss, und die Anzahl der Bäume. Es ist allgemein bekannt, dass RF in den meisten Fällen recht gut funktioniert mit den Standardwerten der in den Softwarepaketen angegebenen Hyperparameter. Dennoch kann die Anpassung der Hyperparameter die Leistung von RF verbessern [32].
Einige Parameter sind analog zum Decision Tree (Unterunterabschnitt 3.6.1.2) und werden in diesem Abschnitt nicht erwähnt. Die folgenden Hyperparameter werden abgestimmt:

1. **n_estimators: int, default=100**: Dies ist die Anzahl der Bäume, die der Benutzer erstellen möchte, bevor er die maximale Stimme oder den Durchschnitt der Vorhersagen nimmt. Eine höhere Anzahl von Bäumen führt zu besserer Leistung, macht den Code aber langsamer. Man soll einen Wert wählen, der so hoch ist, wie es der Prozessor verkraften kann, denn dadurch werden die Vorhersagen des Modells stärker und stabiler.

2. **bootstrap: bool, default=True**: Ob bei der Erstellung von Bäumen Bootstrap-Stichproben verwendet werden. Wenn False, wird der gesamte Datensatz zur Erstellung jedes Baums verwendet.

Nach der Kombination zwischen **Randomized Search Cross Validation** und **Grid Search Cross Validation** werden den Parametern die Werte in Tabelle 3.5 zugewiesen:

Parameter	Wert
n_estimators	600
max_depth	70
max_features	sqrt
min_samples_leaf	1
min_samples_split	10
bootstrap	True

Tabelle 3.5: Die besten Hyperparameter für RF

3.6.2.3 Konfusionsmatrix

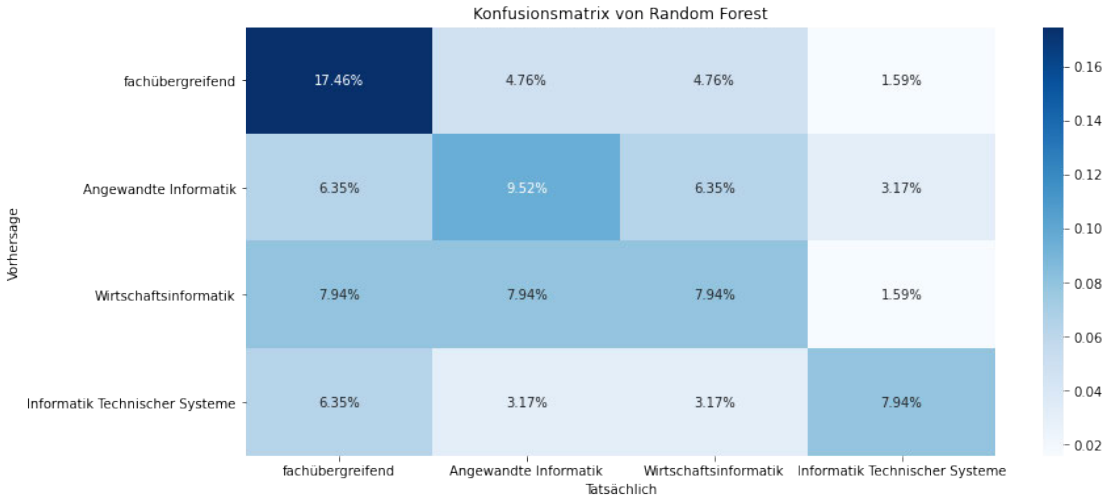


Abbildung 3.14: Konfusionsmatrix von Random Forest

Im Gegensatz zu **Decision Tree** hat **Random Forest** eine deutlich höhere Genauigkeit. In der Abbildung 3.14 ist zu erkennen, dass das Modell zumindest einige Kompetenzen im Studiengang **Wirtschaftsinformatik** korrekt identifizieren kann. Im Gegensatz zum vorherigen Modell hat der Bereich **fachübergreifend** die höchste Genauigkeit und liegt bei 17.46%, ein nicht so schlechter Wert.

3.6.3 Support Vector Machine

Support Vector Machine (SVM) wurde erstmals 1992 von Boser, Guyon und Vapnik in COLT-92 [1] vorgestellt. SVMs sind eine Reihe verwandter überwachter Lernme-

thoden, die für Klassifizierung und Regression verwendet werden. Sie gehören zu einer Familie von verallgemeinerten linearen Klassifikatoren. Anders ausgedrückt, ist die SVM ein Klassifizierungs- und Regressionsvorhersagewerkzeug, das die Theorie des maschinellen Lernens nutzt, um die Vorhersagegenauigkeit zu maximieren und gleichzeitig und dabei automatisch eine Überanpassung an die Daten zu vermeiden. SVMs können als Systeme definiert werden, die den Hypothesenraum einer linearen Funktion in einem hochdimensionalen Merkmalsraum verwenden und mit einem Lernalgorithmus aus der Optimierungstheorie trainiert werden, der einen aus der statistischen Lerntheorie abgeleiteten Lernbias implementiert [17].

3.6.3.1 Klassifizierungsalgorithmus

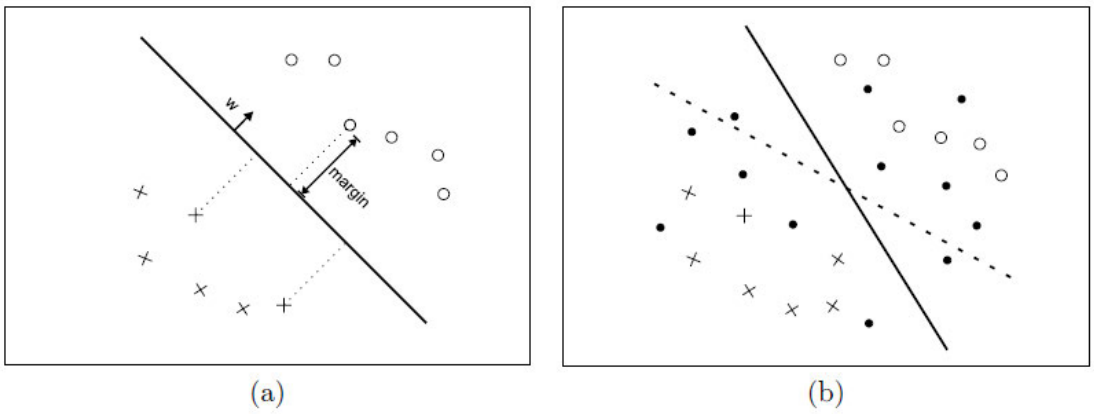


Abbildung 3.15: (a) Eine einfache lineare Support-Vektor-Maschine. (b) Eine SVM (gepunktete Linie) und eine transduktive SVM (durchgezogene Linie). Die durchgezogenen Kreise stellen unbeschriftete Instanzen dar. [39]

SVMs im Rahmen der binären Klassifikation wird betrachtet. Es gibt Trainingsdaten $\{x_1...x_n\}$, die Vektoren in einem Raum $X \subseteq \mathbf{R}^d$ sind. Die Labels $\{y_1...y_n\}$ werden auch vergeben, $y_i \in \{-1, 1\}$. In ihrer einfachsten Form sind SVMs Hyperebenen, die die Trainingsdaten Daten durch einen maximalen Rand trennen (siehe Abbildung 3.15). Alle Vektoren, die auf einer Seite der Hyperebene liegen, werden als -1 bezeichnet, und alle Vektoren, die auf der anderen Seite liegen, werden als 1 bezeichnet. Die Trainingsinstanzen Instanzen, die der Hyperebene am nächsten liegen, werden als *Support-Vektoren* bezeichnet. Allgemeiner ausgedrückt, erlauben SVMs die Projektion der ursprünglichen Trainingsdaten im Raum X auf einen höherdimensionalen Merkmalsraum F über einen

Mercer-Kernel-Operator K. Mit anderen Worten, die Menge der Klassifikatoren in der Form wird bezeichnet:

$$f(x) = \left(\sum_{i=1}^{n} \alpha_i K(x_i, x) \right).$$

Wenn K die Bedingung von Mercer (Burges, 1998) erfüllt, das gilt: $K(u, v) = \Phi(u) \cdot \Phi(v)$ wobei $\Phi : X \to F$ und "$\cdot$" ein inneres Produkt bezeichnet. f kann umgeschrieben werden als:

$$f(x) = w \cdot \Phi(x), wobei w = \sum_{i=1}^{n} \alpha_i \Phi(x_i).$$

Durch die Verwendung von K werden die Trainingsdaten also implizit in einen anderen (oft höherdimensionalen) Merkmalsraum F projiziert. Die SVM berechnet dann das $\alpha_i s$, das der Hyperebene mit der maximalen Marge in F entspricht. Durch die Wahl verschiedener Kernel-Merkmale können die Trainingsdaten aus X implizit in Räume F projiziert werden, für die die Hyperebenen in F den komplexeren Entscheidungsgrenzen im ursprünglichen Raum X entsprechen.
Zwei häufig verwendete Kernel sind der Polynom-Kernel, der durch $K(u, v) = (u \cdot v + 1)^p$ gegeben ist und polynomielle Grenzen des Grades p im ursprünglichen Raum X^1 induziert, und der Radialbasisfunktions-Kernel $K(u, v) = (e^{-\gamma(u-v)\cdot(u-v)})$, der Grenzen induziert, indem er gewichtete Gauß auf wichtige Trainingsinstanzen legt. Für den größten Teil dieser Arbeit wird angenommen, dass der Modulus der Merkmalsvektoren der Trainingsdaten konstant ist, d.h. für alle Trainingsinstanzen $x_i, \| \Phi(x_i) \| = \lambda$ für einige feste λ. Die Größe $\| \Phi(x_i) \|$ ist bei Radialbasisfunktionskernen immer konstant, so dass die Annahme für diesen Kernel keine Wirkung hat. Damit $\| \Phi(x_i) \|$ mit den Polynomkernen konstant ist, muss es verlangen, dass $\| x_i \|$ konstant ist [39].

3.6.3.2 Einstellung der Hyperparamete

Der Prozess der Schätzung des für die Entwicklung eines Softwareprodukts erforderlichen Aufwands wird als Software-Aufwandsschätzung (SEE) bezeichnet. Hyperparameter sind Wertesätze, die vor der Konstruktion des SEE-Modells definiert werden und die Leistung

des Modells beeinflussen können. Geeignete Hyperparameterwerte können die Vorhersagegenauigkeit des Modells im Vergleich zum Standardwert erhöhen [40].
Die folgenden Hyperparameter werden abgestimmt:

1. **C: float, default=1.0**: Regularisierungsparameter. Die Stärke der Regularisierung ist umgekehrt proportional zu C und muss streng positiv sein. Der Parameter C fügt eine Strafe (quadrierte $l2$-Strafe) für jeden falsch klassifizierten Datenpunkt hinzu. Wenn C klein ist, ist die Strafe für falsch klassifizierte Punkte gering, so dass eine Entscheidungsgrenze mit einer großen Spanne auf Kosten einer größeren Anzahl von Fehlklassifikationen gewählt wird. Wenn C groß ist, versucht SVM, die Anzahl der falsch klassifizierten Beispiele aufgrund der hohen Strafe zu minimieren, was zu einer Entscheidungsgrenze mit einem kleineren Spielraum führt. Der Regularisierungsparameter steuert den Kompromiss zwischen einer glatten Entscheidungsgrenze und der richtigen Klassifizierung der Trainingspunkte. Ein großer Wert von C bedeutet, dass mehr Trainingspunkte korrekt klassifiziert werden.

2. **kernel: {'linear', 'poly', 'rbf', 'sigmoid'} or callable, default='rbf'**: Gibt den Kernel-Typ an, der im Algorithmus verwendet werden soll. Wird keiner angegeben, so wird 'rbf' (Radial Basis Function) verwendet. Wenn ein Callable angegeben wird, wird es zur Vorberechnung der Kernelmatrix aus Datenmatrizen verwendet; diese Matrix sollte ein Array der Form $(n\_samples, n\_samples)$. Durch dieses Parameter wählt das Modell die Art der Hyperebene aus, die zur Trennung der Daten verwendet wird. Bei der Verwendung von "linea" wird eine lineare Hyperebene (eine Linie im Falle von 2D-Daten) verwendet. Bei 'rbf' und 'poly' wird eine nicht lineare Hyperebene verwendet.Es gibt jedoch drei gängige Kernel-Typen:

   - Linear: $K(x_1, x_2) = x_1 \cdot x_2$
   - Polynomial: $K(x_1, x_2) = (x_1 \cdot x_2 + 1)^p$
   - Radial basis function (RBF): $K(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2}$

3. **degree: int, default=3**: Grad der Polynom-Kernel-Funktion ('poly'). Wird von allen anderen Kerneln ignoriert. Es handelt sich im Grunde um den Grad des Polynoms, das verwendet wird, um die Hyperebene zur Aufteilung der Daten zu finden.

4. **gamma: {'scale', 'auto'} or float, default='scale'**: Kernelkoeffizient für 'rbf', 'poly' und 'sigmoid':

- wenn gamma='scale' (default) übergeben wird, wird $1/(n\_features * X.var())$ als Gammawert verwendet
- wenn 'auto', verwendet $1/n\_features$.

Es definiert, wie weit der Einfluss eines einzelnen Trainingsbeispiels reicht. Ein niedriger Wert bedeutet, dass jeder Punkt eine große Reichweite hat, und ein hoher Gamma-Wert bedeutet, dass jeder Punkt eine geringe Reichweite hat. Wenn gamma einen sehr hohen Wert hat, hängt die Entscheidungsgrenze nur von den Punkten ab, die sehr nahe an der Linie liegen, was dazu führt, dass einige der Punkte, die sehr weit von der Entscheidungsgrenze entfernt sind, ignoriert werden. Dies liegt daran, dass die näheren Punkte mehr Gewicht erhalten und eine wackelige Kurve entsteht, wie im vorherigen Diagramm gezeigt wird, während bei einem niedrigen Gamma-Wert auch die weit entfernten Punkte ein beträchtliches Gewicht erhalten und eine linearere Kurve entsteht.

Nach der Kombination zwischen **Randomized Search Cross Validation** und **Grid Search Cross Validation** werden den Parametern die Werte in Tabelle 3.6 zugewiesen:

Parameter	Wert
C	0.0001
kernel	poly
degree	3
gamma	100

Tabelle 3.6: Die besten Hyperparameter für SVM

3.6.3.3 Konfusionsmatrix

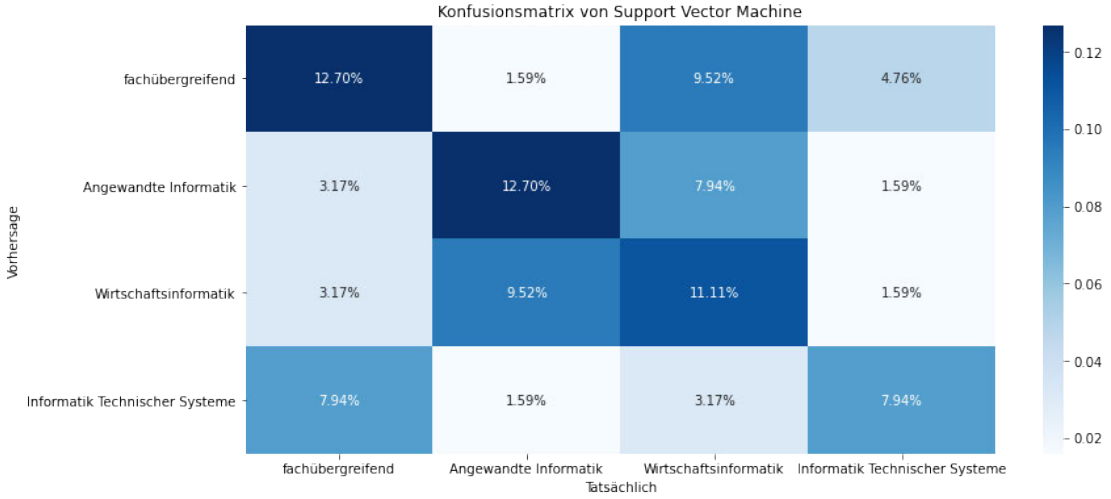


Abbildung 3.16: Konfusionsmatrix von Support Vector Machine

Im Vergleich zu den beiden oberen Modellen hat SVM eine recht gute Qualität bei der Klassifizierung. Die Genauigkeit der Bereiche “fachübergreifend”, “Angewandte Informatik” und “Wirtschaftsinformatik” ist nahezu ausgeglichen (siehe Abbildung 3.16). SVM kann weniger Kompetenzen der “Informatik Technische Systeme” richtig erkennen, aber die Differenz ist nicht so groß.

3.6.4 K-Nearest Neighbors

Der **k-Nearest Neighbors** (kNN) ist einer der beliebtesten Algorithmen für die Textkategorisierung. Viele Forscher haben festgestellt, dass der kNN in ihren Experimenten mit verschiedenen Datensätzen sehr gute Leistungen erzielt. Die Idee hinter dem kNN ist recht einfach. Um ein neues Dokument zu klassifizieren, findet das System die k nächsten Nachbarn unter den Trainingsdokumenten und verwendet die Kategorien der k nächsten Nachbarn zur Gewichtung der Kategoriekandidaten. Einer der Nachteile des kNNs ist seine Effizienz, da er ein Testdokument mit allen Mustern im Trainingssatz vergleichen muss. Außerdem hängt die Leistung dieses Algorithmus in hohem Maße von zwei Faktoren ab, nämlich von einer geeigneten Ähnlichkeitsfunktion und einem geeigneten Wert für den Parameter k [21].

3.6.4.1 Klassifizierungsalgorithmus

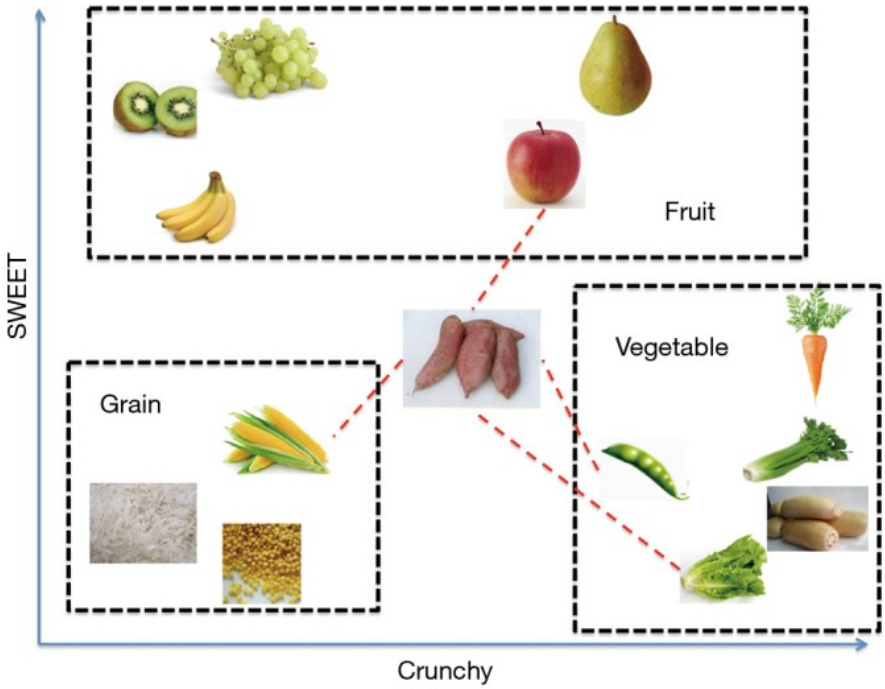


Abbildung 3.17: Veranschaulichung der Funktionsweise des K-Nearest-Neighbors Algorithmus.

Der kNN-Klassifikator klassifiziert unbeschriftete Beobachtungen, indem er sie der Klasse der ähnlichsten beschrifteten Beispiele zuordnet. Die Merkmale der Beobachtungen werden sowohl für den Trainings- als auch für den Testdatensatz erfasst. So lassen sich beispielsweise Obst, Gemüse und Getreide durch ihre Knackigkeit und Süße unterscheiden (siehe Abbildung 3.17). Um sie in einem zweidimensionalen Diagramm darstellen zu können, werden nur zwei Merkmale verwendet. In Wirklichkeit kann es eine beliebige Anzahl von Prädiktoren geben, und das Beispiel kann erweitert werden, um eine beliebige Anzahl von Merkmalen einzubeziehen. Im Allgemeinen sind Früchte süßer als Gemüse. Körner sind weder knusprig noch süß. Ziel ist es, zu bestimmen, zu welcher Kategorie die Süßkartoffel gehört. In diesem Beispiel werden vier nächstgelegene Lebensmittel ausgewählt, nämlich Apfel, grüne Bohne, Salat und Mais. Da das Gemüse die meisten Stimmen erhält, wird die Süßkartoffel der Klasse Gemüse zugeordnet [49].

3.6.4.2 Einstellung der Hyperparameter

Der einzige Parameter, der einen signifikanten Einfluss auf dieses Modell hat, ist die Anzahl k der Nachbarn, die standardmäßig für Abfragen verwendet werden. In Scikit-Learn ist sie als **n_neighbors** definiert. In dieser Arbeit wird eine **Grid Search Cross Validation** eingesetzt, um den besten Wert von K zu finden, der zu einer besseren Erkennungsgenauigkeit von Kompetenzen führen kann. Nach der Suche ergibt der beste Wert von k 11.

3.6.4.3 Konfusionsmatrix

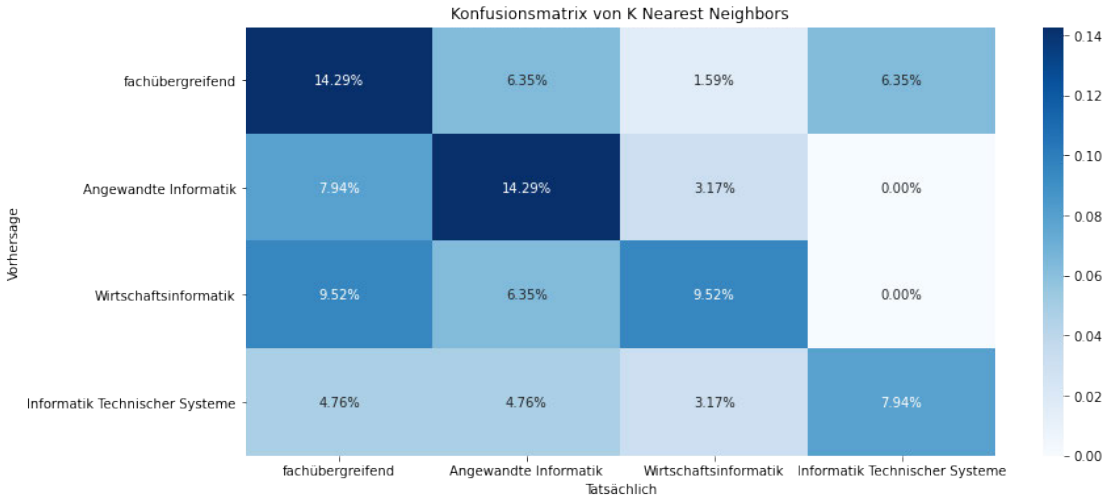


Abbildung 3.18: Konfusionsmatrix von K-Nearest Neighbors

Ähnlich wie SVM (Unterunterabschnitt 3.6.3.3) kann kNN am häufigsten Kompetenzen in den Bereichen **fachübergreifend** und **Angewandte Informatik** korrekt identifizieren. Das **Informatik Technischer Systeme** hat wie üblich die schlechteste Genauigkeit (siehe Abbildung 3.18).

3.6.5 Multinomial Naïve Bayes

Zusätzlich zu den oben genannten Klassifizierungsmethoden ist **Multinomial Naïve Bayes (MNB)** eine beliebte Methode zur Dokumentenklassifizierung, da sie rechnerisch effizient ist und eine relativ gute Vorhersageleistung bietet. Im MNB-Klassifikator wird

jedes Dokument als eine Sammlung von Wörtern betrachtet, und die Reihenfolge der Wörter wird als irrelevant angesehen.

3.6.5.1 Klassifizierungsalgorithmus

Der multinomiale Naive Bayes-Klassifikator arbeitet mit dem Konzept der Begriffshäufigkeit, d. h. wie oft das Wort in einem Dokument vorkommt. In Hinblick auf die Tatsache, dass ein Begriff ausschlaggebend für die Stimmung eines Dokuments sein kann, macht diese Eigenschaft dieses Modells zu einer guten Wahl für die Klassifizierung von Dokumenten. Die Begriffshäufigkeit ist auch hilfreich bei der Entscheidung, ob der Begriff für die Analyse nützlich ist oder nicht. Manchmal kann ein Begriff in einem Dokument sehr oft vorkommen, was seine Begriffshäufigkeit in diesem Modell erhöht. Gleichzeitig kann es sich aber auch um ein Stoppwort handeln, das dem Dokument möglicherweise keine Bedeutung hinzufügt, aber eine hohe Begriffshäufigkeit aufweist, so dass solche Wörter zunächst entfernt werden müssen, um eine bessere Genauigkeit dieses Algorithmus zu erzielen.
In dem MNB-Klassifikator in dieser Arbeit wird eine Kompetenz k mit dem zugehörigen Studiengang s zunächst folgendermaßen berechnet:

$$P(s \mid k) \propto P(s) \prod_{1 \leq m \leq n_d} P(t_m \mid s)$$

wobei $P(t_m \mid s)$ die bedingte Wahrscheinlichkeit darstellt, dass der Begriff t_m in einer Kompetenz des Studiengangs s vorkommt, und wie folgt berechnet wird:

$$P(t_m \mid s) = \frac{count(t_m \mid s) + 1}{count(t_s) + |V|}$$

Dabei bezeichnet $count(t_m \mid s)$ die Anzahl der Vorkommen des Begriffs t_m in den Kompetenzen mit dem Studiengang s und $count(t_s)$ die Gesamtzahl der Token in den Kompetenzen mit dem Studiengang s.
Zusätzlich werden 1 und $|V|$ als Glättungskonstanten hinzugefügt, um Fehler bei der Berechnung zu vermeiden, wenn der Begriff in der Kompetenz überhaupt nicht vorkommt oder die Kompetenz leer oder Null ist. Dieses Konzept ist besser bekannt als Laplace-Glättung. $|V|$ ist die Anzahl der Begriffe im gesamten Wortschatz der Kompetenz.
$P(s)$ stellt die Vorwahrscheinlichkeit dar, dass eine Kompetenz mit dem Studiengang s,

die wie unten angegeben berechnet wird:

$$P(s) = \frac{AnzahlderKompetenzendesStudiengangss}{GesamtzahlderKompetenzen}$$

n_d: steht für die Anzahl der Token in einer Kompetenz
t_m: steht für das m^te Token in der Kompetenz
Die Wahrscheinlichkeit $P(s \mid k)$ wird für alle Studiengänge berechnet, und das Maximum gilt als vorausgesagter Studiengang für eine Kompetenz.

3.6.5.2 Einstellung der Hyperparameter

Im Falle des MNBs werden keinen Hyperparameter eingestellt.

3.6.5.3 Konfusionsmatrix

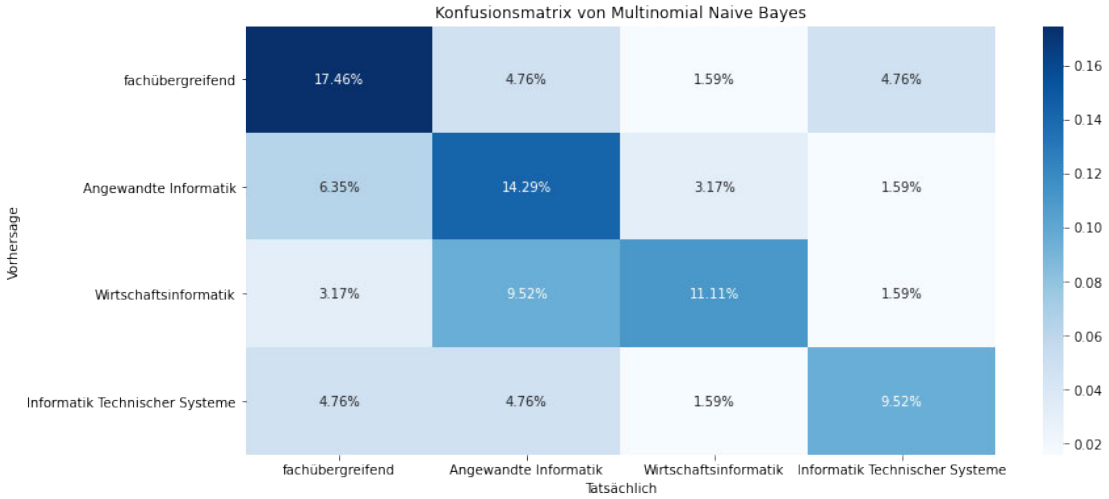


Abbildung 3.19: Konfusionsmatrix von Multinomial Naïve Bayes

Auffällig in der Abbildung 3.19 ist, dass der MNB-Klassifikator mit Ausnahme der **Angewandte Informatik** bei den übrigen Studiengängen eine höhere Anzahl von Kompetenzen hat als der KNN (Unterunterabschnitt 3.6.4.3). Die Unterschiede liegen bei etwa 2 bis 3%, was als erheblich bezeichnet werden kann.

3.6.6 Multinomial Logistic Regression

Multinomial Logistic Regression (MLR), die aufgrund der verwendeten Hypothesenfunktion auch als Softmax-Regression bezeichnet wird, ist ein überwachter Lernalgorithmus, der für verschiedene Probleme, einschließlich der Textklassifizierung, verwendet werden kann [41]. Es handelt sich um ein Regressionsmodell, das die logistische Regression auf Klassifizierungsprobleme verallgemeinert, bei denen die Ausgabe mehr als zwei mögliche Werte annehmen kann.

3.6.6.1 Klassifizierungsalgorithmus

Die logistische Regression ist ein mathematisches Modell [37] mit einer logistischen Funktion, die verwendet wird, um die Beziehung zwischen unabhängigen Zufallsvariablen und einer qualitativen abhängigen Variable mit zwei Werten 0/1 anzugeben. Die logistische Funktion wird als die Funktion $f : R \to R$ bezeichnet, die durch die folgende Gleichung:

$$f(X) = \frac{1}{1 + e^{-X}} = \frac{e^X}{1 + e^X}$$

Ein binärer Klassifikator, der auf einem logistischen Regressionsmodell basiert, lernt die Zuordnung eines Merkmalsvektors x zu einer Kategoriekennzeichnung y_k für die $k - te$ Kategoriekennzeichnung durch direkte Modellierung der bedingten Wahrscheinlichkeit $P(y_k \mid x)$ Die bedingte Wahrscheinlichkeit wird als

$$P(Y = 1 \mid X) = \frac{e^X}{1 + e^X} = \frac{e^{(\alpha + \sum_j \beta_j x_j)}}{1 + e^{(\alpha + \sum_j \beta_j x_j)}}$$

modelliert, wobei X als eine lineare Kombination von x_j.
Für einen Datenpunkt muss eine statistische Entscheidung darüber getroffen werden, ob diese Kompetenz zu einem bestimmten Studiengang gehört oder nicht. Die logistische Regression lässt sich leicht auf mehrere Klassen verallgemeinern. Die MLR ist eine einfache Erweiterung der binären logistischen Regression, die mehr als zwei Kategorien für die abhängige oder Ergebnisvariable zulässt. Für eine abhängige Variable mit K Kategorien erfordert dies die Berechnung von $K - 1$ Gleichungen, eine für jede Kategorie im Verhältnis zur Referenzkategorie, um die Beziehung zwischen der abhängigen Variable und den unabhängigen Variablen zu beschreiben. Für die Vorhersage der Kompetenz eines

Studiengangs durch eine Reihe von bekannten Kursen wird das multinomiale logistische Regressionsmodell wie folgt dargestellt:

1. Der Merkmalsvektor eines Datenpunkts wird bezeichnet als $x = [x_1, \ldots, x_j, \ldots, x_d]^T$.

2. Es wird davon ausgegangen, dass eine Kompetenz einem Studiengang gehört $k \in \{1, 2, 3, \ldots, K\}$ durch einen Vektor $y = [y_1, \ldots, y_K]^T$ wobei $y_k = 1$ und alle anderen Koordinaten 0 sind.

3. Die MLR ist ein bedingtes Wahrscheinlichkeitsmodell, definiert durch die Softmax-Funktion:
$$P(y_k = 1 \mid x, B) = \frac{\exp \beta_k^T x}{\sum_{j=1}^{K} \exp \beta_j^T x}$$

4. $B = [\beta_1, \ldots, \beta_K]^T$ ist eine Parametermodellmatrix, bei der jede Spalte von B ein Parametervektor ist, der einer der Klassen entspricht: $\beta = [\beta_{k1}, \ldots, \beta_{kd}]^T$. Die Klassifizierung einer neuen Beobachtung basiert auf dem Vektor der bedingten Wahrscheinlichkeitsschätzungen des Modells.

5. In dieser Arbeit wird nur die Klasse mit der höchsten bedingten Wahrscheinlichkeitsschätzung zugeordnet:
$$\hat{y}(x) = \arg\max_k P(y_k = 1 \mid x).$$

6. Betrachtet man einen Satz von Trainingsbeispielen $D = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\}$. Die Maximum-Likelihood-Schätzung der Parameter B ist gleichbedeutend mit der Minimierung der negierten Log-Likelihood:
$$l(B \mid D) = -\sum_i \left[\sum_k y_{ik} \beta_k^T x_i - ln \sum_k \exp \beta_k^T x_i \right]$$

3.6.6.2 Einstellung der Hyperparameter

Die folgenden Hyperparameter werden abgestimmt:

1. **C: float, default=1.0**: Kehrwert der Regularisierungsstärke; muss eine positive Fließkommazahl sein. Wie bei Support-Vektor-Maschinen geben kleinere Werte eine stärkere Regularisierung an.

2. **multi_class: {“auto”, “ovr”, “multinomial”}, default=“auto”**: Wenn die Option “ovr” gewählt wird, wird ein binäres Problem für jedes Label angepasst. Bei “multinomial” ist der minimierte Verlust der multinomiale Verlust, der über die gesamte Wahrscheinlichkeitsverteilung angepasst wird, auch wenn die Daten binär sind. “multinomial” ist nicht verfügbar, wenn solver=“liblinear”. “auto” wählt “ovr”, wenn die Daten binär sind, oder wenn solver=“liblinear”, und wählt ansonsten “multinomial”.

3. **solver: {“newton-cg”, “lbfgs”, “liblinear”, “sag”, “saga”}, default=“lbfgs”**: Für das Optimierungsproblem zu verwendender Algorithmus. Voreinstellung ist “lbfgs”. Bei der Auswahl eines Solvers sollten Sie die folgenden Aspekte berücksichtigen:

   - Für kleine Datensätze ist “liblinear” eine gute Wahl, während “sag” und “saga” bei großen Datensätzen schneller sind;
   - Bei Mehrklassenproblemen beherrschen nur “newton-cg”, “sag”, “saga” und “lbfgs” multinomiale Verluste;
   - “liblinear” ist auf Eins-gegen-Rest-Schemata beschränkt.

4. **class_weight: dict or “balanced”, default=None**: Mit den Klassen verbundene Gewichte in der Form {class_label: weight}. Wenn nichts angegeben wird, wird angenommen, dass alle Klassen das Gewicht eins haben. Der Modus “balanced” verwendet die Werte von y, um die Gewichte automatisch umgekehrt proportional zu den Klassenhäufigkeiten in den Eingabedaten als $n\_samples/(n\_classes * np.bincount(y))$ anzupassen. Es ist zu beachten, dass diese Gewichte mit $sample\_weight$ (das durch die Fit-Methode übergeben wird) multipliziert werden, wenn $sample\_weight$ angegeben ist.

5. **penalty: {“l1”, “l2”, “elasticnet”, “none”**, default=“l2”}: Festlegung der Norm der Strafe:

   - “none”: wird keine Strafe hinzugefügt;
   - “l2”: einen Begriff der L2-Strafe hinzufügen, der dann die Standardwahl ist;
   - “l1”: einen Begriff der L1-Strafe hinzufügen
   - “elasticnet”: werden sowohl L1- als auch L2-Strafbegriffe hinzugefügt.

Nach der Kombination zwischen **Randomized Search Cross Validation** und **Grid Search Cross Validation** werden den Parametern die Werte in Tabelle 3.7 zugewiesen:

Parameter	Wert
C	0.6666666666666667
class_weight	None
multi_class	multinomial
penalty	l2
solver	saga

Tabelle 3.7: Die besten Hyperparameter für MLR

3.6.6.3 Konfusionsmatrix

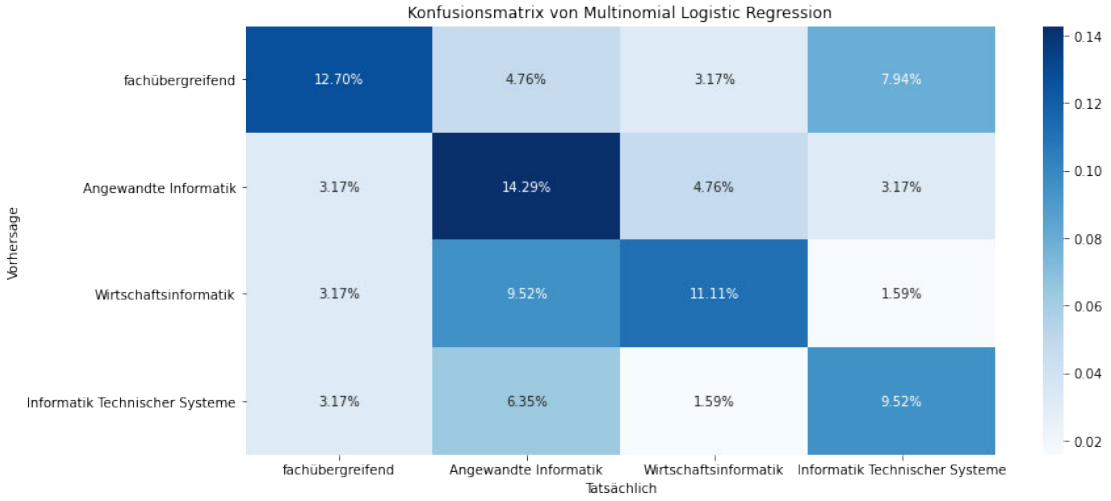


Abbildung 3.20: Konfusionsmatrix von Multinomial Logistic Regression

Es fällt auf, dass MLR der einzige Klassifizierungsalgorithmus ist, bei dem die höchste Anzahl von Kompetenzen nicht von der **fachübergreifend**, sondern von der **Angewandte Informatik** richtig erkannt wird. Allerdings gibt es eine klare Tendenz, dass die Kompetenzen durch das **Informatik Technischer System** nur unzureichend abgedeckt sind (siehe Abbildung 3.20). Der Grund dafür wird im Kapitel 4 besprochen.

3.6.7 Gradient Boosting Machine

Nicht zuletzt wird ein Modell betrachtet, das die gleiche Ensemble-Technik wie **Random Forest** verwendet, nämlich **Gradient Boosting Machine (GBM)**. Bei GBM, passt das Lernverfahren nacheinander neue Modelle an, um eine genauere Schätzung der Antwortvariablen zu erhalten. Die Grundidee dieses Algorithmus besteht darin, die neuen Basis-Lernmodelle so zu konstruieren, dass sie maximal mit dem negativen Gradienten der Verlustfunktion korreliert sind, die dem gesamten Ensemble zugeordnet ist. Die angewandten Verlustfunktionen können beliebig sein, aber zur besseren Veranschaulichung sei gesagt, dass das Lernverfahren zu einer konsekutiven Fehleranpassung führt, wenn die Fehlerfunktion der klassische quadratische Fehlerverlust ist. Im Allgemeinen ist die Wahl der Verlustfunktion dem Forscher überlassen, wobei es sowohl eine Vielzahl von bisher abgeleiteten Verlustfunktionen gibt als auch die Möglichkeit, eine den eigenen aufgabenspezifischen Verlust. Diese hohe Flexibilität macht die GBMs in hohem Maße anpassbar an jede bestimmte data-driven Aufgabe [31].

3.6.7.1 Klassifizierungsalgorithmus

Wie andere Boosting-Methoden kombiniert auch Gradient Boosting [22] iterativ schwache "Lerner" zu einem einzigen starken Lerner. Am einfachsten lässt es sich im Rahmen der Kleinstquadratregression erklären, bei der das Ziel darin besteht, einem Modell F beizubringen, Werte der Form $\hat{y} = F(x)$ durch Minimierung des mittleren quadratischen Fehlers $\frac{1}{n}\sum_i (\hat{y}_i - y_i)^2$ vorherzusagen, wobei i über eine Trainingsmenge der Größe n tatsächlicher Werte der Ausgangsvariablen y indiziert ist:

- $\hat{y}_i$ = der vorhergesagte Wert $F(x_i)$
- y_i = der beobachtete Wert
- n = die Anzahl der Stichproben in y

Es wird nun einen Gradient-Boosting-Algorithmus mit M Stufen betrachtet. In jeder Stufe $m\,(1 \leq m \leq M)$ des Gradient Boosting, wird angenommen, ein unvollkommenes Modell F_m, dieses Modell kann einfach zurückgeben $\hat{y}_i = \overline{y}$, wobei die rechte Seite ist der Mittelwert von y. Um F_m zu verbessern, sollte unser Algorithmus einen neuen Schätzer $h_m\,(x)$ hinzufügen. Also,

$$F_{m+1}(x) = F_m(x) + h_m(x) = y$$

oder äquivalent,

$$h_m(x) = y - F_m(x).$$

Daher wird Gradient Boosting h an den Restwert $y - F_m(x)$ anpassen. Wie bei anderen Boosting-Varianten versucht jeder $F_{m+1}(x)$, die Fehler seines Vorgängers $F_m(x)$ zu korrigieren. Eine Verallgemeinerung dieser Idee auf andere Verlustfunktionen als den quadratischen Fehler und auf Klassifizierungs- und Rangordnungsprobleme ergibt sich aus der Beobachtung, dass die Residuen $h_m(x)$ für ein bestimmtes Modell proportional zu den negativen Gradienten der Verlustfunktion des mittleren quadratischen Fehlers (MSE) gleichwertig sind:

$$L_{MSE} = \frac{1}{n}(y - F(x))^2$$

$$-\frac{\partial L_{MSE}}{\partial F} = \frac{2}{n}(y - F(x)) = \frac{2}{n}h_m(x).$$

Gradient Boosting könnte also auf einen Gradientenabstiegsalgorithmus spezialisiert werden, und seine Verallgemeinerung bedeutet, dass man einen anderen Verlust und dessen Gradienten "einfügt".
Bei vielen Problemen des überwachten Lernens gibt es eine Ausgangsvariable y und einen Vektor von Eingangsvariablen x, die mit einer probabilistischen Verteilung zueinander in Beziehung stehen. Das Ziel ist es, eine Funktion $\hat{F}(x)$ zu finden, die die Ausgangsvariable am besten aus den Werten der Eingangsvariablen approximiert. Dies wird formalisiert durch die Einführung einer Verlustfunktion $L(y, F(x))$ und deren Minimierung:

$$\hat{F} = \arg\min_F \mathbb{E}_{x,y}[L(y, F(x))].$$

Die Gradient-Boosting-Methode geht von einem reellwertigen y aus und sucht eine Approximation $\hat{F}(x)$ in Form einer gewichteten Summe von Funktionen $h_i(x)$ aus einer Klasse $\mathcal{H}$, die als Basis- (oder schwache) Lerner bezeichnet werden:

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const.}$$

In der Regel wird eine Trainingsmenge $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ von bekannten Stichprobenwerten von x und entsprechenden Werten von y vorgegeben. In Übereinstimmung mit dem Prinzip der empirischen Risikominimierung versucht das Verfahren, eine Approximation $\hat{F}(x)$ zu finden, die den Durchschnittswert der Verlustfunktion auf der Trainingsmenge minimiert, d.h., das empirische Risiko minimiert. Dabei wird mit einem Modell begonnen, das aus einer konstanten Funktion $F_0(x)$ besteht, und dieses schrittweise auf gierige Weise erweitert:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma),$$

$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)) \right],$$

wobei $h_m \in \mathcal{H}$ eine Basis-Lernfunktion ist. Leider ist die Wahl der besten Funktion h in jedem Schritt für eine beliebige Verlustfunktion L im Allgemeinen ein rechnerisch unlösbares Optimierungsproblem. Daher wird der Ansatz auf eine vereinfachte Version des Problems beschränkt. Die Idee ist, einen steilsten Abstiegsschritt auf dieses Minimierungsproblem anzuwenden (funktionaler Gradientenabstieg). Die Grundidee des steilsten Abstiegs besteht darin, ein lokales Minimum der Verlustfunktion durch Iteration auf der $F_m(x)$ zu finden. Tatsächlich ist die Richtung des lokalen maximalen Abstiegs der Verlustfunktion der negative Gradient. Daher muss ein kleiner Betrag γ so verschoben werden, dass die lineare Approximation gültig bleibt:

$$F_m(x) = F_{m-1}(x) - \gamma \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))$$

wobei $\gamma > 0$. Dies impliziert (für kleine $\gamma : L(y_i, F_m(x_i)) \leq L(y_i, F_{m-1}(x_i))$.
Außerdem lässt sich γ optimieren, indem man den $\gamma - Wert$ findet, für den die Verlustfunktion ein Minimum aufweist:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_m)\right) = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))\right),$$

Wenn der kontinuierliche Fall betrachtet wird, d. h. wenn $\mathcal{H}$ die Menge der beliebig differenzierbaren Funktionen auf $\mathbb{R}$ ist, würde das Modell gemäß den folgenden Gleichungen aktualisiert:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))$$

wobei:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))\right),$$

wobei die Ableitungen in Bezug auf die Funktionen F_i für $i \in \{1, ..., m\}$ genommen werden und γ_m die Schrittlänge ist. Im diskreten Fall jedoch, d. h. wenn die Menge $\mathcal{H}$ endlich ist, wählen wir die Kandidatenfunktion h, die der Steigung von L am nächsten kommt, für die der Koeffizient γ dann mit Hilfe der Liniensuche über die obigen Gleichungen berechnet werden kann. Es ist zu beachten, dass dieser Ansatz eine Heuristik ist und daher keine exakte Lösung für das gegebene Problem liefert, sondern eher eine Annäherung. Die Abbildung 3.21 zeigt den Pseudocode des allgemeinen Gradient-Boosting-Verfahrens:

Algorithm 1 Friedman's Gradient Boost algorithm

Inputs:

- input data $(x, y)_{i=1}^{N}$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

1: initialize $\widehat{f}_0$ with a constant
2: **for** $t = 1$ to M **do**
3: compute the negative gradient $g_t(x)$
4: fit a new base-learner function $h(x, \theta_t)$
5: find the best gradient descent step-size ρ_t:
$\rho_t = \arg\min_\rho \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right]$
6: update the function estimate:
$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \rho_t h(x, \theta_t)$
7: **end for**

Abbildung 3.21: Friedman's Gradient Boost Algorithmus [31]

Im Rahmen dieser Arbeit wird das **Gradient Tree Boosting** angewendet, das normalerweise mit Entscheidungsbäumen (insbesondere CART-Bäumen) einer festen Größe als Basislerner verwendet wird. Generisches Gradient Boosting im m-ten Schritt würde einen Entscheidungsbaum $h_m(x)$ auf Pseudo-Residuen anpassen. Sei J_m die Anzahl seiner Blätter. Der Baum unterteilt den Eingaberaum in J_m disjunkte Regionen $R_{1m}, \ldots, R_{J_m m}$ und sagt in jeder Region einen konstanten Wert voraus. Unter Verwendung der Indikatorschreibweise kann die Ausgabe von $h_m(x)$ für die Eingabe x als Summe geschrieben werden:

$$h_m(x) = \sum_{j=1}^{J_m} b_{jm} \mathbf{1}_{R_{jm}}(x),$$

wobei b_{jm} der in der Region R_{jm} vorhergesagte Wert ist. Dann werden die Koeffizienten b_{jm} mit einem Wert γ_m multipliziert, der mittels Zeilensuche so gewählt wird, dass die Verlustfunktion minimiert wird, und das Modell wird wie folgt aktualisiert:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x), \quad \gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

Friedman schlägt vor, diesen Algorithmus so zu modifizieren, dass er einen separaten optimalen Wert γ_{jm} für jede der Regionen des Baumes wählt, anstatt eines einzigen γ_m für den gesamten Baum. Er nennt den modifizierten Algorithmus "TreeBoost" [10] [11]. Die Koeffizienten b_{jm} aus dem Baumanpassungsverfahren können dann einfach verworfen werden, und die Modellaktualisierungsregel wird:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(x), \quad \gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma).$$

3.6.7.2 Einstellung der Hyperparameter

Wie im obigen Abschnitt erwähnt, verwendet GBM Entscheidungsbäume zur Klassifizierung, daher sind einige Parameter analog zum Entscheidungsbaum (Unterunterabschnitt 3.6.1.2). Diese Parameter werden in diesem Abschnitt nicht erwähnt. Darüber hinaus werden die folgenden Hyperparameter angepasst:

1. **learning_rate: float, default=0.1**: Mit der *learning_rate* wird der Beitrag jedes Baumes um die Lernrate verringert. Es besteht ein Kompromiss zwischen *learning_rate* und *n_estimators*. Dadurch wird die Auswirkung der einzelnen Bäume auf das Endergebnis bestimmt. GBM geht von einer anfänglichen Schätzung aus, die anhand der Ergebnisse der einzelnen Bäume aktualisiert wird. Der Lernparameter steuert das Ausmaß dieser Änderung der Schätzungen. Niedrigere Werte werden im Allgemeinen bevorzugt, da sie das Modell robust gegenüber den spezifischen Merkmalen des Baumes machen und es somit gut verallgemeinern können. Niedrigere Werte würden eine höhere Anzahl von Bäumen erfordern, um alle Beziehungen zu modellieren, und sind daher rechenintensiv.

2. **subsample: float, default=1.0**: Der Anteil der Stichproben, der für die Anpassung der einzelnen Basis-Learner verwendet wird. Ist er kleiner als 1.0, führt dies zu Stochastic Gradient Boosting. *subsample* steht in Wechselwirkung mit dem Parameter *n_estimators*. Werte, die etwas unter 1 liegen, machen das Modell robust, indem sie die Varianz verringern. Typische Werte von 0.8 funktionieren im Allgemeinen gut, können aber noch feiner abgestimmt werden.

Nach der Kombination zwischen **Randomized Search Cross Validation** und **Grid Search Cross Validation** werden den Parametern die Werte in Tabelle 3.8 zugewiesen:

Parameter	Wert
learning_rate	0.5
max_depth	10
max_features	sqrt
min_samples_leaf	1
min_samples_split	90
n_estimators	200
subsample	1.0

Tabelle 3.8: Die besten Hyperparameter für GBM

3.6.7.3 Konfusionsmatrix

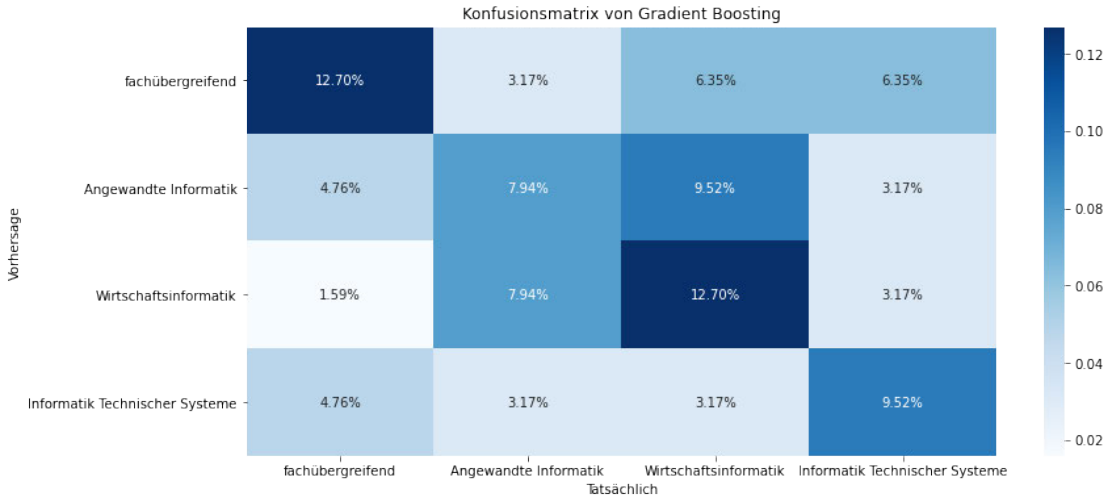


Abbildung 3.22: Konfusionsmatrix von Gradient Boosting Machine

Der höchst interessante Punkt in der Abbildung 3.22 ist, dass das GBM die geringste Genauigkeit bei der Identifizierung der Kompetenzen im Studiengang **Angewandte Informatik** hat. Die Anteile der korrekten Identifizierung der Kompetenzen aus anderen Studiengängen sind bei den genannten Modellen recht ähnlich.

4 Auswertung

Im folgenden Kapitel werden die experimentellen Ergebnisse dieser Arbeit bewertet und beurteilt. Zu diesem Zweck werden zunächst die verschiedenen Möglichkeiten zur Bewertung der Leistung eines maschinellen Lernmodells vorgestellt. Dann wird das beste Modell auf der Grundlage der gemessenen Leistung aller Modelle während der Implementierung ausgewählt.

4.1 Leistungsbewertung

Der Einfachheit halber spricht man in der Regel von einem binären Klassifikationsproblem, bei dem es darum geht, ob ein Patient Krebs hat (positiv) oder gesund ist (negativ). Zunächst müssen einige allgemeine Begriffe definiert werden:

- **True positives (TP)**: Positiv vorhergesagt und tatsächlich positiv.
- **False positives (FP)**: Positiv vorhergesagt und tatsächlich negativ.
- **True negatives (TN)**: Negativ vorhergesagt und tatsächlich negativ.
- **False negatives (FN)**: Negativ vorhergesagt und tatsächlich positiv.

Nach der Abstimmung der Hyperparameter mit den Trainingsdaten durch Kreuzvalidierung und der Anpassung des Modells an diese Trainingsdaten gilt es, die Leistung des Modells bei völlig ungesehenen Daten (dem Testsatz) zu bewerten. Bei Klassifizierungsproblemen gibt es mehrere Metriken, die verwendet werden können, um Erkenntnisse über die Leistung des Modells zu gewinnen. Einige davon sind:

- *Accuracy*: Die Genauigkeitsmetrik misst das Verhältnis der richtigen Vorhersagen zur Gesamtzahl der ausgewerteten Instanzen.

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

- *Precision*: Die *Precision* misst den Anteil der korrekt vorhergesagten positiven Muster an der Gesamtzahl der vorhergesagten Muster in einer positiven Klasse.

$$Precision = \frac{TP}{TP + FP}$$

- *Recall*: Der Erinnerungswert wird verwendet, um den Anteil der positiven Muster zu messen, die richtig klassifiziert werden.

$$Recall = \frac{TP}{TP + FN}$$

- *Specificity*: Prozentualer Anteil der negativen Instanzen an der Gesamtzahl der negativen Instanzen. Daher ist der Nenner (TN + FP) hier die tatsächliche Anzahl der negativen Instanzen im Datensatz. Es ist ähnlich wie der *Recall*, aber die Verschiebung liegt auf den negativen Instanzen.

$$Specificity = \frac{TN}{TN + FP}$$

- *F1-score*: Er ist das harmonische Mittel aus *Precision* und *Recall*. Dabei wird der Beitrag beider Werte berücksichtigt. Je höher der F1-Wert, desto besser. Es ist zu beachten, dass aufgrund des Produkts im Zähler die endgültige F1-Punktzahl deutlich sinkt, wenn einer der beiden Werte zu niedrig ist. Ein Modell schneidet also beim F1-Score gut ab, wenn die vorhergesagten positiven Ergebnisse tatsächlich positiv sind (Präzision) und wenn es keine positiven Ergebnisse verpasst und sie negativ vorhersagt (Rückruf).

$$F - measure = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 * precision * recall}{precision + call}$$

  Ein Nachteil ist, dass sowohl Präzision als auch Recall die gleiche Bedeutung beigemessen wird, weshalb je nach Anwendung ein Wert höher sein kann als der andere und der F1-Score möglicherweise nicht die richtige Metrik dafür ist. Daher kann entweder der gewichtete F1-Score oder die PR- oder ROC-Kurve hilfreich sein.

- *ROC curve*: ROC steht für Receiver Operating Characteristic, und das Diagramm wird gegen TPR und FPR für verschiedene Schwellenwerte aufgetragen. Mit zunehmender TPR nimmt auch die FPR zu. Wie aus dem linken Graph in Abbildung 4.1

ersichtlich ist, gibt es vier Kategorien, und der Schwellenwert, der näher an die obere linke Ecke heranreicht, ist der richtige. Der Vergleich verschiedener Prädiktoren auf einem gegebenen Datensatz wird ebenfalls einfach (siehe rechten Graph in Abbildung 4.1), man kann den Schwellenwert entsprechend der jeweiligen Anwendung wählen. ROC AUC ist einfach die Fläche unter der Kurve, je höher ihr numerischer Wert, desto besser.

$$TruePositiveRate(TPR) = Recall = \frac{TP}{TP + FN}$$

$$FalsePositiveRate(FPR) = 1 - Specificity = \frac{FP}{TN + FP}$$

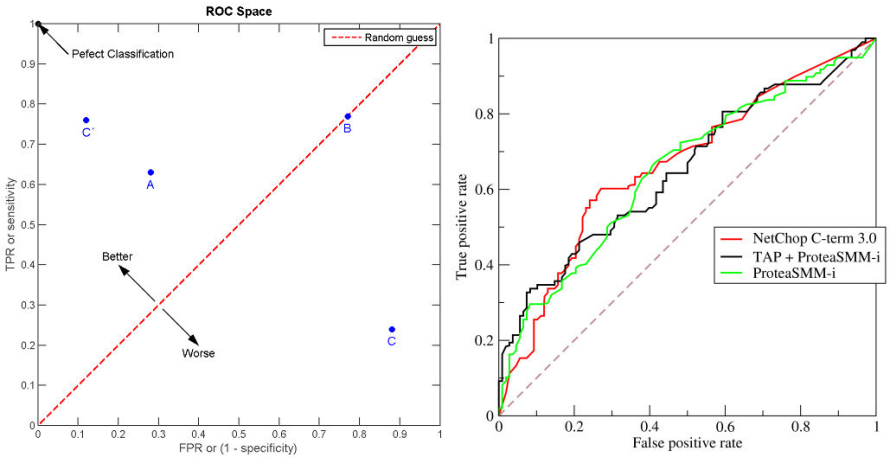


Abbildung 4.1: ROC-Kurve. URL

Diese Metriken sind stark erweitert und werden in der binären Klassifikation häufig verwendet. Bei der Mehrklassen-Klassifikation werden sie jedoch komplexer zu berechnen und weniger interpretierbar. Darüber hinaus wollen im Rahmen dieser speziellen Anwendung nur die Dokumente korrekt vorhergesagt werden. Die Kosten für falsch-positive oder falsch-negative Ergebnisse sind für die Analyse gleich hoch. Aus diesem Grund spielt es für einen Klassifikator keine Rolle, ob er spezifischer oder empfindlicher ist, solange er so viele Dokumente wie möglich richtig klassifiziert. Daher wurde die Genauigkeit (*accuracy*) beim Vergleich der Modelle und bei der Auswahl der besten Hyperparameter untersucht. Im ersten Fall wurde die Genauigkeit sowohl für die Trainings- als auch für die Testmenge berechnet, um überangepasste Modelle zu erkennen. Allerdings haben für

jedes Modell auch die Konfusionsmatrix und der Klassifizierungsbericht (der die Präzision, den Recall und den F1-Score für alle Klassen berechnet) zur Verfügung gestanden, so dass deren Verhalten weiter interpretiert werden konnte.

4.2 Auswahl des besten Modells

In Tabelle 4.1 werden die verschiedenen Modelle und ihre Bewertungsmaßstäbe zusammengefasst:

Modell	**Trainingssatz Genauigkeit**	**Testsatz Genauigkeit**
Decision Tree	0.582192	0.238095
Gradient Boosting	0.986301	0.428571
K Nearest Neighbors	0.582192	0.460317
Logistic Regression	0.924658	0.476190
Multinomial Naive Bayes	0.890411	0.523810
Random Forest	0.986301	0.428571
Support Vector Machine	0.986301	0.444444

Tabelle 4.1: Zusammenfassung der Evaluierung aller Modelle mit dem ursprünglichen Datensatz

Insgesamt ergibt sich eine sehr geringe Genauigkeit für jedes Modell. Es ist zu beobachten, dass die Modelle "Gradient Boosting", "Support Vector Machine" und "Random Forest" anscheinend *overfitted* sind, da sie eine extrem hohe Genauigkeit in der Trainingsmenge, aber eine geringere Genauigkeit in der Testmenge aufweisen. Das einzige Modell, das eine kleine Lücke zwischen den Genauigkeiten im Trainings- und Testsatz aufweist, ist "K Neareast Neighbor". Allerdings kann es nur etwa 46% des Testsatzes richtig klassifizieren. Der Grund dafür liegt wahrscheinlich in der Menge des ursprünglichen Datensatzes. Die Anzahl der Datenpunkte ist nicht ausreichend, um zuverlässige Modelle zu entwickeln. Daher wurde die Datenmenge verfünffacht, um eine große Datenmenge (mehr als 1000 Datenpunkte) zu erhalten. Die Tabelle 4.2 zeigt die Ergebnisse aller Modelle nach dem Training mit dem neuen Datensatz.

Modell	Trainingssatz Genauigkeit	Testsatz Genauigkeit
Decision Tree	0.980848	0.964968
Gradient Boosting	0.980848	0.964968
K Nearest Neighbors	0.741450	0.646497
Logistic Regression	0.965800	0.904459
Multinomial Naive Bayes	0.879617	0.786624
Random Forest	0.980848	0.964968
Support Vector Machine	0.980848	0.964968

Tabelle 4.2: Zusammenfassung der Evaluierung aller Modelle mit dem vervielfältigten Datensatz

Es lässt sich ein deutlicher Trend erkennen: Je mehr Daten für das Training zur Verfügung stehen, desto genauer können die Modelle Vorhersagen treffen. In diesem Fall ist "K Nearest Neighbor" immer noch das schlechteste Modell, obwohl seine Genauigkeit um etwa 18% steigt. Es ist interessant, dass "Decision Tre", "Gradient Boosting", "Random Forest" und "Support Vector Machine" die höchsten, aber gleichen Genauigkeiten sowohl in der Trainings- als auch in der Testmenge aufweisen. Um das beste Modell auszuwählen, muss ein weiterer Faktor evaluiert werden: die Trainingszeit.

Modell	Trainingszeit in ms
Decision Tree	1.97
Support Vector Machine	103.00
Random Forest	739.00
Gradient Boosting	2930.00

Tabelle 4.3: Trainingszeit aufsteigend sortieren

In der Tabelle 4.3 sind die Modelle in aufsteigender Reihenfolge nach ihrer Trainingszeit geordnet. Der Decision-Tree-Klassifikator wird gewählt, weil er die höchste Genauigkeit in der Testmenge aufweist, die der Genauigkeit in der Trainingsmenge sehr ähnlich ist, und seine Trainingszeit extrem gering ist.
Daraus lässt sich schließen, dass es im Vergleich zu anderen Modellen Vorteile bringt:

- **Umfassend**: Ein wesentlicher Vorteil eines DTs besteht darin, dass er die Berücksichtigung aller möglichen Ergebnisse einer Entscheidung erzwingt und jeden Weg zu einer Schlussfolgerung nachzeichnet. Er erstellt eine umfassende Analyse

der Konsequenzen entlang jedes Zweiges und identifiziert Entscheidungsknoten, die einer weiteren Analyse bedürfen.

- **Spezifisch**: DTs weisen jedem Problem, jedem Entscheidungspfad und jedem Ergebnis bestimmte Werte zu. Die Verwendung monetärer Werte macht Kosten und Nutzen explizit. Dieser Ansatz identifiziert die relevanten Entscheidungspfade, reduziert die Unsicherheit, klärt Unklarheiten und verdeutlicht die finanziellen Konsequenzen verschiedener Handlungsoptionen. Wenn keine faktischen Informationen zur Verfügung stehen, werden in DTs Wahrscheinlichkeiten für die Bedingungen verwendet, um die verschiedenen Möglichkeiten miteinander zu vergleichen.
- **Einfacher Gebrauch**: DTs sind einfach zu verwenden und mit einfacher Mathematik zu erklären, ohne komplexe Formeln. Sie stellen alle Entscheidungsalternativen für schnelle Vergleiche in einem leicht verständlichen Format mit nur kurzen Erklärungen visuell dar. DTs können auch als intuitiv betrachtet werden und folgen denselben Denkmustern, die Menschen bei ihren Entscheidungen anwenden.

5 Fazit

Ziel dieser Arbeit ist es, ein Vorhersagemodell auf Basis der Kompetenzen der Modulhandbücher des Departments Informatik der HAW zu konstruieren. Darüber hinaus soll die Anwendbarkeit auf Vorhersagen für Informatikkurse durch maschinelles Lernen praktisch durchgeführt werden. Bei der Datenauswahl und Datenvorverarbeitung wurden detaillierte Schritte und viele verschiedene Methoden vorgestellt. In der Phase des maschinellen Lernens wurden verschiedene Algorithmen eingesetzt, um die Bewertungsvielfalt der Zielführung zu vergleichen. Durch die Einbeziehung von externen Daten und technischen Merkmalen konnte die Vorhersagequalität deutlich verbessert werden.
Die hier durchgeführten Experimente haben wertvolle Erkenntnisse geliefert und werden als Grundlage für die weitere Arbeit mit dem intelligenten System dienen, das in Zukunft Empfehlungen für Schüler und fundierte Entscheidungen für Lehrer und Tutoren liefern könnte. Um das endgültige Ziel zu erreichen, muss sich das System noch in vielerlei Hinsicht weiterentwickeln.
Diese Arbeit wurde auf einem lokalen Computer durchgeführt und unterliegt daher einigen technischen Beschränkungen. Ein Betrieb auf leistungsfähigeren Servern oder GPUs würde es ermöglichen, mehr Daten für das Training hinzuzufügen, und würde wahrscheinlich auch mehr Durchläufe ermöglichen, um genauere Ergebnisse zu erzielen. Anstatt den Dimensionsraum zu vergrößern, könnte eine Dimensionsreduktion sinnvoll sein. Hier könnte die Merkmalsauswahl eine Abhilfe schaffen, um den Merkmalsraum auf die wichtigsten Merkmale zu reduzieren und so dem Modell weniger komplexe Daten zu liefern.
Abschließend lässt sich sagen, dass Python eine wunderbare Programmiersprache ist, die eine Menge großartiger Bibliotheken für die Erstellung leistungsstarker maschineller Lernmodelle und die Verarbeitung natürlicher Sprache bietet. Für die Aufgabe, ein maschinelles Lernmodell für die Textklassifizierung mit einem NLP-Ansatz zu erstellen, werden die beliebtesten Bibliotheken für maschinelles Lernen verwendet, wie z. B.: Pandas, Scikit-learn, Numpy, NLTK und haben das Textklassifizierungsmodell mit einem NLP-Ansatz erstellt.

Literaturverzeichnis

[1] : *COLT '92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. New York, NY, USA : Association for Computing Machinery, 1992. – ISBN 089791497X

[2] Bartels, Sören: *Dünnbesetzte Matrizen und Vorkonditionierung*. S. 141–147. In: *Numerik 3x9: Drei Themengebiete in jeweils neun kurzen Kapiteln*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2016. – URL https://doi.org/10.1007/978-3-662-48203-2_17. – ISBN 978-3-662-48203-2

[3] Bartosch, Ulrich ; Grygar, Ann-Kathrin: Hochschulbildung mit Kompetenz. Eine Handreichung zum Qualifikationsrahmen für deutsche Hochschulabschlüsse (HQR). In: *HRK Hochschulrektorenkonferenz* (2019)

[4] Bininda-Emonds, Olaf R. P. ; Jones, Kate E. ; Price, Samantha A. ; Cardillo, Marcel ; Grenyer, Richard ; Purvis, Andy: *Garbage in, Garbage out*. S. 267–280. In: Bininda-Emonds, Olaf R. P. (Hrsg.): *Phylogenetic Supertrees: Combining information to reveal the Tree of Life*. Dordrecht : Springer Netherlands, 2004. – URL https://doi.org/10.1007/978-1-4020-2330-9_13. – ISBN 978-1-4020-2330-9

[5] Coroiu, Adriana M.: Tuning model parameters through a Genetic Algorithm approach. In: *2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2016, S. 135–140

[6] Cox, Victoria: *Exploratory Data Analysis*. S. 47–74. In: *Translating Statistics to Make Decisions : A Guide for the Non-Statistician*. Berkeley, CA : Apress, 2017. – URL https://doi.org/10.1007/978-1-4842-2256-0_3. – ISBN 978-1-4842-2256-0

[7] Dietterich, Tom: Overfitting and undercomputing in machine learning. In: *ACM computing surveys (CSUR)* 27 (1995), Nr. 3, S. 326–327

[8] DQR, Deutscher Qualifikationsrahmen für lebenslanges L.: Diskussionsvorschlag eines Deutschen Qualifikationsrahmens für lebenslanges Lernen/Erarbeitet vom „Arbeitskreis Deutscher Qualifikationsrahmen“. (2009). – URL https://www.koopson.de/uploads/media/DQR_Diskussionsvorschlag_.pdf

[9] Egger, Roman: Machine Learning in Tourism: A Brief Overview. In: *Applied Data Science in Tourism* (2022), S. 85–107

[10] Friedman, Jerome H.: Greedy Function Approximation: A Gradient Boosting Machine. In: *The Annals of Statistics* 29 (2001), Nr. 5, S. 1189–1232. – URL http://www.jstor.org/stable/2699986. – Zugriffsdatum: 2022-05-20. – ISSN 00905364

[11] Friedman, Jerome H.: Stochastic gradient boosting. In: *Computational Statistics & Data Analysis* 38 (2002), Nr. 4, S. 367–378. – URL https://www.sciencedirect.com/science/article/pii/S0167947301000652. – Nonlinear Methods and Data Mining. – ISSN 0167-9473

[12] Ghannay, Sahar ; Favre, Benoit ; Estève, Yannick ; Camelin, Nathalie: Word Embedding Evaluation and Combination. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia : European Language Resources Association (ELRA), Mai 2016, S. 300–305. – URL https://aclanthology.org/L16-1046

[13] Goldberg, Yoav: Neural network methods for natural language processing. In: *Synthesis lectures on human language technologies* 10 (2017), Nr. 1, S. 1–309

[14] Gupta, Heena ; Asha, V: Impact of encoding of high cardinality categorical data to solve prediction problems. In: *Journal of Computational and Theoretical Nanoscience* 17 (2020), Nr. 9-10, S. 4197–4201

[15] Hancock, John T. ; Khoshgoftaar, Taghi M.: Survey on categorical data for neural networks. In: *Journal of Big Data* 7 (2020), Nr. 1, S. 1–41

[16] Henderson, Matthew L. ; Al-Rfou, Rami ; Strope, Brian ; Sung, Yun-Hsuan ; Lukács, László ; Guo, Ruiqi ; Kumar, Sanjiv ; Miklos, Balint ; Kurzweil, Ray: Efficient Natural Language Response Suggestion for Smart Reply. In: *CoRR* abs/1705.00652 (2017). – URL http://arxiv.org/abs/1705.00652

[17] Jakkula, Vikramaditya: Tutorial on support vector machine (svm). In: *School of EECS, Washington State University* 37 (2006), Nr. 2.5, S. 3

[18] James, Gareth ; Witten, Daniela ; Hastie, Trevor ; Tibshirani, Robert: *An introduction to statistical learning*. Bd. 112. Springer, 2013

[19] Khurana, Udayan ; Samulowitz, Horst ; Turaga, Deepak: Feature Engineering for Predictive Modeling Using Reinforcement Learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (2018), Apr., Nr. 1. – URL https://ojs.aaai.org/index.php/AAAI/article/view/11678

[20] Langley, Pat: Human and machine learning. In: *Machine Learning* 1 (1986), Nr. 3, S. 243–248

[21] Li, Baoli ; Yu, Shiwen ; Lu, Qin: An Improved k-Nearest Neighbor Algorithm for Text Categorization. In: *CoRR* cs.CL/0306099 (2003). – URL http://arxiv.org/abs/cs/0306099

[22] Li, Cheng: A gentle introduction to gradient boosting. In: *URL: http://www.ccs. neu. edu/home/vip/teach/MLcourse/4_ boosting/slides/gradient_ boosting. pdf* (2016)

[23] Liaw, Andy ; Wiener, Matthew u. a.: Classification and regression by randomForest. In: *R news* 2 (2002), Nr. 3, S. 18–22

[24] Länder, Kultusministerkonferenz der: Qualifikationsrahmen für deutsche Hochschulabschlüsse. (2017). – URL https://www.kmk.org/fileadmin/Dateien/veroeffentlichungen_beschluesse/2017/2017_02_16-Qualifikationsrahmen.pdf

[25] Mantovani, Rafael G. ; Horváth, Tomáš ; Cerri, Ricardo ; Vanschoren, Joaquin ; Carvalho, André C.P.L.F. d.: Hyper-Parameter Tuning of a Decision Tree Induction Algorithm. In: *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, 2016, S. 37–42

[26] Mantovani, Rafael G. ; Horváth, Tomáš ; Cerri, Ricardo ; Junior, Sylvio B. ; Vanschoren, Joaquin ; Carvalho, André Carlos Ponce de Leon F. de: An empirical study on hyperparameter tuning of decision trees. In: *arXiv preprint arXiv:1812.02207* (2018)

[27] Mendling, Jan ; Decker, Gero ; Hull, Richard ; Reijers, Hajo A. ; Weber, Ingo: How do machine learning, robotic process automation, and blockchains affect the human factor in business process management? In: *Communications of the Association for Information Systems* 43 (2018), Nr. 1, S. 19

[28] Morgenthaler, Stephan: Exploratory Data Analysis. In: *WIREs Comput. Stat.* 1 (2009), jul, Nr. 1, S. 33–44. – URL https://doi.org/10.1002/wics.2. – ISSN 1939-5108

[29] Myles, Anthony J. ; Feudale, Robert N. ; Liu, Yang ; Woody, Nathaniel A. ; Brown, Steven D.: An introduction to decision tree modeling. In: *Journal of Chemometrics: A Journal of the Chemometrics Society* 18 (2004), Nr. 6, S. 275–285

[30] Nadkarni, Prakash M. ; Ohno-Machado, Lucila ; Chapman, Wendy W.: Natural language processing: an introduction. In: *Journal of the American Medical Informatics Association* 18 (2011), 09, Nr. 5, S. 544–551. – URL https://doi.org/10.1136/amiajnl-2011-000464. – ISSN 1067-5027

[31] Natekin, Alexey ; Knoll, Alois: Gradient boosting machines, a tutorial. In: *Frontiers in Neurorobotics* 7 (2013). – URL https://www.frontiersin.org/article/10.3389/fnbot.2013.00021. – ISSN 1662-5218

[32] Probst, Philipp ; Wright, Marvin N. ; Boulesteix, Anne-Laure: Hyperparameters and tuning strategies for random forest. In: *WIREs Data Mining and Knowledge Discovery* 9 (2019), Nr. 3, S. e1301. – URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1301

[33] Rácz, Anita ; Bajusz, Dávid ; Héberger, Károly: Effect of dataset size and train/test split ratios in QSAR/QSPR multiclass classification. In: *Molecules* 26 (2021), Nr. 4, S. 1111

[34] Refaeilzadeh, Payam ; Tang, Lei ; Liu, Huan: *Cross-Validation*. S. 1–7. In: Liu, Ling (Hrsg.) ; Özsu, M. T. (Hrsg.): *Encyclopedia of Database Systems*. New York, NY : Springer New York, 2016. – URL https://doi.org/10.1007/978-1-4899-7993-3_565-2. – ISBN 978-1-4899-7993-3

[35] Ren, Qiubing ; Li, Mingchao ; Han, Shuai: Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives. In: *Big Earth Data* 3 (2019), 02, S. 1–18

[36] Rodriguez, Pau ; Bautista, Miguel A. ; Gonzalez, Jordi ; Escalera, Sergio: Beyond one-hot encoding: Lower dimensional target embedding. In: *Image and Vision Computing* 75 (2018), Nr. 05

[37] Salillari, Denisa ; Prifti, Luela: A multinomial logistic regression model for text in Albanian language. In: *Journal of Advances in Mathematics* 12 (2016), Nr. 7, S. 6407–6411

[38] Scott, Sam ; Matwin, Stan: Feature engineering for text classification. In: *ICML* Bd. 99 Citeseer (Veranst.), 1999, S. 379–388

[39] TONG, S.: Support Vector Machine Active Learning with Applications to Text Classification. In: *17th International Conference on Machine Learning, 2000* (2000), S. 999–1006. – URL https://ci.nii.ac.jp/naid/10025099493/en/

[40] Villalobos-Arias, Leonardo ; Quesada-López, Christian ; Guevara-Coto, Jose ; Martínez, Alexandra ; Jenkins, Marcelo: *Evaluating Hyper-Parameter Tuning Using Random Search in Support Vector Machines for Software Effort Estimation.* S. 31–40. In: *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering.* New York, NY, USA : Association for Computing Machinery, 2020. – URL https://doi.org/10.1145/3416508.3417121. – ISBN 9781450381277

[41] Vruniotis, Vasilis: Machine Learning Tutorial: The Multinomial Logistic Regression (Softmax Regression)| Datumbox. In: *Blog. datumbox. com. Np* (2017)

[42] Weerts, Hilde J. P. ; Mueller, Andreas C. ; Vanschoren, Joaquin: Importance of Tuning Hyperparameters of Machine Learning Algorithms. In: *CoRR* abs/2007.07588 (2020). – URL https://arxiv.org/abs/2007.07588

[43] Weinert, Franz E.: Vergleichende Leistungsmessung in Schulen - eine umstrittene Selbstverständlichkeit. In: Weinert, Franz E. (Hrsg.): *Leistungsmessungen in Schulen.* Weinheim : Beltz, 2001, S. 17–31

[44] Williamson, David F. ; Parker, Robert A. ; Kendrick, Juliette S.: The box plot: a simple visual method to interpret data. In: *Annals of internal medicine* 110 (1989), Nr. 11, S. 916–921

[45] Xia, Linzhong ; Luo, Dean ; Zhang, Chunxiao ; Wu, Zhou: A Survey of Topic Models in Text Classification. In: *2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2019, S. 244–250

[46] Xu, Baoxun ; Guo, Xiufeng ; Ye, Yunming ; Cheng, Jiefeng: An Improved Random Forest Classifier for Text Categorization. In: *J. Comput.* 7 (2012), Nr. 12, S. 2913–2920

[47] Ying, Xue: An Overview of Overfitting and its Solutions. In: *Journal of Physics: Conference Series* 1168 (2019), feb, S. 022022. – URL https://doi.org/10.1088/1742-6596/1168/2/022022

[48] Zhang, Yin ; Jin, Rong ; Zhou, Zhi-Hua: Understanding bag-of-words model: a statistical framework. In: *International Journal of Machine Learning and Cybernetics* 1 (2010), Nr. 1, S. 43–52

[49] Zhang, Zhongheng: Introduction to machine learning: k-nearest neighbors. In: *Annals of translational medicine* 4 (2016), June, Nr. 11, S. 218. – URL https://europepmc.org/articles/PMC4916348. – ISSN 2305-5839

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original