Juristische Mustererkennung: Einsatz von Decision Trees und Random Forests auf Bundesgerichtsfälle in der Schweiz

Im Rahmen des Moduls "Machine Learning"

an der Digital Business University of Applied Sciences DBU Berlin

Autor:	Simon Sahli
Studiengang:	Data Science und Management (M.Sc.)
Immatrikulation:	190252
Adresse:	CH-3065 Bern
E-Mail:	simon.sahli@student.dbuas.de
Datum der Einreichung:	23.12.2023

Inhaltsverzeichnis

1 Einleitung **3**

2 Datensatz und Problemstellung **4**

3 Theorie und Modellierung **8**

4 Anwendung **11**

4.1 Vorbereitung der Daten 11

4.2 Anwendung und Ergebnisse der Methoden 13

4.2.1 Ergebnisse des Decision-Tree-Modells 13

4.2.2 Ergebnisse des Random-Forest-Modells 16

4.3 Evaluation der Ergebnisse 18

5 Fazit **18**

6 Quellenverzeichnis **20**

A Appendix **21**

A.1 Datensatz 21

A.2 Code 22

1 Einleitung

Die folgende Studienarbeit hat das Ziel, den von Geering und Merane (2023)[11] aufbereiteten Datensatz des Bundesgerichts der Schweiz mit Hilfe von Supervised Machine Learning-Algorithmen zu klassifizieren. Dabei wird ein Klassifikations-Algorithmus auf 107'497 von dem obersten Gericht entschiedenen Fällen zwischen 2007 und September 2023 trainiert. Das grundlegende Ziel ist es, ein Modell zu trainieren, welches mit einer Auswahl aus 31 dokumentierten Fallvariablen die vom Gericht behandelten Fälle nach ihrem Ausgang klassifizieren kann. Der Fokus der Arbeit liegt in der Analyse von Metadaten – eine vertiefte qualitative Auseinandersetzung mit den Urteilstexten (Textanalysen) wird nicht durchgeführt.

Die Ergebnisse zeigen, dass Machine Learning-Modelle wie Decision Trees und Random Forests eingesetzt werden können, um Muster in juristischen Daten zu identifizieren und Vorhersagen über die Ausgänge von Gerichtsverfahren zu treffen.
Es ist jedoch wichtig, die Grenzen der Vorhersagekraft zu erkennen und zu berücksichtigen, dass juristische Entscheidungen komplex sind und durch eine Vielzahl von Faktoren beeinflusst werden, die nicht vollständig in den Metadaten des Datensatzes erfasst werden können. Dennoch: Ein Machine Learning-Modell, das die Ausgänge von Gerichtsverfahren klassifiziert, kann in der Forschung/Analytik der Rechtswissenschaft und allgemein in der Verwaltung sowie im öffentlichen Verständnis von Nutzen sein. So könnte das öffentliche Verständnis für die Funktionsweise des Rechtssystems verbessert werden, indem es Muster und Tendenzen in Gerichtsentscheidungen aufzeigt. Zudem könnte eine Weiterentwicklung eines solchen Modells in der juristischen Ausbildung eingesetzt werden, um Studierenden und Forschenden zu helfen, Muster und Trends in Gerichtsentscheidungen zu verstehen. Es könnte auch genutzt werden, um zu untersuchen, wie verschiedene Faktoren wie die Zusammensetzung des Gerichts oder der Bereich des Rechts die Urteile beeinflussen.

Eine Übersicht über dieses Machine Learning Projekt, ein Screencast so wie auch die Datengrundlage ist unter folgendem Link einsehbar:
`https://www.simonsahli.ch/project_bundesgerichtsfaelle.html`

2 Datensatz und Problemstellung

Der von Geering und Merane (2023)[11] zur Verfügung gestellte Bundesgerichtsdatensatz ist eine Sammlung der Rechtsprechungen, die mit Hilfe eines Webscrapers und einer Pipeline automatisiert aufbereitet wurde. Die Quelle des Webscrapings bzw. die einzelnen Rechtsprechungen können auf dem Online-Auftritt des Bundesgerichtes eingesehen werden[1].
Im Folgenden werden die wichtigsten Variablen des Datensatzes vorgestellt und eine explorative Datenanalyse (EDA) durchgeführt. Eine vollständige Übersicht über alle Variablen des Datensatzes ist im Anhang dieser Arbeit aufgeführt.

Im Datensatz sind 118'443 Beobachtungen, also Rechtsfälle des schweizerischen Bundesgerichtes der Jahre 2007 bis 2023, aufgeführt. Durchschnittlich existieren 6'967 Rechtsfälle je Jahr, über welche Daten einsehbar sind.
Das Feature, welches in dieser Studienarbeit als abhängige Variable Y definiert wird, ist die Variable mit dem Namen "outcome" bzw. "outcome_binary", welche den Ausgang des Falles beim Bundesgericht charakterisiert. Urteilsausgänge wurden in fünf ("outcome") bzw. zwei ("outcome_binary") verschiedene Kategorien eingeteilt:
Die Berufung wurde angenommen (gutgeheissen) = "granted",
die Berufung wurde teilweise angenommen (teilweise gutgeheissen) = "partly granted",
die Berufung wurde abgelehnt (abgewiesen) = "rejected",
die Berufung war unzulässig (nicht eingetreten) = "inadmissible",
die Berufung wurde abgeschrieben = "writeoff".
Die Reduktion der Variable auf "outcome_binary" entspricht der Frage, ob die Berufung erfolgreich war oder nicht. Wenn die Berufung angenommen oder teilweise angenommen wurde, wird WAHR kodiert. Wenn die Berufung unzulässig oder abgelehnt wurde, wird FALSCH kodiert. Berufungen, die abgeschrieben wurden, werden als NA kodiert.

Es wurde festgestellt, dass 3'088 (2.6 %) der Beobachtungen in keinem Urteilsausgang ("outcome") klassifiziert wurden. Dies kann verschiedenartig interpretiert werden. Am plausibelsten ist, dass bei diesen Beobachtungen das eigentliche Urteil noch ausstehend ist.

[1] `https://www.bger.ch/index/juridiction/jurisdiction-inherit-template/jurisdiction-recht.htm`

Abbildung 1 zeigt die Verteilung der Urteilsausgänge über die Jahre. Es fällt auf, dass die meisten Berufungen abgewiesen wurden oder unzulässig waren. Die Beobachtungen in den Jahren 2007 und 2023 umfassen nicht das gesamte Kalenderjahr, was zu entsprechenden Schwankungen führt.

Abbildung 1: Verteilung der Urteilsausgänge über die Jahre

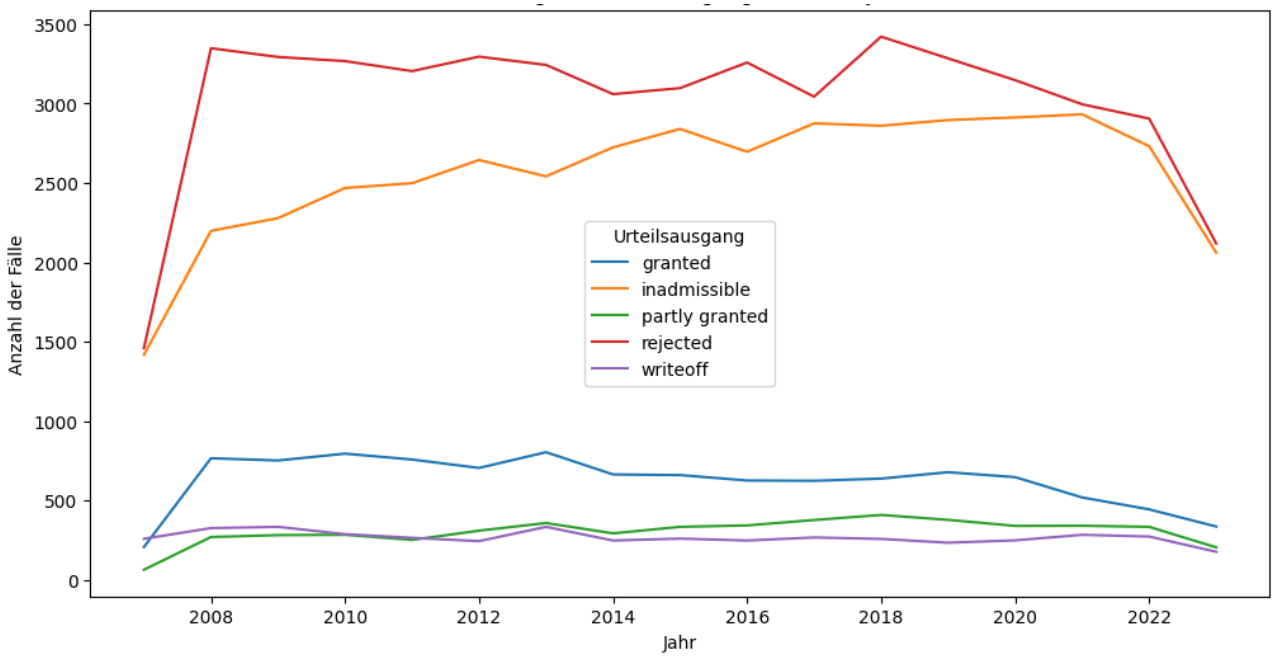


Abbildung 2: Durchschnittliche Urteilsausgänge pro Jahr und Rechtsgebiet - Gestapeltes Balkendiagramm

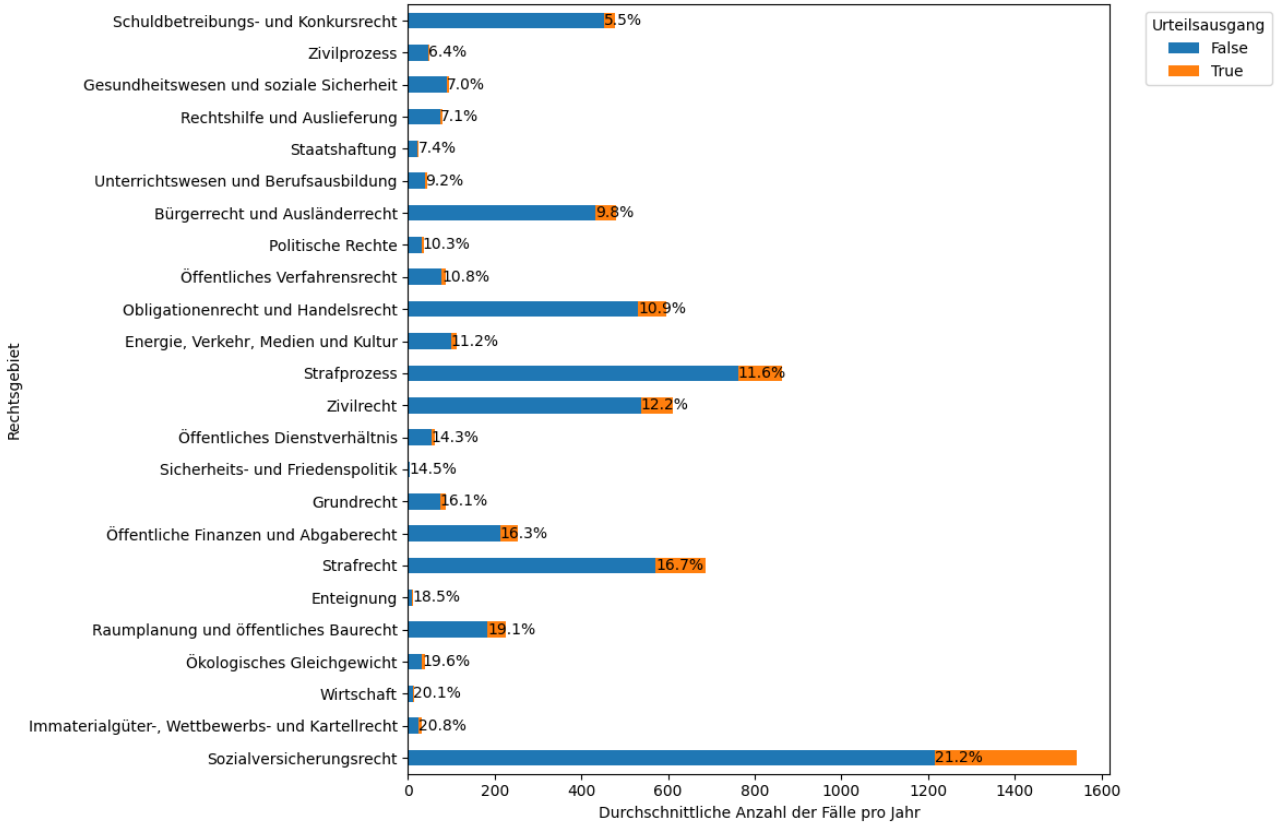


Abbildung 2 stellt den binär codierten Erfolg der Berufungen am Bundesgericht, aufgeteilt auf verschiedene Rechtsgebiete ("area_intermediate"), dar. Durchschnittlich werden jährlich am meisten Berufungen im Sozialversicherungsrecht behandelt, wobei im Schnitt 21.2 % der Berufungen in diesem Gebiet erfolgreich sind.

Die meisten Fälle werden in deutscher Sprache abgewickelt (Abbildung 3) und betreffen das öffentliche Recht (Abbildung 4). Die durchschnittliche Verfahrensdauer beträgt im öffentlichen Recht über 175 Tage, wie in Abbildung 5 ersichtlich.

Abbildung 3: Fälle nach Landessprache (de/fr/it)

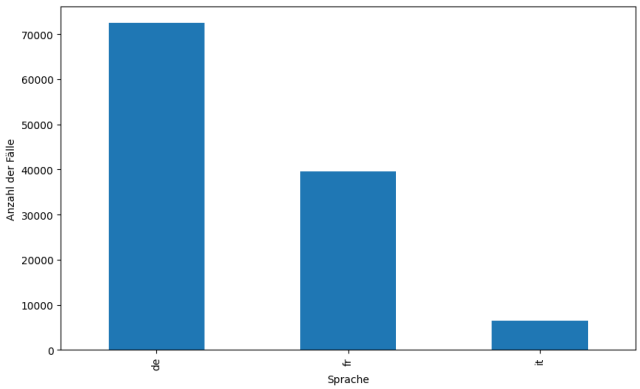


Abbildung 4: Fälle nach Rechtsgebiet

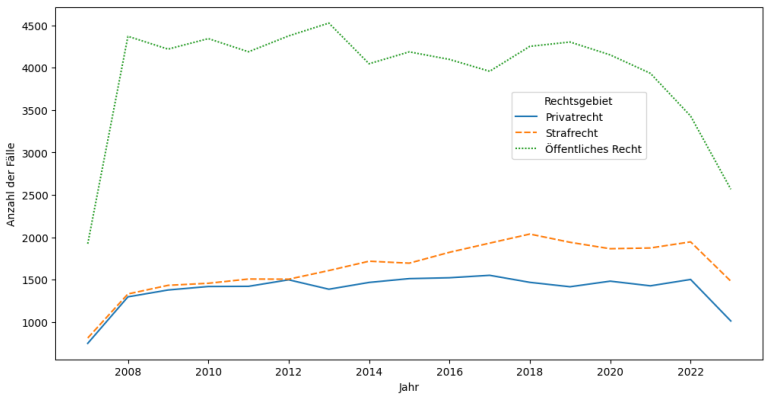


Abbildung 5: Verfahrensdauer je Rechtsgebiet

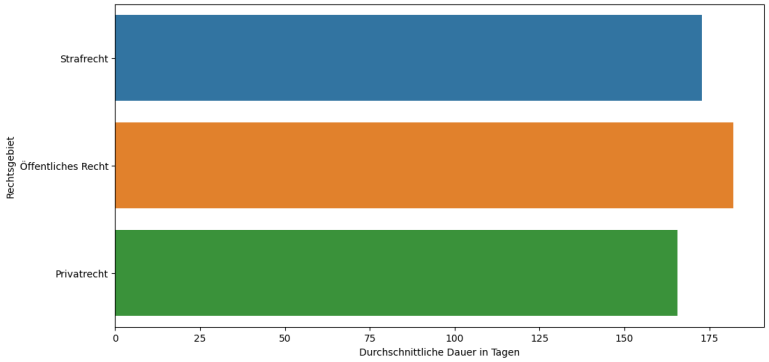


Abbildung 6: Anzahl der Fälle je Ursprungskanton

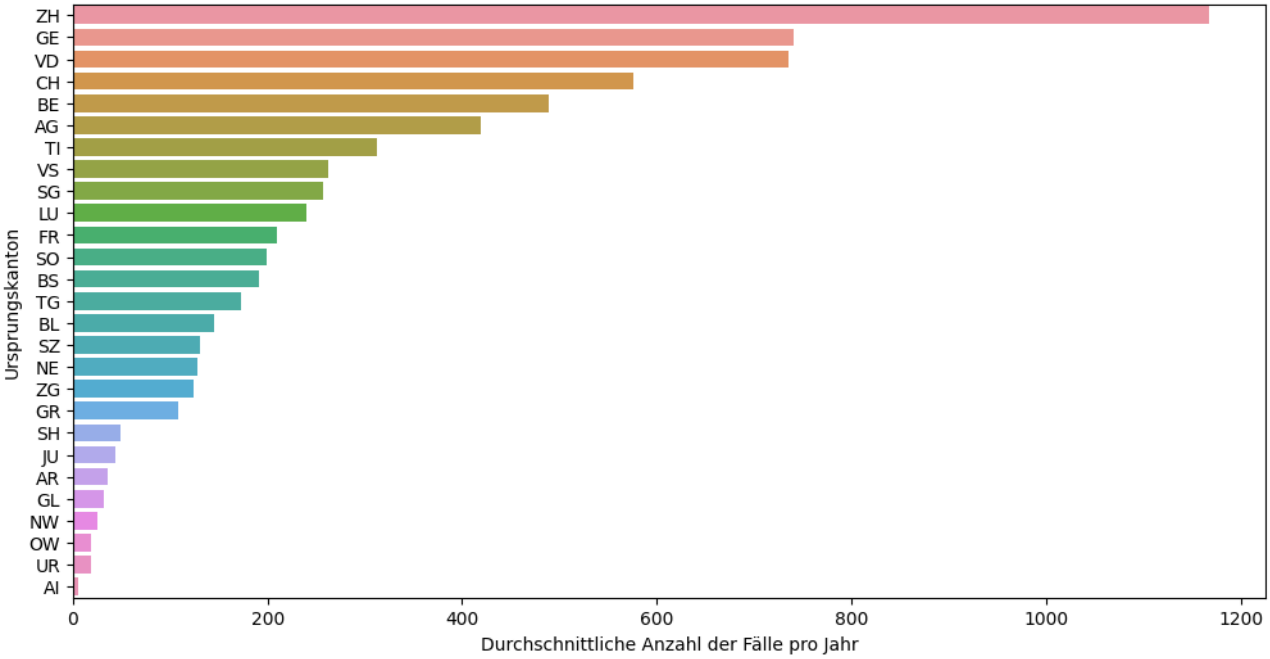


Abbildung 6 zeigt die durchschnittliche Anzahl der Fälle je Jahr und Ursprungskanton. Die meisten Fälle stammen aus dem Kanton Zürich, was aufgrund der Bevölkerungszahlen plausibel ist.

Abschliessend im Rahmen der explorativen Datenanalyse wurde die anwaltliche Vertretung sowohl auf Seiten der Berufungskläger (Abbildung 7) als auch auf Seiten der beklagten Partei (Abbildung 8) analysiert. Es ist auffallend, dass insbesondere natürliche Personen als Berufungskläger auftreten und eine beachtliche Anzahl dieser Personen sich nicht durch einen Anwalt vertreten lässt, was auf individuelle Entscheidungen oder finanzielle Überlegungen zurückzuführen sein könnte.
Auf der anderen Seite sind es vor allem staatliche Stellen, die beklagt werden. Hier sind die meisten Fälle ohne Anwaltsvertretung, was darauf hindeuten könnte, dass der Staat als Beklagter auftritt und möglicherweise nicht immer externe rechtliche Vertretung benötigt, da er über interne Rechtsberatung verfügen könnte.

Abbildung 7: Vertretung - Berufungskläger

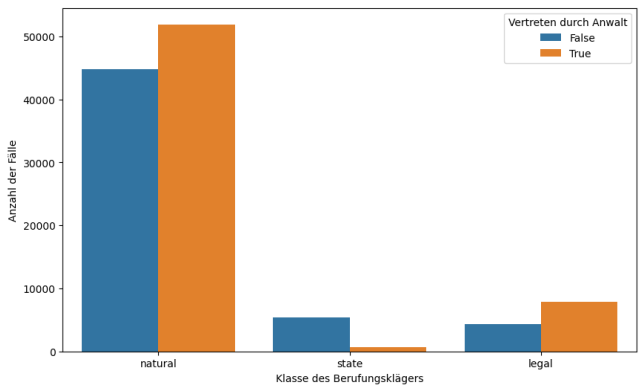
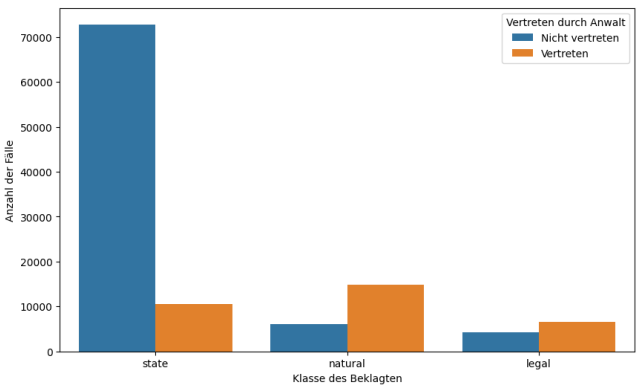


Abbildung 8: Vertretung - Beklagte



Die zentrale Frage ist, ob mit einem supervised Machine Learning Algorithmus, der auf einem Teil des Datensatzes der Metadaten ohne Outcome-Features trainiert wird, vorausgeahnt werden kann, in welche Richtung die Outcomes gehen könnten. Ein bestimmtes Muster von Rechtsgebiet, Thema, Verfahrensdauer, Anzahl der Richter und Anzahl zitierter Urteile könnte bereits bis zu einem gewissen Grad die Outcomes andeuten.
Im nächsten Kapitel werden Decision Trees als mögliche Methode innerhalb von supervised Learning Modellen vorgestellt, bevor ein Modell mit Hilfe des Datensatzes trainiert wird. Zudem wird eine Ensemble Learning Methode vorgestellt, welche Decision Trees als Grundlage verwendet: Random Forest.

3 Theorie und Modellierung

Im Folgenden werden die theoretischen Grundlagen eines Supervised Learning-Modells dargelegt, wobei der Fokus auf einem Decision Tree-Modell liegt. Mohri, Rostamizadeh und Talwalkar (2012)[8] erläutern die Grundprinzipien des "Supervised Learning": Trainingsdaten werden verarbeitet, um eine Funktion zu bilden, die Daten (X) auf erwartete Ausgabewerte (y) abbildet. Betrachtet wird entsprechend eine Funktion $y = f(X)$, wobei f unbekannt ist. Das Hauptziel eines Supervised Learning-Modells ist es, ein Modell $\tilde{f}$ zu finden, das f möglichst genau approximiert, ohne sich zu sehr auf die Trainingsdaten zu fixieren (Overfitting) und ohne nur an der allgemeinen Oberfläche zu bleiben (Underfitting). In der Realität ist die Datenerhebung immer mit Zufälligkeiten – oder Noise – behaftet. Ziel ist es, ein Modell zu finden, welches die Modellkomplexität zwischen Verzerrung und Varianz ausbalanciert, so dass der Fehler der Verallgemeinerung (Generalization Error) möglichst gering ist. Da keine weiteren Datensätze vorliegen und die Funktion f unbekannt bleibt, werden Techniken angewendet, um $\tilde{f}$ bzw. den Generalization Error bestmöglich zu approximieren. Eine dieser Techniken ist die Cross-Validation (CV), die Lyashenko (2022)[6] in einem Blogbeitrag detailliert vorstellt. Hierbei wird die Datengrundlage zunächst in Trainings- und Testdaten aufgeteilt. Um die Leistungsfähigkeit eines Algorithmus über verschiedene Eingabedaten hinweg zu testen, wird der Trainingsdatensatz im Rahmen der CV weiter in k Teilmengen unterteilt (K-Fold-CV). Der sogenannte CV-Fehler entspricht dem Durchschnitt der Fehler der einzelnen K-Folds und wird schliesslich mit dem Fehler der Trainingsdaten verglichen.

Im folgenden Abschnitt wird das angewendete Modell, der sogenannte Klassifizierungsbaum (englisch: "Decision Tree"), genauer erläutert. Ein Klassifizierungsbaum macht an einer Knotenstelle, der Wurzel, eine Aussage (Beispiel: "Beim betrachteten Rechtsfall waren mehr als ein Richter involviert") und teilt die Beobachtungen je nach Antwort auf die Aussage in zwei Äste, auch Blätter genannt (Beispiel: "TRUE/1", "FALSE/0"). Die Pfade von der Wurzel zum Blatt repräsentieren die Klassifizierungsregeln.

Der Baum wird auf einem Datensatz mit bekannten Labels trainiert, um Muster aus den Merkmalen zu lernen, sodass möglichst "reine" Blätter entstehen. Das heisst, der Baum wird so trainiert, dass in jedem Blatt vorwiegend ein Klassen-Label vorherrscht. Bei Decision Trees ist entscheidend, welche Fragen an welchen Knoten gestellt werden. Unreine (englisch: "impure") Abfragen implizieren eine nicht perfekte Teilung des Knotens.

$$IG(\underbrace{f}_{\text{feature}}, \underbrace{sp}_{\text{split-point}}) = I(\text{ parent }) - \left(\frac{N_{\text{left}}}{N} I(\text{ left }) + \frac{N_{\text{right}}}{N} I(\text{ right })\right) \tag{1}$$

Es gibt verschiedene Methoden, diese Unreinheit zu bestimmen. Charbuty und Abdulazeez (2021)[4] stellen in ihrer Literatur-Review zu Decision Trees die verschiedenen Ansätze übersichtlich gegenüber. Häufig werden der Gini-Index oder Entropie-Masse im Rahmen des Konzepts des Informationsgewinns IG (englisch: "Information gain") verwendet.
Der Gini-Index kann beispielsweise als Mass für die Unreinheit I in der Formel des Informationsgewinns IG (1) genutzt werden. Der Informationsgewinn ist die Unreinheit des Elternknotens minus die gewichtete Summe der Unreinheiten der Kindsknoten innerhalb einer bestimmten Stelle eines Klassifizierungsbaums. Die beste Aufteilung ist jene, die den höchsten Informationsgewinn und somit die grösste Reduktion der Unreinheit bringt. Ein niedriger Gini-Index deutet auf hohe Reinheit in einem Knoten hin, während ein hoher Gini-Index auf eine gemischte Gruppe von Daten hinweist. Der Gini-Index wird für jede mögliche Aufteilung der Daten berechnet, um die Aufteilung zu finden, die den grössten Rückgang des Gini-Index – also den höchsten Informationsgewinn – verursacht. Der Informationsgewinn misst somit, wie viel "Unsicherheit" über die Klassenzugehörigkeit durch eine Aufteilung entfernt wird.

Wie diese kurze Erläuterung bereits ahnen lässt, sind Entscheidungsbäume – im Vergleich mit anderen supervised Learning Modellen – einfach zu verstehen und interpretieren. Song und Lu (2015)[10] verdeutlichen in Ihrer Arbeit die Vorteile eines solchen Modells. Nebst der einfachen Interpretierbarkeit sind Entscheidungsbäume zudem in Ihrer Anwendung vergleichsweise flexibel: So lassen sich unterschiedliche Datentypen mischen und es können nicht-lineare Zusammenhänge dargestellt werden. Aufgrund des Fehlens von Distanzmassen, müssen die Features auch keinem grossen Data-Preprocessing unterziehen und sind gemäss Ausführungen der Autoren robust gegen Ausreisser. Trotz den vielen Vorteilen bleiben die Nachteile von Decision Trees vorhanden: Modelle mit Charakteristiken eines Decision Trees leiden unter einer hohen Varianz, wenn sie ohne grossen Einschränkungen trainiert werden. In verschiedenen Arbeiten, wie beispielsweise in der Arbeit von Bramer (2002)[3], wird deutlich, dass ein Decision Tree Modell unter dem Problem von zu vielen Abzweigungen (zu lange Bäume) und Regeln auf Trainingsdaten leiden kann, was zu Overfitting (zu geringe Vorhersagekraft für zuvor ungesehene Daten) führt.

Eine Lösung, die sich die Flexibilität der Decision Tree Modelle zunutze macht, und gleichzeitig ihre Tendenz, Noice in den Daten zu lernen, reduziert, ist Ensemble Learning. So erklären Mohammed und Kora (2023)[7], dass Ensemble Learning sich auf eine Lernmethode bezieht, welche mehrere Basismodelle kombiniert, um ein leistungsfähigeres und wenn möglich genaueres Modell zu erstellen. Beispielsweise könnten für ein Klassifikationsproblem sowohl ein Decision Tree als auch ein knn Modell verwendet werden - jedes Modell gibt seine Vorhersage ab. Das Metamodell sammelt diese Vorhersagen und gibt eine endgültige Vorhersage, die von der Art des Problems abhängt. Beim sogenannten Bagging, wie von Abellán und Masegosa (2009)[1] aufgezeigt, wird das Ensemble aus Modellen gebildet, die den gleichen Trainingsalgorithmus verwenden. Jedoch werden diese Modelle nicht auf der gesamten Trainingsmenge trainiert. Stattdessen wird jedes Modell auf einer anderen Teilmenge der Daten trainiert. Wird ein Bagging-Ansatz mit Entscheidungsbäume als Grundlage weiterverfolgt, dabei jeder einzelne Baum auf ein varrierendes Bootstrap-Sample trainiert, welches den gleichen Umfang hat wie die Trainingsmenge, und dabei ebenfalls nur ein zufällig gewählte Anzahl features d je Entscheidungsknoten ausgewählt werden, wobei d kleiner definiert wird als maximal verfügbar stehende Features, dann wird diese Art von Bagging auch Random Forest genannt. Solche "Zufallsswälder" kombinieren die Einfachheit von Entscheidungsbäumen mit Flexibilität, was zu einer erheblichen Verbesserung der Genauigkeit führt.

Ein Ziel ist es folglich, einerseits mit Hilfe von Grid-Search Cross Validation einen "optimalen" Decision Tree aus den Daten abzuleiten. Grid-Search Cross Validation, wie von Beheshti (2022)[2] anschaulich dargestellt, teilt den gesamten verfügbaren (Lern-)Datensatz in verschiedene kleinere Datensätze auf, welche einerseits zum lernen und anderseits zum testen von verschiedenen Parameterkombinationen (Hyperparameter-Tuning) verwendet werden können. Anderseits ist das Ziel, in einem zweiten Schritt ein Random Forest aufzugeleisen, d.h. mit Hilfe der "Bagging"-Methode eine Vielzahl von Klassifikationsbäume gleichzeitig zu berechnen. Die Ergebnisse beider Vorgehensweisen sollen verglichen werden.

In einem Parameter-Grid für die Methode der Grid-Suche, angewendet auf Entscheidungsbäume oder Random-Forest-Modelle, werden verschiedene Parameterkombinationen gegeneinander abgewogen. So zeigt sich beispielsweise im Artikel von von Dash (2022)[5], dass diese Parameter folgende Einstellungen umfassen können, welche – nebst dem Random Forest Ansatz innerhalb von Ensemble Learning – die Gefahr von Overfitting bei Decision Trees reduzieren können, was gleichzeitig die Modelle optimiert:

- "n_estimators": Bestimmt die Anzahl der Bäume in einem Random-Forest Modell. Mehr Bäume können zu einer höheren Genauigkeit führen, erfordern aber Rechenleistung.
- "max_depth": Begrenzung der Ebenen / Tiefe eines Baumes. Je grösser die Tiefe, desto genauer das Modell, desto höher die Wahrscheinlichkeit für Overfitting.

- "min_samples_split": Minimale Anzahl von Datenpunkte je Knoten, welche für Aufteilung vorhanden sein müssen.
- "min_samples_leaf": Minimale Anzahl von Datenpunkte innerhalb eines Blattes.
- "max_features": Begrenzt die Anzahl der Merkmale, die für die Entscheidungsfindung in einem Baum verwendet werden.
- "boostrap": Ob Boostrap-Sampling (d.h. Ziehen mit Zurücklegen) für das Erstellen des Klassifikationsbaumes verwendet werden darf.
- "criterion": Das Kriterium, das für die Entscheidung zur Aufteilung der Daten an den Knoten verwendet wird, wie Gini-Index oder Entropie.

Für die Anwendung im Programmcode wird Pyothon-Bibliothek "scikit-learn"[9] verwendet.

4 Anwendung

4.1 Vorbereitung der Daten

Im Folgenden wir das im Kapitel 3 erklärte Modell des Entscheidungsbaums und Random Forst konkret auf den verfügbaren Datensatz angewendet. Zu Beginn stellt sich die Frage, was mit den Daten noch gemacht werden muss, damit die Modelle eingesetzt werden können (englisch: "Data preprocessing").

Laut der Dokumentation von scikit-learn können für die Anwendung der hier betrachteten Klassifikationsalgorithmen keine Datensätze mit fehlenden Werten (NaN) verwendet werden. Dies bedeutet, dass der Datensatz zunächst bereinigt werden muss. Im betrachteten Datensatz existieren fehlende Werte (NaN), die sich jedoch je nach Feature stark unterscheiden. Die meisten fehlenden Werte finden sich bei den Variablen "leading_case" (114'563) und "cited_bge" (20'387). Die Variable "leading_case" zeigt an, ob ein Urteil in der offiziellen Sammlung der Bundesgerichtsurteile (BGE) veröffentlicht wurde. Falls ja, wurde das Aktenzeichen des veröffentlichten Urteils als String kodiert, andernfalls als NaN. In diesem Kontext sind NaN-Werte also keine fehlenden Beobachtungen, sondern bedeuten "nicht veröffentlicht". Deshalb können sie im Modell berücksichtigt werden. Um dies zu ermöglichen, wird diese Variable im Programmcode binär kodiert (1 für veröffentlicht, 0 für nicht veröffentlicht). Ähnlich könnte mit der Variable "cited_bge" verfahren werden. Da jedoch auch eine Variable "n_cited_bge" im Datensatz existiert, welche die Anzahl an zitierten Fällen zurückgibt und dabei keine fehlenden Werte aufweist, wird "cited_bge" als Feature nicht weiter berücksichtigt.

Im Rahmen der EDA (vgl. Kapitel 2) wurde festgestellt, dass 7'636 Beobachtungen (6.4 %) nicht in einem Urteilsausgang ("outcome_binary") klassifiziert werden konnten. Aufgrund der geringen Anzahl dieser Fälle werden diese NaN-Einträge im Code nicht imputiert, sondern aus der Stichprobe entfernt. Folgende Zeilen an Beobachtungen werden ebenfalls aufgrund fehlender Einträge aus dem Sample entfernt:

- "source_canton" - 2.230 NaN-Beobachtungen
- "n_judges" - 595 NaN-Beobachtungen
- "area_intermediate" - 50 NaN-Beobachtungen
- "area_general" - 50 NaN-Beobachtungen
- "app_class" - 2'911 NaN-Beobachtungen
- "resp_class" - 2'911 NaN-Beobachtungen
- "proc_duration" - 1'064 NaN-Beobachtungen

Zudem werden Features aus dem Datensatz entfernt, die für die Implementierung in die Klassifikationsalgorithmen nicht relevant sind:

- "docref", "url', "date", "doi_version" - nicht relevant für Decision Trees
- "division", "area_detailed", "topic", "issue', 'source_date' - zu viele verschiedene Ausprägungen, Risiko des Overfitting
- "outcome", "cited_bger", "cited_bge" - andere Variablen sind besser geeignet

Nach dem Entfernen dieser nicht relevanten Features und dem Löschen der entsprechenden NaN-Werte verbleiben insgesamt 107'497 Datenzeilen mit 19 Features im Sample, was 90.76 % der ursprünglichen Beobachtungen entspricht.
Die 19 verbleibenden Features, die in das Datenmodell einfliessen, sind:

- "year", "proc_type", "merged_cases", "division_type", "n_judges", "language", "length", "area_general", "area_intermediate", "source_canton", "proc_duration", "app_class", "app_represented", "resp_class", "resp_represented", "outcome_binary", "n_cited_bger", "n_cited_bge", "leading_case"

Für alle verbleibenden kategorialen Features im Sample wird ein One-Hot-Dummy-Encoding durchgeführt. Beispielsweise wird das Feature "language", das drei Ausprägungen hat (Deutsch, Französisch, Italienisch), in zwei neue Variablen aufgeteilt: "language_deutsch" und "language_französisch", die jeweils 1 oder 0 annehmen können. Sind beide Variablen 0, wird implizit Italienisch als Sprache angenommen. Durch das One-Hot-Dummy-Encoding der verbleibenden kategorialen Variablen steigt die Anzahl der Features von 19 auf 99.

In einem nächsten Schritt werden X und y definiert, wobei X das gesamte Sample abzüglich der Variable "outcome_binary" umfasst. "Outcome_binary" wird separat als y abgetrennt. Es ist essenziell, vor weiteren Schritten den Datensatz in Trainings- und Testdaten zu unterteilen. Hierfür wird ein Verhältnis von 30 % Testdaten (X_test, y_test) zu 70 % Trainingsdaten (X_train, y_train) angewendet. Dies bedeutet, dass die Decision Trees und der Random Forest auf 70 % der zur Verfügung stehenden Daten trainiert werden. Das anschliessende Testen erfolgt auf den verbleibenden 30 % der Daten, die bis zum Testzeitpunkt nicht im Training verwendet wurden, um eine Verzerrung des Trainings zu verhindern.

Bei der Anwendung von Decision Tree- oder Random Forest-Modellen ist es nicht erforderlich, Data Preprocessing-Schritte wie das Skalieren von Daten durchzuführen. Dies liegt in der Natur dieser Algorithmen begründet, da sie nicht distanzbasiert sind. Das bedeutet, diese Modelle treffen Entscheidungen auf der Grundlage von bedingten Wahrscheinlichkeiten und nicht auf Distanzen zwischen Datenpunkten. Zusätzlich ist auch die Anwendung einer Hauptkomponentenanalyse (PCA) in diesem Kontext nicht notwendig.

4.2 Anwendung und Ergebnisse der Methoden

4.2.1 Ergebnisse des Decision-Tree-Modells

Die Anwendung eines DecisionTreeClassifier mit einem Random_state = 42 für die Reproduzierbarkeit ohne Hyperparamter-Tuning mit Hilfe Methoden wie Grid-Search Cross Validation, liefert das Ergebnis eines ersten vorläufigen Baumes (preliminary tree), welches mit Hilfe einer Confusion Matrix wie folgt dargestellt werden kann:

```
1 tree = DecisionTreeClassifier(random_state=42, class_weight="balanced")
```

		Vorhergesagt	
		Klasse 0	Klasse 1
Tatsächlich	Klasse 0	24'357	3'261
	Klasse 1	2'948	1'684

Tabelle 1: Confusion Matrix – preliminary tree

Klasse	Precision	Recall	F1-Score	Support
0.0	0.89	0.88	0.89	27'618
1.0	0.34	0.36	0.35	4'632
			Genauigkeit	0.81
			Makro-Durchschnitt	0.62
			Gewichteter Durchschnitt	0.81

Tabelle 2: Classification Report – preliminary tree

Einzig was bereits jetzt definiert wurde, ist "class_weight=balanced", um eine ausgewogenere Leistung über alle Klassen hinweg zu erzielen, da "outcome_binary" sehr unausgewogen im Datensatz vorhanden ist.

Die Ergebnisse aus der Confusion Matrix lassen sich so interpretieren, dass 24'357 Fälle, die Label 0 vorweisen, auch wirklich als 0 klassifiziert wurden. Falsch-positiv wurden jedoch in 3'261 Fällen gemessen: dies sind Fälle, die fälschlicherweise als outcome_binary = 1 identifiziert wurden. Etwas genaueren Aufschluss über den Erfolg der Klassifikation zeigt der Classification Report, welcher insbesondere drei Kennzahlen ausweist:

- Präzision/Genauigkeit: Das Verhältnis der richtig vorhergesagten positiven Beobachtungen zu den gesamten vorhergesagten positiven Beobachtungen.
- Recall/Sensitivität: Das Verhältnis der korrekt vorhergesagten positiven Beobachtungen zu allen Beobachtungen in der tatsächlichen Klasse.
- F1-Score: Der gewichtete Durchschnitt von Precision und Recall.

Zusammenfassend lässt sich sagen, dass dieser erste Entscheidungsbaum gut darin ist, die Klasse 0 zu identifizieren (hohe Präzision und Recall), aber weniger effektiv in der Identifizierung der Klasse 1 (niedrigere Präzision und Recall). Die Gesamtgenauigkeit des Modells ist mit 81 % relativ hoch, aber die Leistung in Bezug auf die Klasse 1 könnte verbessert werden.

Im folgenden Abschnitt wird erneut ein Entscheidungsbaum trainiert, diesmal jedoch mit Hyperparameter-Tuning mithilfe von GridSearch. Dabei werden 300 Kombinationen von Parametern

in einem Grid definiert, wobei jede Kombination als Kandidat für die optimale Konfiguration des Klassifikators gilt. Für jeden dieser 300 Kandidaten erfolgt eine 5-fache Kreuzvalidierung (englisch: cross-validation). Das bedeutet, der Trainingsdatensatz wird in 5 Teile (Folds) unterteilt. In jedem Durchlauf wird ein anderer Teil als Validierungsdatensatz verwendet. Dies wiederholt sich 5 Mal, sodass jeder Teil des Datensatzes einmal als Validierungsdatensatz dient. Da jeder der 300 Kandidaten mit einer 5-fachen Kreuzvalidierung bewertet wird, werden insgesamt 1500 individuelle Anpassungen (Fits) des Modells durchgeführt. Bei jedem Fit wird das Modell mit einer spezifischen Kombination von Hyperparametern auf den Trainingsdaten trainiert und anschliessend anhand der Validierungsdaten bewertet. Die genauen Definitionen des Grids und Spezifizierungen sind im Programmcode an entsprechender Stelle ausgewiesen.

Basierend auf unseren Daten scheint die folgende Kombination an Parametern den besten Decision Tree trainiert zu haben:

```
1 [{'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, '
    min_samples_split': 2}.
```

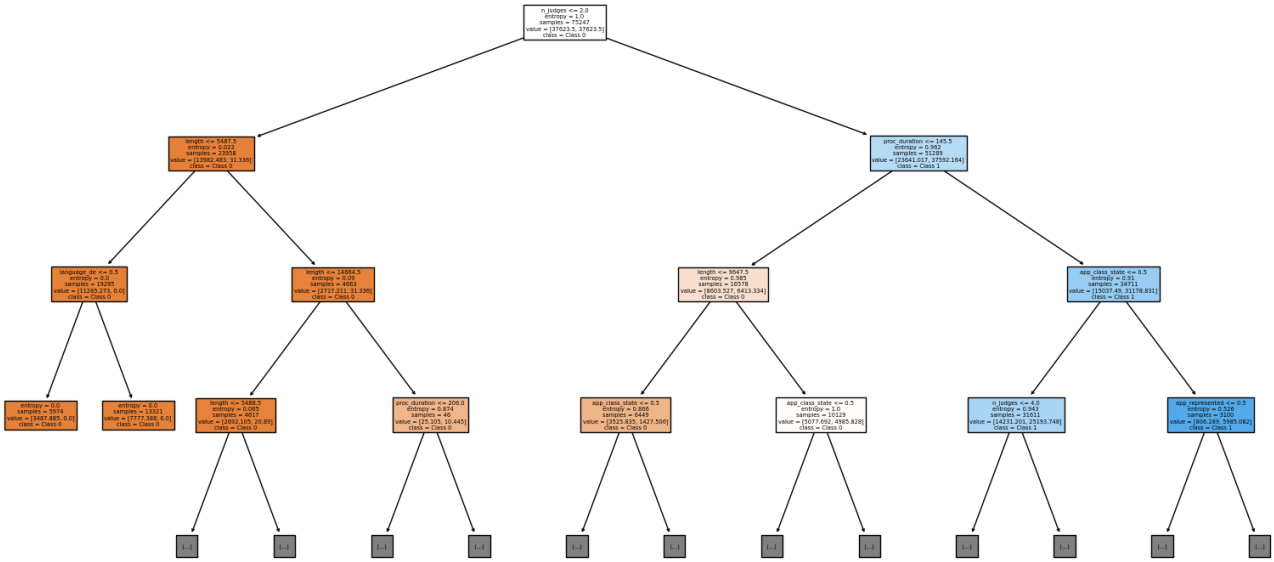
Zu erwähnen bleibt, dass der f1-score der Klasse 1 ("outcome_binary" = True) nur minimal verbessert werden konnte (vgl. Tabelle 3).

Klasse	Precision	Recall	F1-Score	Support
0.0	0.89	0.88	0.89	27'618
1.0	0.34	0.37	0.36	4'632
			Genauigkeit	0.81
			Makro-Durchschnitt	0.62
			Gewichteter Durchschnitt	0.81

Tabelle 3: Classification Report – nach Hyperparameter-Tuning

Abbildung 9 zeigt die ersten drei Ebenen des soeben hergeleiteten Entscheidungsbaumes und dient als Veranschaulichung des Klassifikators.

Abbildung 9: Erste drei Ebenen - Entscheidungsbaum nach Hyperparameter-Tuning



4.2.2 Ergebnisse des Random-Forest-Modells

Wie bereits in Kapitel 3 erörtert, stellt das Bagging eine Schlüsselkomponente in der Ensemble-Lernmethode dar. Laut Abellán und Masegosa (2009)[1] werden beim Bagging mehrere Modelle generiert, die alle denselben Trainingsalgorithmus verwenden, jedoch auf unterschiedlichen Teilmengen der Daten trainieren. Ein spezifischer Anwendungsfall ist der Einsatz von Entscheidungsbäumen innerhalb dieses Konzepts: Hierbei wird jeder Baum auf einer eigenen Bootstrap-Stichprobe trainiert, die denselben Umfang wie der gesamte Trainingsdatensatz aufweist. Diese spezielle Ausprägung des Bagging-Ansatzes ist als Random Forest bekannt und soll die Unkompliziertheit von Entscheidungsbäumen mit einer signifikanten Steigerung der Vorhersagegenauigkeit vereinen.
Dieser Ansatz wird auf die soeben hergeleiteten Trainings- und Testdaten angewendet.

Ähnlich wie im vorherigen Unterkapitel wird zunächst ein Random Forest ohne Hyperparameter-Tuning durchgeführt – das heisst, es wird ein vorläufiger Wald trainiert und ausgewertet. Dabei werden 100 Bäume gleichzeitig trainiert und in einem Metamodell des Waldes zusammengeführt. Der dadurch erzeugte Classification Report ist in Tabelle 4 dargestellt.

Klasse	Precision	Recall	F1-Score	Support
0.0	0.88	0.98	0.93	27'618
1.0	0.62	0.22	0.33	4'632
			Genauigkeit	0.87
			Makro-Durchschnitt	0.63
			Gewichteter Durchschnitt	0.84

Tabelle 4: Classification Report – Random Forest

Und ähnlich wie im letzten Unterkapitel werden im Rahmen des Random Forest erneut verschiedene Kombinationen von Parametern definiert, die mit Hilfe von GridSearch über verschiedene Subsamples des Datensatzes trainiert und getestet werden. Die genauen Spezifikationen der Parameter und was getestet wird, sind erneut im Programmcode an entsprechender Stelle definiert. Es wurden 5 Folds für 72 Parameterkombinationen durchgeführt, was 360 Fits von Random Forests entspricht. Zwei Parameter, die im Decision Tree Modell zunächst nicht spezifiziert werden konnten, sind die Anzahl der Bäume im Wald ("n_estimators"), welche an dieser Stelle 100, 200 und 300 annehmen können. Zudem wurde auch die Einstellung "bootstrap" = True oder False ins Parametergrid eingearbeitet, um die Auswirkung des Bootstrappings der einzelnen Ziehungen zu analysieren.

Klasse	Precision	Recall	F1-Score	Support
0.0	0.88	0.98	0.93	27'618
1.0	0.64	0.21	0.32	4'632
			Genauigkeit	0.87
			Makro-Durchschnitt	0.62
			Gewichteter Durchschnitt	0.84

Tabelle 5: Classification Report – Random Forest nach Hyperparameter-Tuning

Der Classification Report des optimierten Waldes mit der optimalen Parameterkombination hat die Präzision der Klasse 1 erhöht, während die Sensitivität reduziert wurde. Im folgenden Unterkapitel werden die Ergebnisse des optimierten Waldes und des Baumes verglichen.

4.3 Evaluation der Ergebnisse

Der Random Forest übertrifft den Decision Tree in Bezug auf die Gesamtgenauigkeit, mit 86.98 % gegenüber 80.74 %. Beide Modelle zeigen hohe Präzision und Recall für Klasse 0, wobei der Random Forest in dieser Klasse einen etwas höheren Recall hat.

Anders sieht es bei der Klasse 1 aus. Die Präzision für Klasse 1 beim Decision Tree Modell ist niedrig. Die zu identifizierende Zielvariable stellt in diesem Fall keine kritische Kondition dar, wie beispielsweise die Erkennung von Krebs oder Betrugsfällen. Das bedeutet, dass eine gewisse Anzahl an falsch positiven Fällen akzeptabel sein kann. Es sind dennoch keine guten Werte für die Präzision der Klasse 1. Besser sieht es beim Random Forest aus, welcher die Präzision in der Klasse 1 von 0.34 auf 0.64 erhöht – gleichzeitig sinkt jedoch der Recall von 0.37 auf 0.21. Eine höhere Präzision im Random Forest Modell bzgl. Klasse 1 bedeutet, dass weniger Fälle fälschlicherweise als Klasse 1 eingestuft wurden (weniger falsch positive Ergebnisse) als im Vergleich zum Decision Tree. Jedoch bedeutet die Reduktion des Recalls, dass die Fähigkeit des Modells, alle tatsächlichen positiven Fälle zu erkennen, abgenommen hat. Somit: Die Modelle zeigen einen Trade-off zwischen Präzision und Recall. Im Kontext dieser Daten ist es wichtig zu entscheiden, welcher Aspekt wichtiger ist – die Reduzierung falsch positiver Fälle (höhere Präzision) oder die Sicherstellung, dass keine positiven Fälle übersehen werden (höherer Recall). Dass der Random Forest eine höhere Präzision für Klasse 1 aufweist, deutet darauf hin, dass die Ensemble-Technik des Random Forest (die mehrere Entscheidungsbäume kombiniert und deren Ergebnisse mittelt), in diesem Fall effektiver ist, um eine genauere Klassifikation zu erreichen. Der niedrige Recall für Klasse 1 bleibt jedoch ein Bereich, der in beiden Modellen verbessert werden könnte.

5 Fazit

Diese Studienarbeit hat das Ziel verfolgt, einen umfangreichen Datensatz des Bundesgerichts der Schweiz mit Hilfe von Supervised Machine Learning-Algorithmen zu klassifizieren. Der Fokus lag dabei auf der Anwendung und Analyse von Metadaten, ohne die Urteilstexte qualitativ zu betrachten. Der Datensatz umfasst eine grosse Anzahl von Fällen und Variablen, die verschiedene Aspekte der Rechtsfälle abbilden.

Zentrales Element der Studienarbeit ist es, die Fähigkeit von Machine Learning-Modellen, insbesondere Decision Trees und Random Forests, zu bewerten, um die Ausgänge von Gerichtsverfahren vorherzusagen. Die Modelle wurden umfangreich analysiert, mit dem Ziel, ein Gleichgewicht zwischen Overfitting und Underfitting zu erreichen und den Generalization Error zu minimieren.

Beim Decision Tree-Modell zeigte sich eine hohe Genauigkeit in der Identifikation der Klasse 0, aber Schwächen in der Klasse 1. Das Random Forest-Modell verbesserte die Präzision in der Klasse 1, hatte jedoch einen geringeren Recall, was auf einen Trade-off zwischen Präzision und Recall hindeutet. Trotz der Herausforderungen im Zusammenhang mit der Klasse 1 übertraf der Random Forest in der Gesamtgenauigkeit das Decision Tree-Modell.

Die Ergebnisse zeigen, dass Machine Learning-Modelle nützlich sein können, um Muster und Trends in juristischen Daten zu erkennen. Sie könnten sowohl in der Forschung als auch in der Praxis eingesetzt werden, um das Verständnis für Gerichtsentscheidungen zu verbessern. Es ist jedoch wichtig, die Grenzen dieser Modelle zu erkennen, insbesondere im Hinblick auf komplexe juristische Entscheidungen, die nicht vollständig durch Metadaten erfasst werden können.

Insgesamt leistet diese Arbeit einen Beitrag zur möglichen Anwendung von Machine Learning in der Rechtswissenschaft und bietet einen Ausgangspunkt für weitere Forschungen in diesem Bereich. Zukünftige Arbeiten könnten darauf abzielen, die Modelle weiter zu verfeinern und andere Machine Learning-Techniken zu untersuchen, um die Vorhersagegenauigkeit zu verbessern.
Ein besonders vielversprechender Forschungszweig könnte sich der Kombination von qualitativen Inhaltsanalysen und Machine Learning-Techniken widmen. Diese Herangehensweise hätte das Potenzial, ein tieferes Verständnis der Rechtssprechungen zu entwickeln. Durch die Analyse von Texten und Urteilen könnten zukünftige Modelle entwickelt werden, die nicht nur Ausgänge von Fällen vorhersagen, sondern auch als Rechtsberatungs-Tools fungieren könnten.
Ein solches fortgeschrittenes Modell könnte als eine Art Rechtsschutz oder Rechtsexperte dienen, welches auf der gesamten Rechtssprache der Schweiz trainiert ist. Es könnte in der Lage sein, Vorhersagen darüber zu treffen, wie erfolgreich ein juristisches Anliegen sein könnte, wenn es verfolgt wird. Dies wäre besonders wertvoll, da juristische Verfahren oft mit hohen Kosten verbunden sind. Ein solches Tool könnte daher dazu beitragen, die Kosten und den Nutzen von Rechtssprechungen im Voraus abzuwägen, wodurch sowohl Einzelpersonen als auch Organisationen unterstützt werden könnten, fundiertere Entscheidungen zu treffen.

6 Quellenverzeichnis

[1] Abellán, Joaquín, and Andrés R. Masegosa. 2009. "An Experimental Study about Simple Decision Trees for Bagging Ensemble on Datasets with Classification Noise." In Symbolic and Quantitative Approaches to Reasoning with Uncertainty, edited by Claudio Sossai and Gaetano Chemello, 446–56. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-02906-6_39.

[2] Beheshti, Nima. 2022. "Cross Validation and Grid Search." Medium. February 5, 2022. https://towardsdatascience.com/cross-validation-and-grid-search-efa64b127c1b.

[3] Bramer, Max. 2002. "Using J-Pruning to Reduce Overfitting in Classification Trees." In Research and Development in Intelligent Systems XVIII, edited by Max Bramer, Frans Coenen, and Alun Preece, 25–38. London: Springer. https://doi.org/10.1007/978-1-4471-0119-2_3.

[4] Charbuty, Bahzad, and Adnan Abdulazeez. 2021. "Classification Based on Decision Tree Algorithm for Machine Learning." Journal of Applied Science and Technology Trends 2 (01): 20–28. https://doi.org/10.38094/jastt20165.

[5] Dash, Shailey. 2022. "Decision Trees Explained — Entropy, Information Gain, Gini Index, CCP Pruning.." Medium. November 2, 2022. https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c.

[6] Lyashenko, Vladimir. 2022. "Cross-Validation in Machine Learning: How to Do It Right." Neptune.Ai. July 21, 2022. https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right.

[7] Mohammed, Ammar, and Rania Kora. 2023. "A Comprehensive Review on Ensemble Deep Learning: Opportunities and Challenges." Journal of King Saud University - Computer and Information Sciences 35 (2): 757–74. https://doi.org/10.1016/j.jksuci.2023.01.014.

[8] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. Foundations of Machine Learning. Adaptive Computation and Machine Learning. Cambridge, Mass. London: The MIT Press.

[9] "Scikit-Learn: Machine Learning in Python — Scikit-Learn 1.3.2 Documentation." n.d. Accessed December 9, 2023. https://scikit-learn.org/stable/.

[10] SONG, Yan-yan, and Ying LU. 2015. "Decision Tree Methods: Applications for Classification and Prediction." Shanghai Archives of Psychiatry 27 (2): 130–35. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/

A Appendix

A.1 Datensatz

Quelle

[11] Florian Geering and Jakob Merane. (2023). Swiss Federal Supreme Court Dataset (SCD). Version 2023-4. Zenodo.
https://zenodo.org/records/10038641

Eine detaillierte Übersicht über die im Zusammenzug des Datensatzes von Geering und Merone gespeicherten Variablen und deren Ausprägungen ist im sogenannten "codebook"[2] ersichtlich.
Anbei die wichtigsten Variablen knapp erläutert:

docref: Eindeutige Fall-Nummer gemäss dem Case Numbering System. (String)

date: Datum des im Urteil des Bundesgerichts angegebenen Urteils.

proc_type: Verfahrensart, wie sie in der Fallnummer (docref) angegeben ist.

merged_cases: Zeigt an, ob das Urteil zusammengeführte Fälle beinhaltet.

division: Abteilung des Gerichts, die den Fall entschieden hat.

division_type: Typ der Abteilung des Gerichts, basierend auf docref und division.

n_judges: Anzahl der am Fall beteiligten Richter.

language: Sprache des Urteilstextes.

length: Länge des Urteilstextes in Zeichen.

area_general: Allgemeines Rechtsgebiet des Urteils auf niedriger Granularität. Die Variable area_general nimmt drei verschiedene Stufen an und wird aus den 24 Stufen in der Variable area_intermediate aggregiert, die wiederum aus den 56 Stufen in der Variable area_detailed aggregiert wird.

topic: Thema des Urteils, wie vom Bundesgericht angegeben.

issue: Urteilsthema, wie im Text angegeben.

source_date: Datum der angefochtenen Entscheidung.

source_canton: Ursprung der angefochtenen Entscheidung.

proc_duration: Ungefähre Dauer des Verfahrens beim Bundesgericht.

[2] https://zenodo.org/records/10038641/preview/codebook-2023-4.pdf?include_deleted=0

app_class: Typ des Berufungsklägers.

app_represented: Vertretung der Seite des Berufungsklägers.

resp_class: Typ des Beklagten.

resp_represented: Vertretung der Seite des Beklagten.

outcome: Ausgang des Falles beim Bundesgericht.

outcome_binary: Erfolg der Berufung (wahr/falsch).

cited_bger: Zitierte unveröffentlichte Urteile des Bundesgerichts.

n_cited_bger: Anzahl zitierter unveröffentlichter Urteile.

cited_bge: Zitierte veröffentlichte Urteile des Bundesgerichts (BGE).

n_cited_bge: Anzahl zitierter veröffentlichter Urteile (BGE).

leading_case: Veröffentlichung des Urteils in der BGE-Sammlung.

doi_version: DOI der spezifischen Version des Datensatzes.

A.2 Code

Der für diese Arbeit verwendete Programmcode ist in zwei Python Jupyter Notebooks dokumentiert und online verfügbar:

1. Explorative Datenanalyse EDA:
`https://www.simonsahli.ch/code/bundesgerichtsfaelle-a-eda.ipynb`
2. Modellierung von Decision Trees und Random Forests:
`https://www.simonsahli.ch/code/bundesgerichtsfaelle-b-classification.ipynb`

Um die Analysen dieser Arbeit vollständig nachzuvollziehen, sollten die Notebooks in der angegebenen Reihenfolge betrachtet werden.