UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Entwicklung einer schwachen künstlichen Intelligenz als Gegner beim virtuellen Skat

Bachelorarbeit

Zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von

Dennis Bartschat

Erstgutachter: Prof. Dr. Steffan Müller
(Institut für Computervisualistik)

Zweitgutachter: M.Sc. Bastian Krayer
(Institut für Computervisualistik)

Koblenz, im Juli 2019

Inhaltsverzeichnis

Abstract (deutsch)

Das Ziel dieser Arbeit ist, die Entwicklung eines Skatcomputers unter Verwendung von Algorithmen aus dem Teilgebiet des künstlichen Lernens und deren Vergleich mit nicht lernenden Algorithmen. Dazu stellen sich zwei Forschungsfragen. Zum einen, ob maschinelles Lernen besser geeignet ist, als ein nicht lernender Algorithmus. Zum Anderen, welcher Lernalgorithmus am besten geeignet ist. Um diese Fragen zu klären, wurden verschiedene Algorithmen implementiert und anschließend gegeneinander getestet. Außerdem wurde getestet, wie gut die einzelnen Algorithmen, nach einer Trainingsphase, Spielsituationen während des Reizens, dem Vorgang bei dem der Alleinspieler im Skat bestimmt wird, zu bewerten.

Die Auswertung dieser Arbeit zeigte, dass es große Datenmengen benötigt, damit lernende Algorithmen überlegen sind, aber auch, dass es auf Dauer immer besser ist, einen solchen zu verwenden. Zudem zeigte sich, dass die Verteilung von Spielkarten auf drei mal zehn Karten nicht stochastisch unabhängig genug ist, um mit Ansätzen wie dem naiven Bayes Klassifikator gute Ergebnisse erzielen zu können. Auf dieser Grundlage ist es empfehlenswert, weitere Algorithmen zu testen, die keine stochastische Unabhängigkeit voraussetzen und über eine Möglichkeit nachzudenken, effizienter große Datensätze zu generieren, auf deren Basis die lernenden Algorithmen Entscheidungen treffen können.

Abstract (English)

The objective of this work is the development of a skatcomputer using algorithms of the field of machine learning and their comparison with non-learning algorithms. There are two main research questions. Firstly whether machine learning is better suited for this problem than a non-learning algorithm and secondly which learning algorithm is best suited. To answer these questions various algorithms were implemented and then tested and compared to each other. In addition it was tested how well the individual algorithms can evaluate a situation, during the bidding, the process in which the declarer is determined in Skat, after completing a training phase.

The evaluation of this work shows that it requires large amounts of data for the learning algorithms to become superior to non-learning algorithms, but also that it is always recommended to use them in the long run. In addition, the distribution of cards on three times ten cards was not stochastically independent enough to be able to achieve good results with some approaches such as the naive Bayes classifier. On this basis it is recommended to test other algorithms that do not require stochastic independence and think about a way to generate large data sets on which the learning algorithms can make decisions more efficiently.

1. Einleitung

Skat ist ein Kartenspiel, das besonders im deutschen Raum von großer Bedeutung ist. Neben unzähligen Regelabänderungen und regionalen Spielunterschieden hat es dieses Spiel, aber auch in einer vereinheitlichten Variante, bis zu internationalen Turnieren, Europa- und Weltmeisterschaften geschafft. Dabei ist Skat ein Spiel, bei dem es auf Erfahrung und Können ankommt, um auf diesen Turnieren bestehen zu können und bei dem gerade die erste Phase des Spiels, das Reizen, vielen Anfängern Kopfzerbrechen bereitet.

Die Forschung zu diesem Spiel begann dabei schon sehr früh und es existieren viele Werke, die gerade Anfängern helfen wollen ins Spiel hinein zu finden, aber auch das Spiel von Fortgeschrittenen verbessern wollen. Auch gibt es Werke, die die stochastischen Aspekte des Spiels darlegen und beschreiben, mit welcher Wahrscheinlichkeit welche Strategie zu welchem Ergebnis führt. Doch wenn es darum geht, nun einem Computer das Skatspielen beizubringen, findet man wenig Literatur. Zwar sind die ersten Ansätze, einen Skatcomputer zu entwickeln, recht alt, wie der „Skatchampion“ von 1980, jedoch ist Skat ein sehr komplexes Spiel, das nur schwer in einem Algorithmus zusammenzufassen ist. Besonders das Reizen ist eine Spielmechanik, die viel Erfahrung und Gespür verlangt, was mit einem Computer nur schwer umsetzbar ist. Doch genau wegen dieser Schwierigkeit, scheint es, ähnlich wie bei Schach und Go, eine faszinierende Herausforderung zu sein, einem Computer dieses Spiel beizubringen und dabei selbst die Schwierigkeit, dass es sich um ein Spiel mit imperfekter Information handelt, zu lösen.

Zwar wurde nachgewiesen, dass sich ein Algorithmus des maschinellen Lernens eignet, um das Reizen zu simulieren, jedoch wurde hier nur einer getestet. Daher soll hier geklärt werden, ob der k-nearest-Neighbour Algorithmus bereits der beste Algorithmus für einen Skatcomputer ist oder ob andere Algorithmen eine bessere Leistung zeigen können. Es werden drei Algorithmen miteinander verglichen und so versucht, einen möglichst optimalen Skatcomputer zu programmieren.

Dazu werden im Rahmen dieser Arbeit drei Implementationen von Skatcomputern gegeneinander antreten und dabei soll bewertet werden, welcher der drei Computer die meisten Spiele gewinnt, aber auch wie nah die Reizungen der Computer an die von menschlichen Spielern kommen.

In Kapitel zwei werden daher zunächst die Skatregeln erklärt. Kapitel drei fasst einige Grundlagen der künstlichen Intelligenz und des maschinellen Lernens zusammen. Kapitel vier skizziert den Aufbau des genutzten Programms. In Kapitel fünf werden die Algorithmen, anhand derer die Computer ihren Reizwert bestimmen, erläutert und deren Umsetzung erklärt. In Kapitel 6 folgt eine Auswertung der gesammelten Daten und in Kapitel sieben folgt eine Zusammenfassung der Arbeit.

2. Skat

Skat ist ein Strategiespiel mit imperfekter Information, da jeder Spieler nur seine eigenen Handkarten kennt und Glücksspielelemente vertreten sind. Es wird mit 3 Spielern gespielt. In jeder Runde spielt ein Spieler gegen die verbleibenden zwei, die dann als Gegenpartei bezeichnet werden.

2.1 Geschichte

Die Anfangszeit des Skatspiels, vor 1810, ist kaum belegbar und besteht im Wesentlichen aus zwei Auffassungen, die Thüringische Auffassung und die Saarländische Theorie. Anerkannt ist jedoch, dass das Skatspiel aus mehreren älteren Spielen hervorgegangen ist {vgl. Wolfgang Rui 2010 #5}. Das erste dokumentierte Spiel fand am 4. September 1813 statt und wurde von Gymnasialprofessor Johann Friedrich Ludwig Hempel (1773–1849), Medizinalrat Dr. Hans Carl Leopold Schuderoff, Hofadvokat und Notar Friedrich Ferdinand Hempel (1778–1836), Ratsherr Carl Christian Adam Neefe (1774–1821) und Kanzler Hans Carl Leopold von der Gabelentz (1778–1831) gespielt. Die Abrechnungen dieses Spiels sind bis heute erhalten {vgl. Deutscher Skatverband e. V. 2019 #6}.

1886 entwickelte sich ein Skatturnier in Altenburg zum ersten Skatkongress mit ca. 1000 Teilnehmern. Hier wurde zum ersten Mal eine allgemeine Skatordnung beschlossen und die Gründung eines Verbandes diskutiert. Die Gründung scheiterte, da man sich nicht entscheiden konnte, ob das Zahlen- oder das Farbreizen als Standard etabliert werden sollte. Am 12.03.1899, auf dem dritten Skatkongress wurden diese Differenzen dann doch beigelegt und der deutsche Skatverband gegründet. Das Farbreizen wurde als offizielle Spielart gewählt, während das Zahlenreizen als mögliche Option als Regelalternative aufgenommen wurde. In den Jahren 1927 und 1928 wurden, nach einer längeren Pause auf Grund des Krieges, der elfte und zwölfte Skatkongress durchgeführt. Hier wurde nun das Zahlenreizen als einzige offizielle Spielart festgelegt und die „neue deutsche Skatordnung“ verabschiedet. Diese bildet den Grundstein des Einheitsskats. Außerdem wurde das deutsche Skatgericht als oberste Entscheidungsgremium im Skatsport etabliert {vgl. Deutscher Skatverband e. V. 2019 #6}.

1976 wurde die International Skat Players Association (ISPA) gegründet. Auf Grund von Differenzen bei der Regelauslegung, kam es nicht zu einer Zusammenarbeit mit dem deutschen Skatbund. Dies änderte sich erst 1998 und eine Internationale Skatordnung wurde beschlossen. Außerdem bildeten beide Verbände im Jahre 2001 gemeinsam das internationale Skatgericht, das das deutsche Skatgericht als oberstes Entscheidungsgremium ablösen sollte {vgl. Deutscher Skatverband e. V. 2019 #6}.

2016 wurde Skatspielen als immaterielles Kulturerbe anerkannt und in das Bundesweite Verzeichnis des immateriellen Kulturerbes aufgenommen {vgl. Unesco 2016 #4}.

2.2 Die Regeln

Zu Beginn des Skatspiels werden die 32 Karten gemischt und jeder Spieler erhält 10 Handkarten. Die verbleibenden 2 Karten werden verdeckt in die Mitte des Tisches gelegt und bilden den namensgebenden Skat. Der Spieler links vom Geber spielt Vorhand, der Geber selbst spielt Hinterhand und der verbleibende Spieler spielt Mittelhand. Das folgende Spiel gliedert sich dann in zwei Phasen, das Reizen und das Stichspiel {vgl. Matthias Bock 2003 #3 S6.}.

2.2.1 Das Reizen

Beim Reizen wird der Alleinspieler bestimmt. Dazu bestimmt jeder Spieler den Wert des Spiels, den er gerne als Alleinspieler spielen würde. Dieser setzt sich zusammen aus dem Grundwert des jeweiligen Spiels und einem Faktor der unter anderem aus dem Vorhandensein oder der Abwesenheit von Buben ermittelt wird. Die verschiedenen Spiele und ihr Grundwert sind: Karo ist Trumpf mit dem Grundwert 9, Herz ist Trumpf mit dem Grundwert 10, Pik ist Trumpf mit dem Grundwert 11, Kreuz ist Trumpf mit dem Grundwert 12 oder ein Spiel, bei dem nur die Buben Trumpf sind, was Grand genannt wird, mit Grundwert 24. Außerdem gibt es noch die Möglichkeit, ein Nullspiel anzusagen, in diesem Fall gilt ein feststehender Wert von 23{International Skat Players Association e.V., Deutscher Skatverband e.V. 2010 #1 S.24}.

Als nächstes wird der Faktor ermittelt, mit dem der Grundwert multipliziert wird, um den Spielwert zu erhalten. Dazu werden die Buben in absteigender Reihenfolge, Kreuz, Pik, Herz und Karo betrachtet und ermittelt wie viele, beginnend beim Kreuz Buben, fehlen oder auf der Hand sind {vgl. Matthias Bock 2003 #3 S.11ff}.

Dies liefert eine Zahl zwischen eins und vier, zu der dann grundsätzlich noch eins addiert wird. Zusätzlich kann ein Spieler sich entscheiden, verschiedene zusätzliche Ansagen zu machen, um für jede noch einmal plus eins auf den Vorfaktor zu rechnen. Zu diesen Ansagen zählen: "Hand", was bedeutet, dass der Spieler den Skat nicht aufnimmt, "overt" bedeutet, dass der Spieler mit offener Hand spielt, "Schneider", der Spieler behauptet die Gegenpartei bekommt keine 30 Punkte und "Schwarz", was bedeutet, dass der Spieler behauptet, dass die Gegenpartei nicht einen Stich gewinnt {International Skat Players Association e.V., Deutscher Skatverband e.V. 2010 #1 S.24}.

Falls eine der Ansagen während des Spiels verletzt wird, hat der Alleinspieler ungeachtet seiner Punkte verloren. Abschließend bestimmt nun jeder den Wert seines Spiels, in dem er die beiden Zahlen, den Faktor und den Grundwert multipliziert.

Nun beginnt das eigentliche Reizen. Der Spieler, der Mittelhand spielt, sagt dem Spieler, der Vorhand spielt, in aufsteigender Reihenfolge alle möglichen Spielwerte an, bis dieser einen Wert nicht mit ja beantwortet. Müsste der Spieler, der Mittelhand spielt, dabei einen Wert nennen, der den von ihm ermittelten Spielwert übersteigt, kann er stattdessen auch passen. Nach dem einer der beiden Spieler das Reizen beendet hat, übernimmt der Rückhandspieler die Aufgabe des Ausgeschiedenen, also nennt entweder in aufsteigender Reihenfolge die Spielwerte oder beantwortet diese mit ja. Auch hier haben beide Spieler die Möglichkeit, falls der genannte Spielwert den ermittelten übersteigt, zu passen. Wenn alle Spieler passen, gilt das Spiel als eingepasst und wird mit null Punkten für jeden notiert {vgl. Matthias Bock 2003 #3 s.8}.

Der letzte Spieler, der nicht gepasst hat, wird nun der Alleinspieler und darf den Skat seiner Hand hinzufügen. Bevor das Spiel nun beginnt, muss der Alleinspieler noch zwei Karten aus seiner Hand verdeckt vor sich ablegen, was als drücken bezeichnet wird. Diese Karten werden später seinen Punkten hinzugefügt.

2.2.2 Das Spiel

Zu Beginn sagt der Alleinspieler an, welches Spiel er spielen möchte. Dabei muss er darauf achten, dass er ein Spiel mit einem höheren Spielwert ansagt, als den Wert, den er gereizt hat. Das angesagte Spiel bestimmt dann auch die Wertigkeiten, um zu ermitteln, welche Karte einen Stich gewinnt. Dabei gilt, die zuerst aufgespielte Karte gewinnt, es sei denn, jemand legt eine höhere Karte derselben Farbe oder einen Trumpf. Hat ein Spieler einen Trumpf aufgespielt, gewinnt er, solange kein anderer Spieler einen höheren Trumpf legt. Folgende Fälle ausgenommen, wird die Trumpffarbe mit dem angesagten Spiel bestimmt: Die Buben sind in jedem Spiel, mit Ausnahme des Nullspiels, die höchsten Trümpfe. Beim Grand sind nur die Buben Trumpf. Beim Nullspiel gibt es keinen Trumpf und die normale Kartenreihenfolge, Sieben-Acht-Neun-Dame-König-Zehn-Ass, wird zu Sieben-Acht-Neun-Zehn-Dame-König-Ass geändert {vgl. Matthias Bock 2003 #3 S.5}.

Anschließend spielt der Vorhandspieler die erste Karte auf. Nun legen reihum die anderen beiden Spieler eine Karte in die Mitte und dann wird ermittelt, wer den Stich gewonnen hat. Dieser Spieler legt der Stich dann verdeckt vor sich ab und spielt die nächste Karte auf. Dies wird so oft wiederholt, bis kein Spieler mehr Karten auf der Hand hat. Dabei muss bei jedem Stich bedient werden. „Es muss grundsätzlich bedient werden. Wer nicht bedienen kann, darf irgendeine Karte abwerfen oder trumpfen.“ {Matthias Bock 2003 #3 S.9}

Zum Schluss zählen die Spieler ihre Punkte. Hat der Alleinspieler mehr Punkte als die Gegenpartei, hat er das Spiel gewonnen und bekommt Punkte in Höhe des tatsächlichen Spielwertes, nicht in Höhe des gereizten Wertes. Hat die Gegenpartei mehr Punkte, als der Alleinspieler, erhält der Alleinspieler Minuspunkte in Höhe des doppelten Spielwerts. Schneider und Schwarz können auch ohne Ansage bei der Abrechnung zum Tragen kommen {vgl. Matthias Bock 2003 #3 S.24}.

3 Künstliche Intelligenz

„Die künstliche Intelligenz ist ein Teilgebiet der Informatik, welches sich mit der Erforschung von Mechanismen des intelligenten, menschlichen Verhaltens befasst (Intelligenz). Dieses geschieht durch Simulation mit Hilfe künstlicher Artefakte, gewöhnlich mit Computerprogrammen auf einer Rechenmaschine“ {Andreas Wichert 2014 #7}. Es mangelt allerdings an einer klaren Definition von Intelligenz, da es zwar viele Modelle für Intelligenz, wie etwa Gardners Intelligenzdimensionen, und sogar eine Möglichkeit gibt, Intelligenz zu messen, mittels des Intelligenzquotienten, diese aber alle nicht verbindlich oder allgemein akzeptiert sind {vgl. Albert Ziegler, Kurt A. Heller 2014 #8}. Daher scheint es fast sinnvoller, das Ziel dieses Teilgebietes, eine Technologie, zu schaffen, die den Turingtest bestehen kann und die mannigfaltigen interdisziplinären Aspekte zu betrachten. So sind neben der Informatik auch die Mathematik, Philosophie, Psychologie, Ingenieurswissenschaften, Sprachwissensschafen und viele weitere an Projekten zur Schaffung einer künstlichen Intelligenz beteiligt. {vgl. Ahrweiler 1995 #9 S.45 ff}

3.1 Geschichte der künstlichen Intelligenz

Um 1960 herum begannen in Deutschland erste Forscher sich mit dem Teilgebiet künstliche Intelligenz zu befassen. Dr.-Ing. Karl Steinbruch begann an der technischen Hochschule Kaiserslautern 1963 mit Projekten zur Sprachverarbeitung, während Gerd Veenker ab 1972 an der Universität Bonn die institutionelle Entwicklung der deutschen KI-Forschung initialisierte. Das erste Treffen von Wissenschaftlern, die sich für die KI-Forschung begeistern konnten, fand 1969 statt {vgl. Ahrweiler 1995 #9 S.60}

.Bis zum nächsten Treffen dauerte es dann allerdings sechs Jahre, da es zu Streitigkeiten zwischen der sich institutionalisierenden Informatik und Wissenschaftlern, die im Teilgebiet der KI forschen wollten, kam. Ein wichtiger Standpunkt dieser Streitigkeiten wurde durch den „Lighthill-Report“ von 1973 begründet {vgl. Ahrweiler 1995 #9 S.62}.

Allerdings rief am 18.02.1975 Gerd Veenker zum ersten KI-Treffen in Bonn auf. Auf diesem Treffen wurde unter anderem beschlossen, dass es in Zukunft einen Rundbrief geben soll, der gesammelt auch als Zeitschrift „KI“ erscheinen soll. Der erste dieser Rundbriefe, vom 26.5.1975, gilt als erstes Schriftstück, das die deutsche KI-Forschung dokumentiert. Zudem wurde beschlossen, dass die Gründung einer Fachgruppe „künstliche Intelligenz“ in der Gesellschaft für Informatik angeregt werden solle, was noch im selben Jahr angenommen wurde {vgl. Ahrweiler 1995 #9 S.75f}.

1980 wurde auf der 10. Jahrestagung der Gesellschaft für Informatik, durch einen Vortrag des Edward A. Feigenbaum über Expertensysteme und Knowledge Engineering, das Interesse Fördergelder aus der Wirtschaft für die KI-Forschung zu akquirieren, geweckt. In Folge dessen wurden zusätzliche Veranstaltungen um das Thema KI etabliert, unter anderem die „European Conference on Artificial Intelligence“, bei der bereits zwölf Prozent der Anwesenden aus der Industrie kamen {vgl. Ahrweiler 1995 #9 S. 105Ff}.

3.2 Wissenschaftliche Grundlagen

„KI [ist] etwas hochinteressantes, ein großes und nicht mal im Ansatz abgeschlossenes Feld, wo man noch viel Forschen kann und das uns womöglich unsere anderen Computerprojekte deutlich verbessert“ {Breuer 2012 #10 S.4}.

3.2.1 Die Zwei Klassen von KI

Grundsätzlich werden künstliche Intelligenzen in zwei Klassen eingeteilt, starke und schwache.

Starke KI‘s sind das, was allgemein unter künstlicher Intelligenz verstanden und über Medien verbreitet wird. Bei einer starken KI handelt es sich um eine Intelligenz, die ein eigenes Bewusstsein entwickelt hat und somit in der Lage ist, Probleme eigenständig und kreativ zu lösen. Dabei ist nicht definiert, ob diese Form der Intelligenz der unseren ähneln wird, also ob sie in der Lage ist, Emotionen zu empfinden oder andere Denkstrukturen mit uns teilt {vgl. Breuer 2012 #10 S.5}.

Mit dem 1950 von Allan Turing formulierten „Imitation Game“, das heute als Turingtest bekannt ist, ist sogar bereits eine Möglichkeit gegeben, eine starke KI zu testen. Dabei stellt ein Mensch, ohne Sicht- oder Hörkontakt, Fragen an zwei ihm unbekannte Gesprächspartner. Einer dieser Gesprächspartner ist ein Mensch, der andere eine Maschine. Ist der Fragensteller am Ende des Tests nicht in der Lage, korrekt anzugeben, welcher der Gesprächspartner menschlich ist, gilt der Test als bestanden {vgl. Epstein 2009 #12 S24.f}.

Allerdings ist auf diesem Forschungsgebiet seit längerem kein Fortschritt erzielt worden, auch wenn bereits Theorien aufgestellt wurden, laut denen die KI an einen Punkt gelangen wird, ab dem sie in der Lage sein wird, sich eigenständig weiterzuentwickeln (Technologische Singularität).

Schwache KI‘s hingegen sind Programme, die lediglich intelligent wirken, was unter anderem mittels Algorithmen, wie Beispielsweise dem maschinellen Lernen (siehe. 3.2.2), erzeugt wird. Außerdem werden schwache KI‘s lediglich im Hinblick auf ein bestimmtes Problem, für dessen Lösung nach menschlichem Empfinden Intelligenz nötig ist, hin entwickelt und müssen nicht mehr können als dieses spezifische Problem zu lösen. Die Forschung auf dem Gebiet der schwachen KI hat in den letzten Jahren große Fortschritte gemacht {vgl. Breuer 2012 #10 S.5f} und ist seit 1980 in der Wirtschaft als Expertensystem sehr gefragt (siehe 3.1). Konkrete Beispiele für schwache KI‘s sind so genannte NPC (Non-Player Characters), Gegner bei Computerspielen, die zwar für den Benutzer intelligent scheinen, deren Verhalten jedoch von Algorithmen bestimmt wird.

Diese Arbeit wird sich ausschließlich mit den schwachen KI‘s befassen.

3.2.2 Maschinelles Lernen

„Um von Intelligenz sprechen zu können, muss ein System in einer sich verändernden Umwelt in der Lage sein zu lernen“ {Alpaydın 2008 #13 S.2}. Daher zeichnet sich das maschinelle Lernen als eines der wichtigsten Teilgebiete der künstlichen Intelligenz aus.

Dabei ist ein Ziel des maschinellen Lernens, durch Training oder Erfahrung ein gegebenes Problem stetig besser lösen zu können. Dazu definieren wir ein Modell des Problems, das durch mehrere Parameter bestimmt ist und nutzen Methoden der Statistik und Stochastik, um diese dann zu optimieren. Schlussendlich soll das System dann in der Lage sein, ausgehend von einer Stichprobe und den gelernten Daten Schlussfolgerungen zu ziehen. {vgl. Alpaydın 2008 #13 S.3}.

Dabei kann der Lernprozess in zwei unterschiedlichen Arten ablaufen, unüberwacht oder überwacht.

Beim unüberwachten Lernen liegen nur Eingabewerte, aber keine Ausgabewerte vor. Das System versucht dann, Muster und Regelmäßigkeiten in den Eingabewerten zu finden und sie anhand dieser Muster zu klassifizieren. Dies wird als Dichteschätzung bezeichnet. Eine oft verwendete Dichteschätzung ist die Clusteranalyse. Dabei werden die Eingabewerte nach Häufungspunkten oder nach gemeinsamen Eigenschaften abgesucht und nach diesen gruppiert.

Beim überwachten Lernen sind sowohl Eingabe-, als auch Ausgabewerte bekannt. Ziel ist es hier eine Funktion zu schaffen, mit der für neue Eingabewerte, mit unbekanntem Ausgabewert, ein solcher geschätzt werden kann. Eine Form des überwachten Lernens ist das bestärkende Lernen. Hierbei wird die vom System getroffene Entscheidung, vom System selbst oder von außen auf seine Qualität geprüft und bewertet. Auf dieser Grundlage soll gelernt werden, welche Entscheidungen zutreffender sind und welche verworfen werden sollten {vgl. Fu 1994 #14}.

3.3 Aktuelle KI in Computerspielen

Ein oft genanntes Beispiel für die Anwendungsmöglichkeiten von KI‘s sind Spiele, die zugleich eines der wichtigsten Forschungsfelder für KI und maschinelles Lernen sind. Spiele sind auf Grund der Einfachheit ihrer Regeln leicht in ein Modell zu übertragen, bieten aber auf Grund der Vielzahl an möglichen Zügen eine Komplexität, an der sich viele KI‘s zu messen versuchen {vgl. Alpaydın 2008 #13 S. 13}.

Als einer der ersten großen Erfolge auf diesem Gebiet ist der Sieg in der ersten Partie eines fünf ründigen Wettkampfes des Schachcomputers DeepBlue am 10. Februar 1996 über den damaligen Schachweltmeister Garri Kasparow zu nennen. Es war das erste Mal, dass ein Schachweltmeister gegen einen Computer verloren hatte. Darauf folgten noch viele weitere Schachcomputer die immer neue Rekorde und bessere Gegner besiegten. Aktuellere Projekte beschäftigen sich allerdings auch mit dem Spiel Go, bei dem es noch um ein Vielfaches mehr Möglichkeiten als beim Schach gibt. So spielten am 11. März 2016 ein Go-Computer, Alphago, gegen den besten Go-Spieler Lee Sedol. Auch dieses Spiel gewann die KI {vgl. Johannes Fischer 2016 #15}.

Aber nicht nur als Gegner, die uns um jeden Preis schlagen wollen, sondern auch als so genannte NPC sind KI‘s von Interesse. So versuchen Spiele wie „Hello Neighbor" und "Echo", den Spieler mit Gegnern zu konfrontieren, die sich an die Strategien des Spielers anpassen. Hierbei steht allerdings mehr der Spielspaß des Spielers, als das perfekte Spiel der KI im Vordergrund. Allerdings ist auch hier ein Anwendungsfeld für künstliche Intelligenz, das ständig weiterentwickelt werden wird, auch wenn es noch am Anfang steht.

4 Das Programm

Das Programm, das verwendet wird, besteht aus zwei Teilen. Zum einen aus verschiedenen Klassen, die ein Skatspiel für drei Spieler simulieren sollen und zum anderen aus der abstrakten Klasse Player und den Klassen, die Player extenden (siehe 5). Zusätzlich dazu wurden noch drei weitere Klassen, zur Verwaltung der Datenbank und zur Erhebung der Trainingsdaten, verwendet, die aber für die eigentliche Skatsimulation keine Rolle spielen.

4.1 Das Spiel

Das Skatspiel wird von insgesamt vier Klassen simuliert. Von diesen stellen zwei Klassen das Spielmaterial dar, also Karten und das Skatblatt mit 32 verschiedenen Karten. Eine Klasse enthält grundlegende Informationen und Punktberechnungen für das Spiel und eine letzte Klasse simuliert den Spielablauf.

4.1.1 Card

Die Klasse Card simuliert einzelne Spielkarten und deren Informationen, die für das Spiel relevant sind. Dazu klassifizieren die beiden Attribute Color und Number jede Karte eindeutig. Zur besseren Verarbeitung werden sowohl die Farbe (Kreuz, Pik, Herz, Karo) der Karte, als auch die Werte (Bube, Ass, 10, König, Dame, 9, 8, 7) der Karte in natürliche Zahlen übersetzt. Dazu wurden die Farben, entsprechend ihrer Reihenfolge beim Skat, mit den Zahlen von eins bis vier durchnummeriert. Die Werte wurden gemäß der Reihenfolge beim Nullspiel, beginnend mit der Sieben, durchnummeriert(siehe 2.2.2). Höheren Karten wurden dann fortlaufende Nummern zugeordnet. Das Attribut pointvalue wurde zur Vereinfachung einiger Methoden der Klasse Player mit aufgenommen und wird im Konstruktor der Klasse berechnet. Die Methode toString dient dann der Rückkonvertierung der Werte Color und Number in die geläufigen Bezeichnungen der Karten.

Die Methode handSort soll die Handkarten der Spieler dem gerade laufenden Spiel entsprechend sortieren. Dies soll zum einen die Übersichtlichkeit für menschliche Spieler fördern, aber zum anderen auch die Verarbeitung der LinkedList hand leichter machen, da eine Sortiertheit der Liste, bei Bedarf vorausgesetzt werden kann. Eine sortierte Hand besteht von links nach rechts, zuerst aus den Buben, in der absteigenden Skatreihenfolge, dann den Trumpfkarten und dann den restlichen Farbkarten, ebenfalls in absteigender Reihenfolge der Farben und Werte.

Die Methode verwendet einen Bubblesort-Algorithmus, da dieser auf kleineren Listen eine gute Laufzeit erzielt (O(n) = n2 im Speziellen schlechtesten Fall dann O(12) = 144). In diesem Fall kann die Hand eines Spielers maximal zwölf Karten beinhalten, von denen keine doppelt ist. Der Vergleich für die handSort-Methode ist in der Methode compareTo ausgelagert. Diese Methode vergleicht Karten nach ihrer Wertigkeit im Skat, mit der Abwandlung, dass die Farben nach der Reihenfolge der Buben sortiert werden. Die Methode berücksichtigt den aktuellen Trumpf, oder greift auf einen default-Wert zurück, nämlich Kreuz.

4.1.2 Game

Game ist die Klasse, die den Spielablauf simuliert. Dazu sind die verschiedenen Phasen des Spiels jeweils als eigene Methoden implementiert. Zusätzlich sind noch einige Helferfunktionen implementiert. Die Attribute der Klasse gliedern sich in zwei Gruppen. Als Erstes sind einige Attribute zur Verwaltung des Spielablaufs deklariert. So sind in Member alle Spieler gespeichert, die am Spiel teilnehmen. In rounds wird die gewünschte Anzahl Runden gespeichert, die ein Spiel dauern soll, in base ist eine Instanz der Klasse Database gespeichert, die Zugriff auf die Datenbank erlaubt. HandsPlayed speichert alle Handkarten, mit denen der Einzelspieler gespielt hat und in score werden die Ergebnisse jeder Runde und das Endergebnis gespeichert. Die andere Gruppe speichert Daten über die aktuelle Spielrunde, die methodenübergreifend genutzt werden. Dazu werden in den Attributen actualRound, trump, player, begin und playValue Informationen, wie die aktuelle Runde, der momentane Trumpf und Alleinspieler, der Spieler der Vorhand sitzt und der gereizte Wert der laufenden Runde gespeichert.

Der Konstruktor stellt den normalen Ablauf des Spiels dar und ruft nacheinander die Methoden auf, die die einzelnen Spielabschnitte simulieren. Zu Beginn wird die Methode setup aufgerufen. Mit dieser Methode werden die drei Spieler in Member eingefügt und deck, handPlayed und score werden instanziiert.

Anschließend wird mittels einer While-Schleife der Ablauf des Spiels bis zur gesetzten Anzahl an Runden wiederholt. Dieser Ablauf setzt sich aus den Methoden giveCards, teasing, play und evaluateGame zusammen.

Dabei ist giveCards ein Verweis auf die Methode giveCards der Stackklasse, die für jeden Spieler einmal ausgeführt wird und das Kartengeben simuliert. Die Methode teasing zeigt mittels der Methoden hear und say der Playerklasse den Spielern die verschiedenen möglichen Reizwerte, die aus der Klasse Info entnommen werden. Nachdem mindestens zwei Methoden false zurückgegeben haben, wird der Wert von Player auf die Position des Alleinspielers im Array Member gesetzt, ihm werden mittels der Methode call der Skat übergeben, der mittels der Methode getSkat der Klasse Stack generiert wurde und es werden weitere Variablen für die Spieler instanziiert. Wenn alle Spieler gepasst haben, wird Player mit -1 belegt und damit die aktuelle Runde beendet.

Anschließend wird die Methode play aufgerufen. Hier wird in einer do-while-Schleife, mit der Methode playCard der Playerklasse, ein Array der Länge drei befüllt und anschließend mit der Methode checkWinner ausgewertet. Dieses Array simuliert dabei einen Stich, in dem die Spieler nacheinander ihre Karten legen. Die Methode checkWinner ermittelt mittels aussagenlogischer Funktionen die Stelle der Karte im Array, die den Stich gewonnen hat. Anschließend wird mit den Methoden getCards und setRemain der Playerklasse die Informationen des Stichs an die einzelnen Spieler vermittelt. Diese Schleife wird zehnmal durchlaufen.

Nach dem Aufruf von play wird mittels der Methode evaluateGame der Punktgewinn oder -verlust des Alleinspielers bestimmt. Dazu werden mittels der Methode countPoints der Playerklasse seine Punkte bestimmt. Dieser Wert wird im Falle eines Nullspiels mit null verglichen und so direkt zu 23 Punkten für ein gewonnenes Spiel oder -46 Punkten für ein verlorenes Spiel ausgewertet. Im Falle eines Trumpfspiels oder eines Grand wird im Folgenden auf der Basis der Handkarten des Einzelspielers und dem Skat der Modifikator des Spiels bestimmt (siehe 2.2.1).

Dabei wird der Modifikator durch die Buben durch die Methode calcMod der Info Klasse ermittelt. Boni, wie Schneider und Schwarz, werden durch If-Abfragen hinzugefügt. Aus Gründen der Vereinfachung wurde auf Handspiel und offene Spiele verzichtet. Der finale Modifikator wird mit dem Grundwert des gespielten Spiels, der durch die Methode convertTrump der Klasse Info ermittelt wird, multipliziert und mit dem gereizten Wert verglichen. Anschließen erhält der Einzelspieler entsprechende Punkte. Die Ausgangshand des Alleinspielers und das Ergebnis des Spiels werden durch die Methode updateDB in die Datenbank übernommen.

Nach dem Durchlaufen der While-Schleife, die die gespielten Runden zählt, wird mittels einer for-Schleife das Endergebnis eines jeden Spielers bestimmt und mit der Methode scoreDB in die Datenbank übernommen.

4.1.3 Info

In der Klasse Info sind Methoden und Informationen gespeichert, welche die Auswertung von gespielten Runden vereinfachen sollen. Dazu befindet sich in Info ein Array mit allen möglichen Reizwerten, die von der Klasse Game in der Methode teasing verwendet wird.

Die Methode convertTrump ist eine Aneinanderreihung von If-Abfragen, um den vereinfachten, gespeicherten Werten für den aktuellen Trumpf, den Grundwert des entsprechenden Spiels, zuzuordnen. Dabei wird Null nicht berücksichtigt. Die Methode calcMod berechnet den Modifikator eines Spiels, der nur auf Grund der Anwesenheit oder Abwesenheit von Buben resultiert. Dazu beginnt die Methode mit dem kleinstmöglichen Modifikator von zwei und erhöht diesen mittels einer For-Schleife für jede passende Karte um eins.

4.1.4 Stack

Stack simuliert einen Kartenstapel. Durch den Konstruktor der Klasse wird das Attribut sackCards, einem Array der Länge 32, instanziiert und mit Instanzen der Klasse Card gefüllt. Für stackCards eignet sich besonders gut ein Array, da so die Größe des Kartenstapels gesichert ist und sichergestellt ist, dass die Karten im Stack vollständig sind.

Mit der Methode shuffle soll ein Mischen der Karten simuliert werden. Dazu werden in einer zufälligen Anzahl Durchgängen, jedes Element des Arrays mit einem Anderen, zufällig gewählten Element, getauscht.

Die Methoden getHand und getSkat unterteilen das Array in vier Bereiche. Die ersten drei Bereiche umfassen je zehn Karten und stellen die Handkarten der Spieler da. Die letzten beiden Karten werden als Skat deklariert.

5. Computergegner

Die Computergegner, die im Rahmen dieser Arbeit entwickelt wurden, sind alle als Kindsklassen der abstrakten Klasse Player entwickelt und implementieren, bis auf den Fall des menschlichen Spielers, nur die Methoden giveCards und call. Also ist der einzige Unterschied zwischen den verschiedenen NPCs, dass sie einen unterschiedlichen Algorithmus zum Reizen nutzen. Der Algorithmus, der bestimmt welche Karte gespielt wird, ist bei allen gleich. Dies ist in der Tatsache begründet, dass es in den Skatratgebern in nur wenigen bis gar keinen Seiten um das Spielen, sondern immer nur um das richtige Reizen geht.

5.1 Player/ DataBaseKI

Player ist eine abstrakte Klasse und sammelt alle Methoden und Attribute, die alle KI‘s gemeinsam haben. Darunter fallen Handkarten, die noch im Spiel befindlichen Karten und die Karten, die ein Spieler gewonnen hat. DataBaseKI ist eine abstrakte Klasse, mit der die beiden lernenden KI‘s auf die Datenbank zugreifen. Hier sind lediglich Zugangsdaten gespeichert.

Neben Methoden, die nur verwaltende Zwecke erfüllen, ist auch das Ausspielen von Karten in der Klasse Player implementiert. Damit bei der Interpretation von Daten der Spielverlauf transparenter ist und für alle drei KI‘s dieselben Bedingungen angenommen werden können, wurde sich dagegen entschieden, einen existierenden Algorithmus zu verwenden. Der hierfür entwickelte Algorithmus versucht grundsätzlich, in jedem Zug die Punktzahl, die die Fraktion des legenden Spielers erzielen kann, zu maximieren. Dazu wird jede noch mögliche Kombination an Karten für den Stich ausgewertet und aufaddiert, wobei verlorene Stiche negativ gewertet werden. Allerdings warf diese Implementation noch einige Schwächen auf.

Zum ersten fiel auf, dass auch die Gegenpartei versuchte, Stiche mit Trumpfkarten zu eröffnen, da diese am wenigsten Stiche verlieren würde. Allerdings sollte es beim Skat das Ziel der Gegenpartei sein, dem Alleinspieler möglichst viele Trümpfe abzunehmen, ohne dabei selbst Trümpfe zu spielen. Daher ist das Aufspielen eines Trumpfes für die Gegenpartei nur in sehr seltenen Fällen eine gute Strategie, da der andere Teamspieler Trumpf bekennen muss. Hier hilft allerdings eine Faustregel für Skatspieler weiter: Dem Freunde kurz, dem Feinde lang. Diese Regel besagt, dass wenn ein Spieler der Gegenpartei aufspielt, sein Aufspiel von der Person links von sich abhängt. Sitzt links sein Mitspieler, wird versucht, eine Farbe aufzuspielen die wenig auf der Hand vorhanden ist, in der Hoffnung, dass der Mitspieler möglichst viel dieser Farbe auf der Hand hat. Das ideale Ergebnis in diesem Fall wäre dann, dass der eine Spieler der Gegenpartei auf einer Farbe blank ist und so trumpfen oder schmieren kann und dass gleichzeitig der Alleinspieler einen Trumpf legen müsste, um das Spiel zu übernehmen. Falls links von dem Spieler, der gerade aufspielt, der Alleinspieler sitzt, gilt, dass eine Farbe ausgespielt werden soll, von der möglichst viel auf der Hand ist, um dasselbe Ergebnis zu erhalten {vgl. Wolfgang Rui 2013 #2 S.4}.

Das zweite Problem, das auffiel, war, dass der Alleinspieler oft versucht hat, Trumpfkarten aufzuspielen, obwohl ihm bekannt war, dass noch höhere Trümpfe im Spiel waren. An dieser Stelle sollte eigentlich davon ausgegangen werden, dass die Gegenpartei den Stich bekommt und somit die Punkte des hohen Trumpfs verloren sind. Gelöst wird dieses Problem, indem der Einzelspieler Trumpfkarten, die keine Buben sind und eine Punktzahl besitzen, nur dann aufspielt, wenn keine höheren Trumpfkarten mehr vorhanden sind.

Ein weiteres Problem, das beim Aufspielen des Einzelspielers auffiel, war, dass versucht wurde, zu Beginn der Partie mit Trumpf Ass oder Zehn zu eröffnen, da der Computer berechnet hatte, dass falls er den Stich gewinnt, so mehr Punkte bekommen wird, als wenn er einen Buben aufspielt. Ähnlich dem zuvor beschriebenem Problem werden so aber viele Punkte unnötig verschenkt. Daher werden für den aufspielenden Spieler nicht die potenziellen Punkte, sondern die potenziell gewonnenen Stiche berechnet. Bei einem Gleichstand, der anfangs dadurch aufgelöst wurde, dass die Karte ausgewählt wurde, die am weitesten links auf der Hand ist, wird nun noch der Punktwert der Karte von den gewonnenen Stichen abgezogen, was den Computer davon abhalten soll, wertvolle Karten aufzuspielen.

Das letzte unerwünschte Verhalten war, dass die Gegenpartei oft versuchte, eigene Buben auf bereits gewonnene Stiche zu legen, um mehr Punkte mit dem Stich gut zu machen. Das hier der Bube allerdings potenziell auch den nächsten Trumpfstich gewinnen könnte, wurde ignoriert. Um das zu verhindern, zählt der Bube dann bei der Kalkulation der potenziellen Punkte als minus fünf, falls man ihn verlieren könnte. Das macht ihn in den Augen des Computers wertvoller, als König und Dame und er wird verstärkt versuchen, die Stiche, in die er einen Buben reinwerfen kann, zu gewinnen.

Mit diesen Anpassungen ist der Algorithmus in der Lage, den Spielablauf eines Skatspiels, wie es unter Menschen vorkommen könnte, zu simulieren.

5.2 Naive KI

Die naive KI ist eine KI im Sinne klassischer Computerspiele. Ein Algorithmus versucht hierbei, ein menschliches Denken vorzutäuschen (siehe 3.2.1), obwohl alle Entscheidungen nach konkreten Kriterien getroffen werden. Dabei erfüllt diese KI im Wesentlichen den Zweck, den beiden lernenden KI‘s Input zu geben und triviale Spiele in die Datenbank mit aufzunehmen. Der Algorithmus, der zum Reizen und zum Abwerfen benutzt wird, basiert auf dem Konzept von J.P: Wergin, einem US-amerikanischen Skatspieler, der versuchte, das Reizen anhand von speziellen Kriterien bestimmbar zu machen und einigen Faustregeln, die Skatanfängern beigebracht werden, um Erfahrungen beim Reizen zu sammeln {vgl. Wolfgang Rui 2013 #2c S.37}.

Nach dem Ansatz von Wergin gibt es zwei Merkmale, die ein Skatblatt spielbar machen: die Anzahl der Buben und das Beiblatt. Damit werden zwar zwei wesentliche Faktoren genannt, jedoch gibt es hier viele Beispiele, die nach seinem Ansatz als spielbar gewertet werden, aber nur mit viel Glück gewonnen werden können und viele Beispiele, die zu Unrecht als nicht spielbar klassifiziert werden.

Daher wurde entschieden, weitere Faktoren mit einzubeziehen. Als weiteren Anhaltspunkt für ein spielbares oder nicht spielbares Blatt gilt noch „4 oder weniger Trümpfe sind für ein klassisches Farbspiel zu wenig“ {Wolfgang Rui 2013 #2 S. 38}. Daraus abgeleitet spielt also auch die Anzahl der Trümpfe eine wichtige Rolle für die Spielbarkeit eines Blattes. Zusammengefasst gilt für die Anzahl der Trümpfe, dass vier zu wenig sind, sechs Standard und sieben als stark. Als letztes Merkmal wurde sich noch für Freifarben, also Farben, von denen der Spieler keine Karte auf der Hand hat, entschieden, da dies das am meisten bemängelte Kriterium am Ansatz von Wergin ist {vgl. Wolfgang Rui 2013 #2 S. 37f}.

Daraus resultiert dann, nach der Gewichtung der Faktoren ein algorithmischer Ansatz für das Reizen, der für ein gegebenes Blatt aus zehn Karten einen Wert zwischen minus vier und plus fünf zuweist. Bei einem Wert größer als null wird das Blatt als spielbar klassifiziert.

Kategorie	Anzahl	Wert des Spiels
Bubenanzahl	0	- 1
	1	0
	> 1	+ 1
Trümpfe	< 5	- 1
	6 oder 7	+ 1
	> 7	+ 2
Beiblatt	< 0	-1
	0 oder 1	0
	> 1	+1
Freifarben	Von jeder Farbe min 2 Karten	-1
	Von jeder Farbe min. 1 Karte	0
	Min eine Freifarbe	+1

Dabei wird besonderer Wert auf die Anzahl der Buben gelegt, da bereits vier Buben jedes Blatt ansatzweise spielbar machen, indem Buben sowohl in der Kategorie Bubenanzahl als auch in der Kategorie Trümpfe mit einbezogen werden. Zudem liefert der Wert „Trümpfe“ das größtmögliche Plus der gesamten Tabelle, da ein Blatt mit sieben Trümpfen, egal wie diese positioniert sind, gewonnen werden sollte. Allerdings wäre nach diesem Ansatz ein weiteres plus eins nötig, um sieben Trümpfe in jedem Fall zu spielen {vgl. Dengel 2008 #16 S. 97}.

Die Kategorie Beiblatt ist hierbei lediglich eine Wertung von anderen Kriterien, die sich ebenfalls an Weigin orientieren. Dabei bringt jedes Ass und jede Zehn, bei der eine weitere Karte derselben Farbe auf der Hand ist und die nicht Trumpf sind, ein plus eins und jede zehn, ohne weitere Karte derselben Farbe, die ebenfalls nicht Trumpf sein darf, minus 1. So sollen Karten, die abseits von Trumpfkarten wahrscheinlich Stiche machen werden, in den Entscheidungsprozess mit einbezogen werden. Allerdings wird erst ein Wert, der größer ist als null, als nicht mehr hinderlich für das Spiel aufgefasst, was dazu führt, dass Spiele wegen schlechtem Beiblatt als nicht zu gewinnen eingestuft werden.

Bei der Kategorie Freifarbe wird die Spielstärke eines Blattes erhöht, falls der Spieler von einer Farbe frei ist. Dies würde dem Spieler die Möglichkeit bieten, Stiche mit dem Ass oder der Zehn diese Farbe zu trumpfen und so hohe Punkte zu erzielen. Allerdings soll in dieser Kategorie auch der so genannte Rollmops, von jeder Farbe mehr als zwei Karten auf der Hand haben, behandelt werden und die Spielstärke dieses Blattes deutlich reduziert werden. Dabei ist zu beachten, dass wenn der Spieler einen Rollmops auf der Hand hat, es keine Möglichkeiten gibt auf einen Wert größer null zu kommen, nach diesem Punktesystem. Da bei einem Rollmops bereits acht von zehn Karten betrachtet sind, kann in der Kategorie Trumpf maximal ein minus eins erreicht werden, in der Kategorie Bubenanzahl kann im besten Fall ein plus eins erreicht werden. Zusammen kommt das Blatt so auf einen Wert von minus eins, was durch die verbleibende Kategorie Beiblatt nicht ausgeglichen werden kann.

Diese Art von Reizen bezieht sich allerdings nur auf Farbspiele und da hier sichere Spiele, für die Datenbank, auch als sicher erkannt werden sollen, bietet es sich an, ein eher abgesichertes Reizverhalten zu simulieren. Daher wird diese KI auf Nullspiele verzichten. Grand hingegen wird mit einer vereinfachten Form des oben beschriebenen Systems bestimmt.

Dazu werden nur die Anzahl der Buben und das Beiblatt betrachtet. Die erste Voraussetzung, damit diese KI einen Grand versucht, ist, dass sie mindestens zwei Buben auf der Hand hat. Zwar ist es auch gut möglich, mit weniger Buben einen Grand zu gewinnen, allerdings sind das oft Erfahrungsspiele, die eine auf Sicherheit reizende KI nicht berücksichtigen sollte. Die zweite Bedingung ist, dass die Summe der Werte für Beiblatt, bevor diese mit der Tabelle abgeglichen wurden, größer sind als vierzehn. Dabei wurde sich für vierzehn entschieden, da bei der Berechnung des Werts für Beiblatt jedes Ass und jede zehn dreimal gezählt wird, nämlich für jede Farbe außer der eigenen einmal. Somit wird eine Hand, bei der die Summe der Beiblattwerte mindestens fünfzehn ist, mindestens fünf Asse oder Zehnen mit entsprechenden Karten dabei haben. Ausgehend davon, dass mit den zwei Buben, die der Einzelspieler auf der Hand hat, die Buben der Mitspieler gezogen werden können, sind diese dann die höchsten Karten und werden sicher Stiche machen. Damit hat der Alleinspieler mindestens 58 Punkte und ist somit in fast jedem Fall der Gewinner (siehe 2.2.2).

Beim Abwerfen wird versucht, die Kategorie Freifarbe zu optimieren. Dabei ist die oberste Priorität, dass der Computer versucht, eine Farbe zu finden, die er abwerfen kann, um danach in dieser Farbe frei zu sein. Dabei bevorzugt er es, die Farbe mit den niedrigeren Punktwerten abzuwerfen, da zu erwarten ist, dass es mit diesen unwahrscheinlicher ist, einen Stich zu machen. Gibt es eine Farbe, von der nur eine Karte auf der Hand ist, so wird der Computer diese Farbkarte und die Karte, die am weitesten links ist auf der Hand, abwerfen, da es sich hierbei mit hoher Wahrscheinlichkeit um eine niedrige Karte handelt.

5.3 KNearest

Die erste der beiden lernenden KI‘s verwendet einen k-Nächste Nachbarn-Schätzer, um die Spielbarkeit eines Blattes zu bewerten. Ein ähnlicher Ansatz wurde bereits von Thomas Keller und Sebastian Kupferschmid diskutiert, allerdings werden für diese Arbeit die Ansätze in einigen Punkten anders gewählt sein, was dann in der Evaluation eine Vergleichsmöglichkeit liefern wird {vgl. Dengel 2008 #16 S.97ff}.

5.3.1 k-Nächste-Nachtbarn Algorithmus

Zu Beginn soll die Funktionsweise eines k-Nächste-Nachtbarn Algorithmus erläutert werden. Bei diesem Algorithmus handelt es sich um eine Art der nichtparametrischen Datenschätzung. Lernalgorithmen dieses Typs versuchen unbekannte Daten zu schätzen, indem sie mit bekannten Daten, die als den unbekannten Daten nahe ermittelt sind, verglichen werden. Dabei werden im Gegensatz zu anderen Lernalgorithmen dieser Art kein festes Intervall untersucht, sondern die k-nächsten Nachbarn. Das eigentliche Lernen ist dabei das Abspeichern neuer Daten, was als „lazy learning“ bezeichnet wird {vgl. Alpaydın 2008 #13 S. 167f}.

Um den Algorithmus anwenden zu können, muss eine Metrik auf den Daten definiert werden, um ermitteln zu können, welche bekannten Daten näher sind und damit betrachtet werden sollten. Zudem muss entschieden werden, wie viele Nachbarn für die Evaluation betrachtet werden sollen. Dabei ist wichtig, dass wenn k zu klein gewählt ist, die Ergebnisse durch besondere Hände, die von einer normalen Spielweise abweichen, die Entscheidung verfälschen. Bei einem zu groß gewählten k allerdings könnten zu weit entfernte Daten mit berücksichtigt werden und solange keine gewichtete Metrik benutzt wird, das Ergebnis verfälschen. Zudem ist es bei dieser Klassifikation irrelevant, ob das k gerade oder ungerade gewählt ist, da es potenziell sieben verschiedene Klassen gibt, denen eine Hand zugeordnet werden kann {vgl. Alpaydın 2008 #13 S. 167f.}.

Um für eine gegebene Hand einen spielbaren Trumpf zu bestimmen, werden zuerst die k nächsten Nachbarn dieser Hand bestimmt. Anschließend wird geprüft, welcher Trumpf unter den Nachbarn am häufigsten vertreten ist. Von diesem wird dann angenommen, dass dieser Trumpf auch mit dem noch nicht klassifizierten Blatt spielbar ist.

5.3.2 Die Metrik

Als Metrik wird im Werk von Kupferschmid und Keller vorgeschlagen die Distanz der verschiedenen Hände anhand von als relevant betrachteten Karten zu bestimmen. In dieser Arbeit soll hingegen ein naiverer Ansatz getestet werden, der allerdings ohne eine vorher festgelegte Gewichtung der einzelnen Karten auskommt {vgl. Dengel 2008 #16 S.97}.

Die in dieser Arbeit verwendete Metrik, geht von der Überlegung aus, dass es die Spielbarkeit nur selten beeinflusst, wenn man eine Karte durch eine ähnliche Karte ersetzt. Dabei werden zwei gewichtete Kriterien definiert, ab wann sich Karten ähnlich sind und so die nächsten Nachbarn ermittelt. Zuerst wird bestimmt, welche Karte ersetzt wurde und durch welche sie ersetzt wurde. Diese werden dann anhand der Kriterien verglichen.

Das erste Kriterium bestimmt sich nach dem Abstand der Zahlwerte der zwei Karten. Es wird davon ausgegangen, dass es, solange nur wenige Karten betroffen sind, egal ist, ob sie im Zahlenwert voneinander abweichen, da im schlimmsten Falle so ein Stich mehr an die Gegenpartei geht, als mit einer anderen Hand möglich gewesen wäre. Das zweite Kriterium bestimmt, ob sich die getauschten Karten in der Farbe unterscheiden. Dies wird allerdings höher gewichtet, da es so passieren kann, dass ein Trumpf durch einen schlechten nicht-Trumpf ersetzt wurde, was die Hand maßgeblich schwächen würde.

Aus diesen Kriterien ergibt sich dann folgende Metrik, um die Nähe zweier Hände zu bestimmen. Als erstes wird die Anzahl der Karten, die sich zwischen den beiden Händen unterscheiden, bestimmt. Diese Zahl geht mit dem Faktor sechzehn in den Abstand ein. Als nächstes wird der Abstand der getauschten Karten zueinander betrachtet. Dabei wird immer das Minimum der möglichen Werte betrachtet. Zuerst wird betrachtet, wie viele Farben sich zwischen den Karten unterscheiden. Diese Anzahl geht mit dem Faktor acht in den Abstand mit ein. Zuletzt wird noch der Abstand der Zahlwerte verglichen. Dieser geht mit dem Faktor 1 in den Abstand ein. Die Faktoren sorgen dafür, dass ein Farbwechsel, auch bei demselben Zahlwert der Karte, als weiter weg deklariert wird, als ein Unterschied in den Zahlwerten, da der maximale Unterschied zwischen den Zahlwerten zweier Karten sieben ist. Ebenso wurde der Faktor sechzehn gewählt, damit der Abstand kleiner ist, je weniger Karten sich unterscheiden, denn der größtmögliche Unterschied, der zwischen zwei Karten errechnet werden kann, ist fünfzehn, sieben als Unterschied zwischen den Zahlwerten plus acht, wenn sich die Farbe der Karten unterscheidet.

5.3.3 Die Umsetzung

Bei der Implementation des Algorithmus, wurden weitere Annahmen gemacht, um die Effizienz des Programms zu erhöhen. Zuerst wird im Programm davon ausgegangen, dass Hände, die nicht in der Datenbank auftauchen, nicht spielbar sind, aber weniger Gewicht haben, als Hände, die gespeichert sind und somit die Datenbank vollständig ist. Außerdem wird davon ausgegangen, dass sobald sich mehr als zwei Karten unterscheiden, die Hände zu verschieden sind, um sinnvoll in die Klassifizierung mit einbezogen werden zu können.

Unter diesen Annahmen wird vom Programm jede mögliche Kombination der Handkarten, wo bis zu zwei Karten getauscht wurden, ermittelt und anschließend nach deren Abstand zur ursprünglichen Hand sortiert. So ergibt sich dann auch das k dieses Ansatzes als die Anzahl der möglichen Hände, bei denen bis zu zwei Karten getauscht wurden, also 2531. Damit ergibt sich das k in dieser Arbeit, anders als bei Keller und Kupferschmid, nicht aus einem Versuch, welches k die kleinsten Fehler liefert, sondern aus einer Annahme über die möglichen Daten-{vgl. Dengel 2008 #16 S.97}.

Die Idee, die damit getestet werden soll, ist, dass der Computer, ähnlich einem unerfahrenen Spieler, in unsicheren Situationen passt, anstatt dass zu weit entfernte Hände Einfluss auf das Reizen nehmen. Falls keine dieser 2531 Hände in der Datenbank gefunden werden, wird davon ausgegangen, dass es nicht möglich ist das Spiel zu gewinnen und der Computer wird passen. Ansonsten wird bis zu dem Trumpf gereizt, der unter den möglichen Nachbarn am häufigsten vorhanden war. Bei einem Gleichstand wird das Spiel bevorzugt, das mehr Punkte bringen würde.

Beim anschließenden Abwerfen werden dann für alle möglichen Konstellationen, die nach einem Abwurf von zwei Karten möglich wären, derselbe Algorithmus durchgeführt, wie um die Spielstärke zu Beginn zu bestimmen. Anschließend wird dann die Hand gespielt, bei der von allen möglichen Händen, die meisten Nachbarn denselben Trumpf nahelegen.

Auch hier gibt es einige Unterschiede zu dem Ansatz von Keller und Kupferschmid. Bei deren Ansatz werden einige Möglichkeiten nicht in Betracht gezogen, wenn es darum geht, welche Karten abgeworfen werden. Dabei wird dann nach zwei Regeln aussortiert. Zum einen werden keine Buben abgeworfen und zum anderen werden nur Spiele berücksichtigt, die nach der von ihnen benutzen Metrik noch spielbar sind. Dabei ist hinzuzufügen, dass die erste Regel nur dann sinnvoll ist, wenn Nullspiele für den Computer nicht in Frage kommen. Zudem wird eine angepasste Datenbank benutzt, auf der die Metrik dann für den Abwurf bessere Ergebnisse liefert {vgl. Dengel 2008 #16 S.100}. In der hier vorgestellten Implementation ist das Nullspiel berücksichtigt, weswegen die erste Regel nicht umzusetzen war. Die zweite Regel ist nicht umzusetzen gewesen, da diese sich auf eine andere Metrik bezogen hätte und somit nicht mit der hier vorgestellten Metrik kompatibel wäre.

5.4 Naive Bayes

Die zweite KI nutzt einen naiven Bayes Klassifikator, um die Spielstärke von Händen zu bewerten. Dabei handelt es sich um einen probabilistischen Klassifikator, der vom Satz von Bayes abgeleitet ist {vgl. Alpaydın 2008 #13 S.46}.

5.4.1 Der Satz von Bayes

Der Satz von Bayes ist ein Satz aus der Wahrscheinlichkeitsrechnung, der 1763 vom englischen Mathematiker Thomas Bayes veröffentlicht wurde. Er ist eine Abwandlung der Definition der bedingten Wahrscheinlichkeit und erlaubt eine teilweise Umkehrung der bedingten Wahrscheinlichkeit, unter der Voraussetzung, dass die a-priori-Wahrscheinlichkeit von A bekannt ist. Der Satz ist wie folgt definiert:

$$P(A \mid B) = \frac{(P(B \mid A) \times P(A))}{(P(B))}$$

Dabei sind P(A|B) und P(B|A) die bedingten Wahrscheinlichkeiten, dass das eine Ereignis eintreten wird unter der Bedingung, dass jeweils andere schon eingetroffen ist. P(A) und P(B) sind die a-priori-Wahrscheinlichkeiten, dass das jeweilige Ereignis eintritt, also die Wahrscheinlichkeit, die aufgrund von Vorwissen ermittelt wurde. Damit der Satz gilt, muss P(B) > 0 gelten. Bewiesen wird der Satz unmittelbar aus der Definition der bedingten Wahrscheinlichkeit.

$$P(A \mid B) = \frac{(P(A \cap B))}{(P(B))} = \frac{(\frac{(P(B \cap A))}{(P(A))} \times P(A))}{(P(B))} = \frac{(P(B \mid A) \times P(A))}{(P(B))}$$

5.4.2 Bayes Klassifikator

Bei dieser Art der Klassifikation werden sich potenziell ausschließende Klassen als Bernoulli-Variabeln dargestellt, die durch alle beobachtbaren Variablen bedingt wird. Somit kann mit der bedingten Wahrscheinlichkeit für jede potenzielle Klasse bestimmt werden, wie wahrscheinlich es ist, dass der beobachtete Fall zu ihr zugeordnet werden sollte. Um diese zu berechnen, wird der Satz von Bayes zur Hilfe genommen. Außerdem werden sich die Wahrscheinlichkeiten der einzelnen Klassen immer zu eins aufaddieren. Dabei werden die a-priori-Wahrscheinlichkeiten aus den bereits beobachteten Beispielen generiert. Die Wahrscheinlichkeit für das Auftreten eines Merkmals wird dabei als Evidenz bezeichnet und relativiert den Einfluss selten auftretender Variablen in den bereits beobachteten Datensätzen auf die Klassifizierung. Die bedingte Wahrscheinlichkeit, dass ein auftretendes Merkmal zu einer bestimmten Klasse gehört, kann dabei ebenfalls aus den vorliegenden Daten, nach der Formel der bedingten Wahrscheinlichkeit, berechnet werden. Diese wird dann Klassen-Likelihood genannt. Abschließend wird die aktuelle Beobachtung der Klasse zugeordnet, bei der die höchste Wahrscheinlichkeit errechnet wurde.

Damit macht der Bayes Klassifikator zwei wesentliche Voraussetzungen. Zum einen müssen die Klassen einander ausschließen und zum anderen sollten die zu beobachtenden Variablen nur von der Klasse und so wenig wie möglich voneinander abhängen {vgl. Anzai 1992 #17 S.278f.}.

5.4.3 Die Umsetzung

Die genannten Voraussetzungen werden bei den Handkarten beim Skat sehr gut erfüllt. Als Erstes schließt natürlich jeder Trumpf jeden anderen aus, wobei auch die Option, zu passen, jeden anderen Trumpf ausschließt.

Die Unabhängigkeit, der einzelnen beobachteten Variabel, also welche Karten in der Hand sind und welche nicht, ist nicht gegeben:

$$Die Wahrscheinlichkeit eine spezielle Karte auf der Starthand zu haben$$

$$P(A) = P(B) = \frac{\binom{10}{1}}{\binom{32}{10}} = \frac{1}{6451224}$$

$$Die Wahrscheinlichkeit zwei spezielle Karte auf der Starthand zu haben$$

$$P(A \cap B) = \frac{\binom{10}{2}}{\binom{32}{10}} = \frac{3}{6451224}$$

$$Formel für stochastische Unabhängigkeit:$$

$$P(A \cap B) = P(A) \times P(B)$$

$$\frac{3}{6451224} \neq \left(\frac{1}{6451224}\right)^2$$

Jedoch kommt diese Unabhängigkeit nur deshalb zustande, da, sobald eine Karte der Hand bekannt ist, die Wahrscheinlichkeit, eine andere Karte in der Hand zu haben sinkt, da es nur noch neun unbekannte Karten auf der Hand des Spielers gibt. Da dies für die Auswahl des Trumpfs zu vernachlässigen ist, scheinen die Variablen schwach genug korreliert zu sein, um diesen Ansatz zu testen.

Ein weiterer Vorteil des naiven Bayes Klassifikator ist, dass er mit sehr wenigen und einfachen Datenbankabfragen auskommt. So werden in diesem konkreten Beispiel für die Bewertung einer Hand lediglich einundsiebzig Datenbankabfragen gebraucht, was im Vergleich zum k-nearest-neighbour gerade mal 2% der Abfragen entspricht.

6. Evaluation

6.1 Datenerhebung

Da in dieser Arbeit zwei lernende KI‘s verwendet wurden, ist Testdata nötig, um am Ende einen Lernfortschritt, beziehungsweise die Stärke der KI‘s zu ermitteln. Da es sich bei beiden Lernalgorithmen um unüberwachtes Lernen handelt, wird das Training der KI‘s durch Spiele gegen andere KI‘s stattfinden(siehe 3.2.2).

Das Training der KI‘s findet im Wesentlichen durch die naive KI statt. Zu diesem Zweck traten die drei KI‘s in Spielen wiederholt gegeneinander an, wobei sowohl die Hand als auch der Ausgang des Spiels in der Datenbank hinterlegt wurden. So sollte auf Grundlage des Algorithmus der naiven KI ein Trainingsdataset generiert werden, das dann von den beiden lernenden KI‘s mit weiterentwickelt werden kann. Zu diesem Zweck spielten die KI‘s insgesamt 1600 Spiele mit je fünf Runden.

Als abschließendes Testdataset dient ein Satz aus zweihundert Händen, die zu Beginn des Trainings von keiner KI zugeordnet werden konnte. Diese Hände wurden von menschlichen Skatspielern bewertet und in einer extra Datenbank gespeichert.

Zudem werden die Ergebnisse der Spielrunden der KI‘s dokumentiert und ausgewertet. Dabei wurde gespeichert, welche KI die Partie gewonnen hat, welche KI als Solospieler in jeder Runde gespielt hat und wie oft ein Spiel gewonnen oder verloren wurde.

6.2 Ergebnisse der Spielrunden

Während des Trainings wurden insgesamt 1300 Spiele gespielt, bei dem jedes Spiel aus fünf Runden bestand. Dabei wurden von den drei Computern Punktestände von -228 bis 261 erreicht. Es gingen 64 Spiele unentschieden aus, also konnte kein eindeutiger Sieger ermittelt werden.

Die naive KI hat von diesen 1300 Spielen insgesamt 714 gewonnen, was einer Gewinnquote von 54,92% entspricht. Dabei erzielte sie Punktwerte zwischen -216 und 261. Im Schnitt machte die naive KI 70,37 Punkte pro Spiel.

Die KI, die den k-nearest-neighbour Algorithmus verwendete, gewann insgesamt 114 der 1300 Spiele. Damit liegt ihre Gewinnquote bei 8,77%. Ihre Punktewerte lagen zwischen -144 und 168 und im Schnitt wurden pro Spiel 16,22 Punkte erreicht.

Die KI, die den Naiven Bayes Klassifizieren benutzt hat, kam insgesamt auf 408 gewonnene Spiele, was einer Gewinnquote von 31,38% entspricht. Dabei erzielte sie Punktwerte zwischen -228 und 256 und im Schnitt machte sie 35,43 Punkte pro Spiel.

Anhand der Punktespannen ist zu erkennen, dass wahrscheinlich alle Arten von Trumpf gespielt wurden, und das es auch viele verlorene Spiele gab. Wenn man davon ausgeht, dass alle Spieler gleich stark sind, hätten die Gewinnquoten ungefähr gleich sein und in etwa bei 30% liegen sollen. Dieser Wert ist jedoch nur bei der dritten KI zu beobachten, die eine Gewinnquote von 31,38% hatte. Die anderen beiden weichen deutlich von dem erwarteten Wert ab.

Dies lässt sich dadurch erklären, dass die beiden KI's, die einen lernenden Algorithmus benutzen, am Anfang mit einer leeren Datenbank spielten und somit am Anfang kaum ein Spiel als spielbar erkennen konnten. Somit ist klar, dass am Anfang jedes Spiel von der ersten KI gewonnen wurde. Auch in der Datenbank ist erkennbar, dass das der wahrscheinliche Grund für die Abweichung der Gewinnquote der ersten KI ist, da erstmals in Spiel 31 beide anderen KI's Punkte erzielt haben. Allerdings ist davon auszugehen, dass die Gewinnquote der dritten KI dadurch auf den erwarteten Wert gehoben wurde, da die zweite auf einer leeren Datenbank nur sehr schlechte Ergebnisse geliefert hat.

Um die Leistungen der KI's auf einer größeren Datenmenge, ca. 1000 verschiedene Hände in der Datenbank, zu testen, werden im folgenden dieselben Daten ausgewertet, allerdings nur von den letzten 100 Spielen. Hierbei wurden Punkte zwischen -110 und 183 erreicht. Von den 100 Spielen konnte bei 12 kein eindeutiger Sieger ermittelt werden.

Von diesen 100 Spielen gewann die naive KI 44%, mit einer durchschnittlichen Punktzahl von 52,87 pro Spiel. Sie erreichte Punktzahlen zwischen -66 und 152.

Die zweite KI gewann 20% aller Spiele mit einem durchschnittlichen Punktestand von 19,75 pro Spiel. Die Punktestände variierten zwischen -110 und 140.

Die dritte KI gewann 24% aller Spiele. Dabei erzielte sie Punktstände zwischen -144 und 183, bei einem durchschnittlichen Wert von 32,81 Punkten pro Spiel.

Hier ist ganz deutlich zu sehen, wie sehr sich die KI, die den k-nearest-Neigbour Algorithmus verwendet, verbessert hat. Ihre Gewinnquote ist um 12% gestiegen und die durchschnittlich erreichten Punkte sind um ca. sechs gestiegen. Interessant ist auch, dass nicht nur die Gewinnquote der ersten, sondern auch der zweiten KI gefallen sind, nämlich um 11% und um 7%. Hier ist ganz klar zu sehen, dass anfangs die beiden stärkeren KI‘s von einem schwachen Mitspieler profitierten und so bessere Quoten erreichen konnten. Außerdem ist zu beobachten, dass sich die beiden lernenden KI‘s annähern und nun bereits auf demselben Niveau spielen. Davon ausgehend, dass sich diese Beobachtung fortsetzen wird, kann man vermuten dass die KI, die den k-nearest-Neigbour Algorithmus verwendet, mit wachsenden Trainingsdaten stärker werden wird, wobei die KI die den Naiven Bayes Klassifizierer benutzt hat, auf einer kleineren Datenmenge bessere Ergebnisse lieferte. Dies wird wahrscheinlich daran liegen, dass die Variablen, also die vorhandenen Karten nicht stochastisch unabhängig genug sind, um auf Dauer mit diesem Algorithmus gute Ergebnisse zu erzielen. Weiterhin interessant zu beobachten ist, dass die Gewinnquote der naiven KI immer noch deutlich höher ist, als die der lernenden KI‘s. Dies wird im Wesentlichen zwei Ursachen haben. Zum einen muss bedacht werden, dass die Datenbank, trotz 6360 Einträgen, gerade mal zu 0,0098%, ausgehend von allen möglichen Händen, gefüllt ist und damit ein relativ kleiner Datensatz vorliegt. Im Gegensatz dazu, ist die naive KI in der Lage alle Blätter zu bewerten und so nicht an Trainingsdaten gebunden. Zudem reizt die naive KI selbst eher vorsichtig, was die lernenden KI‘s in Ermangelung an anderen Trainingspartnern übernehmen und so auch nur vorsichtig reizen.

6.3 Ergebnisse der Einzelrunden

Jedes der 1300 gespielten Spiele bestand aus jeweils fünf Runden. Über diese Runden kann nun ausgewertet werden, wie gut die KI‘s jeweils die Hände klassifiziert haben. Insgesamt wurden 6500 Runden gespielt, von denen 461 unentschieden ausgingen, also bei denen alle 3 Computer gepasst haben. Das entspricht 7,09% der gesamten Spiele.

Die naive KI hat von den insgesamt 6500 Spielen 2708, also 41,66%, als Solospieler gespielt. Von diesen Spielen hat sie 2517 gewonnen und 191 verloren. Damit hat sie 38,72% aller Spiele gewonnen und 2,94% aller Spiele verloren. Bezogen auf die Anzahl der Spiele, die die KI als Solospieler spielen wollte, hat sie 92,95% der Blätter als spielbar erkannt, die auch spielbar waren. 7,05% der Blätter wurden allerdings als spielbar erkannt, obwohl sie nicht spielbar waren.

Die KI, die den k-nearest-neighbour Algorithmus verwendete, spielte768 der 6500 Spiele als Solospieler, das entspricht 11,82%. Von allen Spielen gewann sie 666, also 10,82% und verlor 102, also 1,57%. Von den von ihr als spielbar klassifizierten Spielen waren 86,72% spielbar und 13,28% nicht spielbar.

Die KI, die den Naiven Bayes Klassifizieren benutzt hat, hat insgesamt 2561 Spiele gespielt, was 39,4% aller Spiele ausmacht. Davon waren 2021, also 31,09% aller Spiele, gewonnen und 540, also 7,75% aller Spiele, verloren. Damit waren 78,91% der von ihr als spielbar klassifizierten Hände spielbar und 21,09% waren fälschlicher Weise als spielbar klassifiziert.

Diese Auswertung gibt ein erstes Bild über die Stärke der KI‘s. Potenziell können KI‘s zwei Fehler unterlaufen. Zum einen kann eine Hand als spielbar gewertet werden, die aber nicht spielbar ist, also mit der der Computer dann verliert. Zum andern kann eine Hand, die eigentlich spielbar wäre, als nicht spielbar abgelehnt werden. Durch diese Auswertung kann nur ermittelt werden, wie anfällig die Computer gegenüber dem zuerst genannten Fehler sind. Hierbei fällt auf, dass die naive KI die mit Abstand besten Werte aufweist. Dies ist im Algorithmus begründet, den die KI benutzt, um Blätter zu bewerten, da dieser versucht, nur sehr sichere Blätter zu erkennen und auch nur solche zu spielen versucht. Hier wird vermutet, dass diese KI beim zweiten Fehler einen wesentlich schlechteren Wert aufweisen wird. Bei den anderen beiden KI‘s ist auffällig, dass die zweite KI, die weniger Spiele gewonnen hat, trotzdem einen besseren Wert für den erst genannten Fehler aufweist, als die dritte. Das wird daran liegen, dass wieder alle gespielten Spiele betrachtet werden, also auch die, bei der die lernenden KIs auf Grund mangelnder Trainingsdaten nur passen konnten.

Daher wird auch hier noch einmal ein Teil der Spieldaten gesondert betrachtet und zwar die letzten 500 Spiele. Dies entspricht den Spielen der letzten 100 Runden wie sie in 6.3 betrachtet wurden. Von diesen 500 Spielen wurden 44 eingepasst, was 8,8% der Spiele entspricht.

Die erste KI hat insgesamt 142 Spiele gespielt, was 28,4% aller Spiele ausmacht. Davon waren 135, also 27% aller Spiele, gewonnen und 7, also 1,4% aller Spiele, verloren. Damit waren 95,07% der von ihr als spielbar klassifizierten Hände spielbar und 4,93% waren fälschlicher Weise als spielbar klassifiziert.

Die zweite KI spielte 85 der 500 Spiele als Solospieler, das entspricht 17%. Von allen Spielen gewann sie 70, also 16,4% und verlor 15, also 3%. Von den von ihr als spielbar klassifizierten Spielen waren 82,35% spielbar und 17,65% nicht spielbar.

Die dritte KI hat von den insgesamt 500 Spielen 229, also 45,8%, als Solospieler gespielt. Von diesen Spielen hat sie 172 gewonnen und 57 verloren. Damit hat sie 34,4% aller Spiele gewonnen und 11,4% aller Spiele verloren. Bezogen auf die Anzahl der Spiele die die KI als Solospieler spielen wollte, hat sie 75,11% der Blätter als spielbar erkannt, die auch spielbar waren. 24,89% der Blätter wurden allerdings als spielbar erkannt, obwohl sie nicht spielbar waren.

Hier ist, entgegen der Erwartung, zu beobachten, dass der Fehler bei den beiden lernenden KI‘s größer wird während der der naiven KI sich weiterhin verbessert. Die Verbesserung der naiven KI scheint hierbei jedoch ein Zufall zu sein, da sie immer mit demselben Algorithmus die Spielbarkeit bestimmt und so eine konstante Erkennungsrate liefern sollte. Die Abnahme bei den beiden lernenden KI‘s ist wahrscheinlich dadurch zu erklären, dass mit umfangreicheren Trainingsdaten immer mehr Hände vermeintlich bestimmbar werden und so auch mehrere falsche Hände von den KI‘s getestet werden. Diese Vermutung liegt besonders deswegen nahe, da hier nur der Fehler betrachtet wird, dass eine Hand, die nicht spielbar ist, als spielbar klassifiziert wird. Allerdings ist auch davon auszugehen, dass wenn diese Hand erneut bewertet werden soll, die lernenden KI‘s eine andere Entscheidung treffen werden.

Zudem ist zu bemerken, dass obwohl die KI, die den k-nearest-Neigbour Algorithmus verwendet, zu einem höheren Prozentsatz eine Fehleinschätzung vermeidet, hat die KI, die den Naiven Bayes Klassifizieren benutzt hat, mehr Spiele gewonnen. Das zeigt, dass auch der andere Fehler, ein Blatt das spielbar ist als nicht spielbar zu klassifizieren, einen großen Einfluss auf das Ergebnis hat. Allerdings muss der hier untersuchte Fehler höher gewichtet werden, da laut Skatregeln ein verlorenes Spiel doppelte Minuspunkte bringt und damit nicht mit nur einem gewonnenen Spiel aufgewogen werden kann(siehe 2.2.2).

6.4 Genauigkeit der KI

Um eine Genauigkeit der KI‘s anzugeben, werden sie zuletzt mit 202 von menschlichen Skatspielern bewerteten Händen getestet. Der Test läuft in drei Kategorien ab. In der ersten wird getestet, zu welchen Händen der richtige Trumpf klassifiziert wird und in der zweiten Kategorie geht es darum, dass die KI‘s erkennen sollen, ob eine Hand spielbar ist oder nicht und in der dritten wird getestet, welchen Fehler die KI‘s öfter machen, also ob öfter eine spielbare Hand als nicht spielbar klassifiziert wird oder umgekehrt.

Die Klassifikationen, die die naive KI vorgenommen hat, stimmen in 57,92% der Fälle genau mit den Angaben der menschlichen Spieler überein. In 68,32% der Fälle wird die Spielbarkeit der Hand korrekt bewertet. Dabei werden 37,6% der nicht spielbaren Hände als spielbar anerkannt und 22,08% der Spielbaren Hände werden als nicht spielbar anerkannt.

Die Klassifikationen des k-nearest-Neigbour Algorithmus stimmen in 55,45% der Fälle mit den Angaben der menschlichen Spieler überein und zu 63,37% wird die Spielbarkeit einer Hand korrekt klassifiziert. Dabei werden 32,8% fälschlicherweise als spielbar und 42,86 fälschlicherweise als nicht spielbar klassifiziert.

Die Klassifikationen des naiven Bayes Klassifizier stimmen in 27,23% der Fälle mit den Vorgaben überein und in 41,09% der Fälle wird die Spielbarkeit der Hand richtig klassifiziert. Dabei werden 96,3% der nicht spielbaren Hände als spielbar eingestuft und 2,6% der spielbaren Hände werden als nicht spielbar eingestuft.

Hier ist ganz klar zu sehen, was sich in den vorangegangenen Auswertungen schon abzeichnete. Bei dem naiven Bayes Klassifikator liegt der Fehler bei der Klassifizierung von nicht spielbaren Händen extrem hoch. Eine mögliche Ursache für dieses Verhalten ist, dass beide KI‘s von derselben KI gelernt haben und so der Spielstil der naiven KI auch den gesamten Trainingsdatensatz prägt. Interessant ist hierbei die Beobachtung, dass der k-nearest-Neigbour Algorithmus trotz eines geprägten Trainingsdatensatzes relativ gute Ergebnisse liefert, verglichen mit 75% {vgl. Dengel 2008 #16 S.101}.

Ein weiterer möglicher Grund für die Stärke des k-nearest-Neigbour Algorithmus könnte allerdings auch die relativ kleine Datenmenge sein, auf der die beiden lernenden Algorithmen operieren. Da der Algorithmus so geschrieben ist, dass er bei mangelnden Daten passt, könnte es sein, dass es für die 202 Beispiele lediglich keine Daten gibt, die nah genug sind um die Hände klassifizieren zu können. Das wohl beste Ergebnis liefert aber der Naive Ansatz. Hier scheint die große Zahl an Händen, die nach der Formel klassifiziert werden können, der größte Vorteil zu sein. Allerdings ist hier zu beachten, dass je mehr Daten die lernenden KI‘s zur Verfügung haben werden, desto besser wird deren Klassifizierung werden. Außerdem handelt es sich bei der naiven KI um eine sehr vorsichtig spielende KI, die riskantere Hände grundsätzlich ablehnt, auch wenn diese theoretisch spielbar wären.

Als abschließenden Test werden die beiden lernenden KI‘s noch geprüft, in wie weit sie das Verhalten der naiven KI vorhersagen können. Dazu werden 100 zufällige Hände generiert und anschließend von der naiven KI bewertet. Anschließend wird geprüft, ob die lernenden KI‘s zum selben Ergebnis bestimmen oder zumindest dieselbe Spielbarkeit klassifizieren.

Hierbei sind die Klassifikationen des k-nearest-Neigbour Algorithmus zu 44% korrekt und zu 58% wird zumindest die Spielbarkeit korrekt klassifiziert.

Bei dem naiven Bayes Klassifikator sind 29% der Klassifikationen korrekt und zu 52% wird die Spielbarkeit korrekt bewertet.

Damit ist es sehr wahrscheinlich, dass die Variablen, also die Karten, die in der Hand sind, zu stark korrelieren, als das ein naiver Bayes Klassifikator sich als Skatcomputer eignen würde. Allerdings zeigt sich hier auch, dass der k-nearest-Neigbour Algorithmus nicht nur besser die Hände klassifiziert, sondern auch, dass der Lernfortschritt bei der Imitation der Klassifizierung nach den Vorgaben der naiven KI besser klappt, als bei der anderen lernenden KI.

7. Fazit

Mit der Vorgabe, dass eine KI mit 75% Wahrscheinlichkeit gewinnbare Spiele erkennen kann, sind die Ergebnisse, die in dieser Arbeit erreicht wurden, nicht weit weg. Zum einen wurde eine algorithmische Lösung gefunden, um die Spielbarkeit eines Blattes zu bewerten, die komplett ohne Datenbank auskommt und trotzdem mit 68% Wahrscheinlichkeit richtig klassifiziert. Außerdem ist die These, dass der k-nearest-Neigbour Algorithmus der bestmögliche Algorithmus ist, um die Spielstärke eines Skatblattes zu bewerten, untermauert, da er mit 63% ebenfalls gute Ergebnisse liefert. Außerdem wurde nachgewiesen, dass sich der naive Bayes Klassifikator, nur auf einem sehr unvollständigen Datensatz als Klassifikator für das Reizen beim Skat eignet.

Während der Arbeit ist zudem aufgefallen, dass es selbst bei einem einfach scheinenden Spiel, mit nur zehn Handkarten und 32 Karten im Stapel, eine Vielzahl von Möglichkeiten gibt, die ein Computer verstehen und bewerten können muss. Allein, dass es beim Skat rund 64 Millionen verschiedene Möglichkeiten für die Handkarten am Beginn eines Spieles gibt, lässt es unmöglich scheinen, einen vollständigen Datensatz zu erzeugen. Was aber umso mehr zeigt, dass es notwendig ist, sich solchen Problemen mit maschinellem Lernen zu nähern.

Zukünftig könnten auch noch andere Algorithmen mit demselben Thema getestet werden, um herauszufinden, ob der k-nearest-Neigbour Algorithmus bereits der optimale ist oder ein anderer in Laufzeit oder Klassifikation bessere Ergebnisse liefert. Auf jeden Fall ist es wichtig, dass diese Tests auf einer ausreichend großen Datenmenge durchgeführt werden, um so die Stärken der lernenden Algorithmen hervorzuheben.

Festzuhalten ist, dass das Reizen auch durch Algorithmen ohne Lernen ausreichend gut simuliert werden kann, dass aber nur durch maschinelles Lernen das Niveau eines Profis zu erreichen ist. Außerdem eignet sich der naive Bayes Klassifikator nicht für dieses Problem, da die Variablen, also die Karten auf der Starthand, zu sehr korrelieren, um in ausreichend vielen Fällen ein richtiges Ergebnis zu liefern.

8. Literaturverzeichnis

Ahrweiler, Petra (1995): Künstliche Intelligenz-Forschung in Deutschland. Die Etablierung eines Hochtechnologie-Fachs. Zugl.: Berlin, Freie Univ., Diss., 1993. Münster: Waxmann (Internationale Hochschulschriften, 141).

Albert Ziegler, Kurt A. Heller (2014): Intelligenz. Lexikon der Psychologie. Hg. v. Spektum.de. Online verfügbar unter https://www.spektrum.de/lexikon/psychologie/intelligenz/7263, zuletzt geprüft am 17.02.2019.

Alpaydın, Ethem (2008): Maschinelles Lernen. Unter Mitarbeit von Simone Linke. München: Oldenbourg. Online verfügbar unter http://deposit.d-nb.de/cgi-bin/dokserv?id=2923451&prov=M&dok_var=1&dok_ext=htm.

Andreas Wichert (2014): Künstliche Intelligenz. Lexikon der Neurowissenschaft. Hg. v. Spektrum.de. Online verfügbar unter https://www.spektrum.de/lexikon/neurowissenschaft/kuenstliche-intelligenz/6810, zuletzt geprüft am 17.02.2019.

Anzai, Yūichirō (1992): Pattern recognition and machine learning. Boston: Academic Press. Online verfügbar unter http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=574153.

Breuer, Klaus (2012): Computerspiele programmieren. Künstliche Intelligenz für künstliche Gehirne. München: Oldenbourg (Informatik 10-2012). Online verfügbar unter http://www.oldenbourg-link.com/isbn/9783486717891.

Dengel, Andreas R.; Berns, Karsten; Breuel, Thomas M.; Bomarius, Frank; Roth-Berghofer, Thomas R. (Hg.) (2008): KI 2008: advances in artificial intelligence. 31st Annual German Conference on AI, KI 2008, Kaiserslautern, Germany, September 23 - 26, 2008 ; proceedings. KI; AI; Annual German Conference on AI. Berlin: Springer (Lecture notes in computer science Lecture notes in artificial intelligence, 5243). Online verfügbar unter http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10252174.

Deutscher Skatverband e. V. (2019): Historie, zuletzt geprüft am 17.02.2019.

Epstein, Robert; Beber, Grace; Roberts, Gary (Hg.) (2009): Parsing the Turing Test. Philosophical and Methodological Issues in the Quest for the Thinking Computer. Dordrecht: Springer Netherlands. Online verfügbar unter http://dx.doi.org/10.1007/978-1-4020-6710-5.

Fu, LiMin (1994): Neural networks in computer intelligence. New York: McGraw-Hill (McGraw-Hill series in computer science Artifical intelligence).

International Skat Players Association e.V., Deutscher Skatverband e.V. (2010): Internationale Skatordnung. Skatwettspielordnung. Altenburg. Online verfügbar unter https://www.dskv.de/upload_user/skatgericht/PDF/ISkO.pdf.

Johannes Fischer (2016): Als Deep Blue das Genie Garri Kasparow schlug. Hg. v. Zeit online. Online verfügbar unter https://blog.zeit.de/schach/als-deep-blue-das-genie-garry-kasparow-schlug/, zuletzt geprüft am 17.02.2019.

Matthias Bock (2003): Ich lerne Skat. Online verfügbar unter https://www.deutscherskatverband.de/fileadmin/dskv/content/Gerichte/Ich_lerne_Skat.pdf.

Unesco (2016): Skat spielen. Online verfügbar unter https://www.unesco.de/kultur-und-natur/immaterielles-kulturerbe/immaterielles-kulturerbe-deutschland/bundesweites-8, zuletzt geprüft am 17.02.2019.

Witten, Ian H.; Frank, Eibe (2005): Data mining. Practical machine learning tools and techniques. 2nd ed. Amsterdam, Boston, MA: Morgan Kaufman (Morgan Kaufmann series in data management systems). Online verfügbar unter http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=130260.

Wolfgang Rui (2010): Geschichte des Skatspiels. Hg. v. skat-extra.de. Online verfügbar unter http://www.skat-extra.de/historie/historie.html, zuletzt geprüft am 17.02.2019.

Wolfgang Rui (2013): Besser Skat spielen. Ottweiler. Online verfügbar unter http://www.skat-extra.de/assets/applets/besser-Skat-spielen-V2.pdf.