ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG
Fakultät für Angewandte Wissenschaften
Institut für Informatik

Lehrstuhl für Grundlagen der Künstlichen Intelligenz
Prof. Dr. Bernhard Nebel

18, 20, weg –
Automatisches Reizen und Drücken beim Skat

Studienarbeit von
Thomas Keller

Dezember 2007

Inhaltsverzeichnis

Inhaltsverzeichnis **1**

1 Einleitung **2**

2 Grundlagen des Skatspiels **4**
2.1 Spielmaterial und Spielvorbereitung 4
2.2 Spielermittlung 4
2.3 Ausspielen der Stiche 5

3 Maschinelles Lernen **6**
3.1 Lernen von Reizen und Drücken 6
3.2 Der Double Dummy Skat Solver 9

4 Der Algorithmus der kleinsten Fehlerquadrate **12**
4.1 Hintergrund 12
4.2 Parameter von LMS 13
4.3 Entwicklung der Merkmale M_2 16

5 Der k-Nächste-Nachbar-Algorithmus **19**
5.1 Hintergrund 19
5.2 Vergleich unterschiedlicher Parameter und Methoden 21
5.3 Distanzgewichte 26
5.4 Entwicklung verschiedener Wissensbasen 29
5.5 Analyse der Laufzeit 33

6 KNN und LMS **34**

7 Drücken **36**
7.1 Drücken mit KNN 36
7.2 Reizen auf gedrückten Spielen 38
7.3 Der Schwellenwert th und Evaluation ganzer Spiele 39

8 Zusammenfassung und Ausblick **42**

Literatur **43**

1 Einleitung

Eine beliebte Art zur Erforschung von Methoden und Algorithmen der *Künstlichen Intelligenz* (KI) ist deren Anwendung auf Spiele aller Art. Insbesondere das Schachspiel gegen Computer wurde dabei in den letzten Jahren medienwirksam verkauft, und rückte so dieses Forschungsgebiet in den Blickpunkt der Öffentlichkeit. Lange waren Schachgroßmeister der Maschine überlegen, bis 1997 Garri Kasparow als erster amtierender Schachweltmeister vom IBM-Computer Deep Blue geschlagen wurde (Hsu, 2002). Trotz der langen Entwicklungszeit — schon Alan Turing beschäftigte sich in den 40er Jahren mit Computerschach — bis hin zur Maschine, die selbst dem Weltmeister überlegen ist, gehört Schach nicht zu den kompliziertesten Spielen, mit denen sich die KI beschäftigt. Es ist nämlich ein Spiel, bei dem *vollständige Information* vorliegt. Lediglich die enorm vielen möglichen Stellungen auf dem Schachbrett verhinderten vor Deep Blue eine Maschine, die den Menschen in angemessener Zeit schlagen konnte. Vollständige Information liegt in der Spieltheorie dann vor, wenn den Spielern alle für das Spiel relevanten Informationen zugänglich sind. Ein Standardverfahren zum Lösen solcher Spiele ist zum Beispiel der *Minimax-Algorithmus*.

Auch diese Studienarbeit beschäftigt sich mit der Anwendung von Methoden der KI auf ein Spiel, nämlich auf das in Europa und besonders in Deutschland sehr populäre Skatspiel. Wie auch bei Poker oder Bridge stehen bei diesem keinem Spieler alle relevanten Informationen zur Verfügung, weshalb bewährte Algorithmen wie Minimax dafür nicht oder nur durch zusätzliche Modifikationen geeignet sind. Trotz der so erschwerten Umstände existieren aber auch Programme, die in Spielen, in denen *unvollständige Information* vorliegt, auch gute menschliche Gegner schlagen können. Ein Beispiel hierfür ist M. Ginsberg's Intelligent Bridgeplayer (GIB) (Ginsberg, 1999). Das Problem unvollständiger Information wird dort auf ein Problem mit vollständiger Information reduziert. Näher wird der dafür verwendete *Monte-Carlo-Ansatz* in Kapitel 3 beschrieben.

Der selbe Ansatz wurde von S. Kupferschmid bereits erfolgreich auf das Ausspielen der Stiche im Skat übertragen (Kupferschmid, 2003). Für die in dieser Studienarbeit untersuchten Teile des Spiels, das Reizen und Drücken, zeigte er aber, dass diese so nicht in effizienter Zeit berechenbar sind. Auch J. Schäfer verwendete dafür in seinem Vergleich unterschiedlicher Skatprogramme den regelbasierten Ansatz von XSkat (Schäfer, 2005; Gerhardt, 2007), während M. Ginsberg eine dem Eröffnungsspiel vieler Skatprogramme gleichende Datanbankmethode implementierte. In dieser Studienarbeit soll daher das Problem mit Methoden der KI besser und flexibler gelöst werden.

Beim Reizen gilt es abzuschätzen, wieviele Punkte in einem Spiel mit einer gegebenen Hand erreicht werden können, was durch die unvollständige Information über die Verteilung der Karten, die Art des Spieles sowie die Identität des Alleinspielers erheblich erschwert wird. Zwei alternative Ansätze sollen dafür analysiert und implementiert werden. Nach einer kurzen Einführung in die Grundlagen des Skatspiels sowie einem Überblick über die Problematik des maschinellen Lernens von Reizen und Drücken im

Skat wird in Kapitel 4 zunächst der *Algorithmus der kleinsten Fehlerquadrate* (LMS) sowie in Kapitel 5 der *k-nächste-Nachbar-Algorithmus* (KNN) präsentiert. Kapitel 6 beschäftigt sich dann mit einer möglichen Verbindung der beiden Ansätze. Für das bislang ebenfalls meist regelbasiert implementierte Drücken wird in Kapitel 7 abschliessend eine Lösung mithilfe einer Modifikation der gewonnenen Erkenntnisse angegeben.

2 Grundlagen des Skatspiels

In diesem Kapitel werden die zum Verständnis dieser Arbeit nowtwendigen Regeln des Skatspiels kurz skizziert, Informationen zu Spielkonzepten wie 'Hand', 'Schneider' oder 'Ouvert' werden dabei nicht gegeben. Ebenfalls unberücksichtigt bleiben, wie in der gesamten Studienarbeit, die verschiedenen Varianten der 'Null'. Diese können durch ihr stets ähnliches Erscheinungsbild sehr gut durch regelbasierte Reizer erkannt werden. Ein tieferes Spielverständnis kann durch die Lektüre des vollständiges Regelwerks, zum Beispiel der Internationalen Skatordnung, gewonnen werden (Skatordnung, 1998).

2.1 Spielmaterial und Spielvorbereitung

Skat ist ein Spiel für drei Personen, für das ein Blatt aus 32 Karten benutzt wird, zumeist ein Französisches oder Deutsches. Das französische Blatt, das in dieser Arbeit zur Grundlage genommen werden soll, besteht aus den Karten Sieben (7), Acht (8), Neun (9), Zehn (10), Bube (J), Dame (Q), König (K) und Ass (A) in den vier Farben Karo (♢), Herz (♡), Pik (♠) und Kreuz (♣). Jeder Karte ist so ein bestimmter Wert unabhängig von ihrer Farbe zugeordnet, dass sich eine Gesamtpunktzahl von 120 ergibt:

Karte	A	10	K	Q	J	9	8	7
Wert	11	10	4	3	2	0	0	0

Tabelle 2.1: Kartenwerte

Vor dem Spiel wird der erste Geber ermittelt, der bei jedem weiteren Spiel im Uhrzeigersinn wechselt. Dieser teilt jedem Mitspieler zehn Karten verdeckt aus, die nur von diesem eingesehen werden dürfen. Die beiden übrigen Karten bilden den dem Spiel den Namen gebenden 'Skat', der verdeckt in der Mitte des Tisches plaziert wird. Der Spieler links neben dem jeweiligen Geber wird Vorhand genannt, die weiteren Spieler heißen Mittelhand und Hinterhand. Im Wesentlichen bestimmen zwei voneinander stark verschiedene Teile das Spiel: Auf der einen Seite die Spielermittlung, die aus Reizen, Drücken und Spielansage besteht, sowie auf der anderen Seite das Ausspielen der Stiche.

2.2 Spielermittlung

In der ersten Phase des Spieles wird durch das *Reizen* der Alleinspieler ermittelt. Nach einer festen Abfolge fragt zunächst die Mittelhand bei der Vorhand Spielwerte von möglichen Spielen ab. Die Vorhand kann bei jeder Frage entscheiden, ob sie diesen oder einen höheren Spielwert erreichen kann und geht dementsprechend mit oder muss passen. Kann einer der beiden Spieler keinen höheren Wert mehr ansagen oder akzeptieren, reizt die Hinterhand auf die selbe Weise mit dem Gewinner der ersten Reizrunde weiter. Der Sieger des Reizens ist der neue Alleinspieler, die beiden übrigen Spieler spielen in dieser Runde zusammen und bilden die Gegenpartei. Dennoch ist es ihnen nicht erlaubt,

in die Karten ihres jeweiligen Mitspielers einzusehen. Der Spielwert wird dabei über die Anzahl und Art der Buben sowie den gewünschten Trumpf berechnet. Als Möglichkeiten stehen dafür Grand (G) und die vier möglichen Farbspiele (♣,♠,♡,♢) zur Verfügung. Bietet der spätere Alleinspieler dabei einen Wert, der höher ist als der in der Endabrechnung erzielte, verliert er das Spiel auf jeden Fall.

Nach dem Reizen darf der Alleinspieler den Skat aufnehmen und zwei beliebige Karten in den Skat zurücklegen. Dieser Vorgang wird *Drücken* genannt. Der Skat gehört dabei immer dem Alleinspieler, er ist also Teil seiner Punkte. Die Spielermittlung endet mit der *Trumpfansage* durch den Alleinspieler.

2.3 Ausspielen der Stiche

Ist die Spielermittlung abgeschlossen beginnt die zweite Phase des Spiels, das *Ausspielen der Stiche*. Hiermit beginnt die Vorhand, falls noch kein Stich gemacht wurde, sonst derjenige, der den letzten Stich für sich entschieden hatte. Ist noch keine Karte ausgespielt darf ein Spieler eine beliebige Karte zum Ausspielen wählen, sonst muss die Farbe bekannt werden, die zuerst gelegt wurde, sofern dies dem Spieler möglich ist. Die Trümpfe zählen hierbei als zu einer einzigen Farbe zugehörig, die vier Buben sind also Teil dieser Trumpffarbe. Kann ein Spieler die Farbe nicht bekennen darf er eine beliebige andere Karte legen. Die höchste der drei ausgelegten Karten gewinnt dann den jeweiligen Stich. Die Rangfolge der Karten ergibt sich dafür wie folgt:

Farbspiel	♣J ♠J ♡J ♢J A 10 K Q 9 8 7	(Trümpfe)
	A 10 K Q 9 8 7	(Nicht-Trümpfe)
Grand	♣J ♠J ♡J ♢J	(Trümpfe)
	A 10 K Q 9 8 7	(Nicht-Trümpfe)

Tabelle 2.2: Rangfolge der Karten (höherrangige Karten stehen links)

Der Rang der Nichttrumpffarben untereinander ist dabei immer der selbe, eine Nichttrumpfkarte einer Farbe kann niemals eine Nichttrumpfkarte einer anderen stechen.

Nachdem alle 10 Stiche auf diese Weise ausgespielt wurden, werden die Punkte des Alleinspieler sowie die der Gegenpartei gezählt. Hat der Alleinspieler mindestens 61 Punkte erreicht hat er das Spiel gewonnen, und der Spielwert wird ihm gutgeschrieben. Hat er dagegen das Spiel verloren bekommt er den doppelten Spielwert von seinen Punkten abgezogen.

3 Maschinelles Lernen

Der Vorgang des *maschinellen Lernens* ist in der KI so definiert, dass ein Computerprogramm in Bezug auf eine Menge von Aufgabenstellungen dann aus Erfahrungen lernt, wenn seine Leistungsfähigkeit in der Bearbeitung dieser Aufgaben, gemessen durch ein Erfolgsmaß, zunimmt. Die Grundlage ist also, dass ein für eine bestimmte Aufgabe geschaffenes Programm Erfahrungen sammelt, diese bewertet und dadurch seine Leistung verbessert. Der Ansatz wurde bereits erfolgreich auf ein breites Spektrum möglicher Probleme angewendet, unter anderem auf Sprach- und Schrifterkennung, Roboternavigation, oder eben auf die Strategiefindung in Spielen.

3.1 Lernen von Reizen und Drücken

Das Ziel dieser Studienarbeit ist das Lernen von Reizen und Drücken beim Skat. Die Aufgabe des Reizers besteht dabei darin, den Spielwert abzuschätzen, der als Alleinspieler mit einer gegebenen *Hand* oder *Anfrage* s erreicht werden kann. Dafür ist es notwendig, für alle Trümpfe $t \in T$, $T := \{G, \clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$ eine Vorhersage zu treffen, ob ein Spiel mit s unter diesem Trumpf gewonnen werden kann. Diese Funktion heißt *Bewertungsfunktion* $\widehat{V}_0^t(s) : S \to \{0; 1\}$ und sei definiert als

$$\widehat{V}_0^t(s) = \begin{cases} 0 & \text{s kann ein Spiel mit } t \text{ nicht gewinnen} \\ 1 & \text{s kann ein Spiel mit } t \text{ gewinnen} \end{cases} \tag{3.1}$$

wobei S die Menge aller möglichen Starthände oder Anfragen bezeichne. Eine andere mögliche Definition einer Bewertungsfunktion für das Skatspiel ist, einer Anfrage s die unter einem Trumpf erzielbaren Punkte p zuzuweisen. Sie ist also gegeben als

$$\widehat{V}^t(s) := S \to P, P := [0; 120] \cap \mathbb{N} \tag{3.2}$$

und impliziert eine *Klassifizierung* $C : S \to T \cup \{\emptyset\}$ mit

$$C(s) = \begin{cases} t & \text{falls } V^t(s) > th \text{ und } V^t(s) > V^{t'}(s) \text{ für alle } t' \neq t \in T \\ t & \text{falls } V^t(s) > th \text{ und } V^t(s) = V^{t'}(s),\ t \text{ erlaubt höheres Gebot} \\ \emptyset & \text{sonst} \end{cases} \tag{3.3}$$

Diese bezeichnet denjenigen Trumpf, mit dem eine Hand am ehesten ein Spiel gewinnen kann, da mit diesem die meisten Punkte erzielt werden, oder $\emptyset$, falls gar kein Trumpf aussichtsreiche Chancen auf einen Sieg hat. In Gleichung 3.3 bezeichne th dabei den *Schwellenwert*, der, sofern nicht anders erwähnt, aufgrund der Mindestpunktzahl zum Gewinn eines Spiels 60 beträgt. Eine Klassifizierung aus $\widehat{V}_0^t(s)$ abzuleiten ist dagegen abhängig von der verwendeten Methode und wird an den entsprechenden Stellen dieser Arbeit angegeben. Es sei allerdings darauf hingewiesen, dass $\widehat{V}^t(s)$ die aussagekräftigere Bewertungsfunktion darstellt: Nicht nur ist aufgrund des größeren Wertebereichs die Wahrscheinlichkeit kleiner, für mehrere Trümpfe die selbe Bewertung zu erhalten, sie enthält auch ein Maß für die Wahrscheinlichkeit, mit der die Klassifizierung tatsächlich

auf das Spiel übertragbar ist: Je weiter der vorhergesagte Wert vom Schwellenwert th entfernt ist, desto größer darf der Fehler in der Abschätzung sein, um trotzdem die korrekte Klassifizierung zu erhalten.

Auch der Vorgang des Drückens kann über diese Bewertungsfunktion gelernt werden. Dafür wird für alle möglichen Kombinationen aus zehn der zwölf Karten des Alleinspielers und für jeden mit dem gereizten Wert spielbaren Trumpf $\widehat{V}^t(s)$ berechnet. Die Klassifizierung erfolgt dann derart, dass einer Anfrage d, bestehend aus einer Hand s und dem Skat sk diejenigen zehn Karten c zugewiesen werden, welche die höchste Punktzahl erreichten. Außerdem wird auch der in diesem Spiel verwendete Trumpf gewählt. Eine solche Klassifizierung für das Drücken $DR(s, sk)$ hat also die Form $DR(s, sk) : S \times SK \to S \times T$, wobei SK der Menge aller Zweikartenkombinationen entspricht. Auf eine genauere Angabe dieser Zuweisung einer Klasse über die Bewertungsfunktionen $\widehat{V}^t(s)$ soll verzichtet werden.

Nachdem eine Bewertungsfunktion $\widehat{V}^t(s)$ definiert wurde, mit der ein Reizer eine Vorhersage über den Aussgang eines Spiels macht, um dann über die Klassifizierung $C(s)$ den möglichen Reizwert zu ermitteln, wird noch eine Funktion benötigt, welche die in der Definition des maschinellen Lernens erwähnte Erfahrung widerspiegelt. Diese Funktion heißt *Zielfunktion* $V^t(z)$, an die sich $\widehat{V}^t(s)$ im Verlauf des Lernvorgangs möglichst nahe annähern soll, um so die Leistung des Reizers zu steigern. Für diese Studienarbeit ist $V^t(z)$ gegeben als eine Menge von vorberechneten Trainingsbeispielen, der sogenannten *Trainingsmenge*. Wie diese erstellt wurde wird näher in Abschnitt 3.2 erklärt. Es handelt sich dabei aber — im Unterschied zur Anfrage s — um Spiele, über die nach der verdeckt durchgespielten Berechnung alle Informationen vorhanden sind, also $V^t(z) : Z \to P$, wobei $z = \langle s, c_1, c_2, sk, t, pos\rangle \in Z$ ein Skatspiel darstellt mit den Händen s, c_1 und c_2 sowie dem Skat sk. Alleinspieler ist in einem solchen Spiel z immer Spieler 0, er sitzt an Position $pos \in \{0; 1; 2\}$ und erreichte p Punkte mit Trumpf t. Wie genau die Annäherung vor sich geht hängt vom verwendeten Algorithmus ab und wird im jeweiligen Kapitel über die implementierten Lernalgorithmen beschrieben. Allen gemein sind aber Probleme, die aus der unvollständigen Information resultieren. Man betrachte dafür die folgenden beiden Spiele, in denen Spieler 0 jeweils die selben Karten hält und die dementsprechend durch die selbe Anfrage s beschrieben werden. Die dargestellten Informationen über die Hände der Spieler 1 und 2 sowie den Skat sind dem Reizer dabei nicht gegeben.

Spiel 1

s **(Spieler 0)**	♣J	♠J	♣10	♣K	♣7	♠10	♠9	♡10	♡7	♢10
c_1 **(Spieler 1)**	♡J	♢J	♣A	♣Q	♣9	♣8	♠A	♡A	♢Q	♢9
c_2 **(Spieler 2)**	♠K	♠Q	♠8	♠7	♡K	♡Q	♡9	♡8	♢A	♢K
sk **(Skat)**	♢8	♢7								

Spiel 2

s **(Spieler 0)**	♣J	♠J	♣10	♣K	♣7	♠10	♠9	♡10	♡7	♢10
c_1 **(Spieler 1)**	♡J	♣Q	♣8	♠A	♠8	♠7	♡9	♡8	♢Q	♢8
c_2 **(Spieler 2)**	♢J	♣9	♠K	♠Q	♡K	♡Q	♢A	♢K	♢9	♢7
sk **(Skat)**	♣A	♡A								

Tabelle 3.1: Unterschiedliche Spiele z für identische Anfrage s

Da der Reizer in beiden Fällen eine identische Anfrage s erhalten wird, wird er auch in beiden Fällen identische Bewertungsfunktionen $\widehat{V}^t(s)$ für alle Trümpfe und damit eine identische Klassifizierung $C(s)$ berechnen, unabhängig vom verwendeten Lernalgorithmus. Die resultierende Zielfunktion $V^t(s)$ ist in Tabelle 3.2 abgebildet:

Trumpf t	**G**	♣	♠	♡	♢
Spiel 1	20	20	22	24	42
Spiel 2	65	70	62	58	63

Tabelle 3.2: Zielfunktion $V^t(z)$ der Spiele aus Tabelle 3.1 für alle Trümpfe t

Offensichtlich ergeben sich in dieser signifikante Unterschiede für die beiden Spiele. Insbesondere kann Spieler 0 in Spiel 1 mit keinem Trumpf t ein Spiel gewinnen, im zweiten Spiel wird er aber lediglich mit Trumpf $t = ♡$ verlieren. Da der Reizer aber selbst bei Ausnutzung all seines Wissens die beiden Fälle nicht unterscheiden kann, wird er beiden Fällen die selbe Klasse zuweisen. Für Spiel 1 wäre $C(s) = \emptyset$ die korrekte Einordnung, für Spiel 2 aber $C(s) = ♣$, in mindestens einem der beiden Fälle wird er also falsch liegen.

Man erkennt daran, dass es wichtig ist, das *Maß für die Leistungsfähigkeit* des Reizers gut einzuschätzen. Beobachtet man selbst die weltbesten Skatspieler als allwissender Betrachter ergeben sich häufig Situationen, in denen ein Spieler anscheinend suboptimale Entscheidungen fällt: Der Skat wäre perfekt für seine Hand, aber er reizt nicht darauf; er spielt die eine Farbe an, die er erst in den letzten Stichen hätte anspielen sollen; oder er reizt auf eine knappe Hand und bekommt gerade die beiden Karten aus dem Skat, die ihn das Spiel dennoch verlieren lassen. Unter der vollständigen Information des allwissenden Beobachters sind diese Spielentscheidung tatsächlich nicht optimal, aber unter dem den Spielern gegebenen Wissen können auch optimale Entscheidungen

ein verlorenes Spiel nach sich ziehen. Für das Maß der Leistungsfähigkeit eines Reizers heißt dies insbesondere, dass nicht erwartet werden darf, dass alle Spiele auf den Punkt genau vorhergesagt oder auch nur alle korrekt klassifiziert werden. Es geht vielmehr darum, eindeutige Spiele sicher zu erkennen und unsichere Hände der Wahrscheinlichkeit eines Sieges entsprechend einzuordnen. Die Leistungsfähigkeit aller Implementationen des Reizers wie später auch des Drückers wurde auf einer *Evaluierungsmenge* getestet, deren Zustandekommen ebenfalls im nächsten Abschnitt näher erläutert wird. Das erste verwendete Maß für die Güte der Algorithmen, bei denen $\widehat{V}^t(s)$ als Bewertungsfunktion verwendet wird, ist dabei der *mittlere quadratische Fehler* zwischen $\widehat{V}^t(s)$ und der entsprechenden, berechneten Zielfunktion $V^t(z)$. Das zweite, allgemeinere und auch für die Bewertungsfunktion $\widehat{V}_0^t(s)$ verwendbare Maß ist analog zu Abbildung 3.1 fünfgeteilt, wobei zumeist lediglich die beiden positiven Fälle 1 und 4 näher analysiert werden. Alle Werte werden dabei in Prozent angegeben, wobei die Summe in beiden Spalten separat 100% ergibt. Der erste Fall bezeichnet dabei den Prozentsatz korrekt als nicht gewinnbar vorhergesagter Spiele, wenn kein Trumpf als spielgewinnend berechnet wurde. Wird mindestens ein Trumpf berechnet, mit dem das Spiel gewinnbar ist, ist dieses Teil der rechten Spalte. Fall 4 beziffert den Anteil an diesen Spielen, die mit einem beliebigen gewinnbaren Trumpf klassifiziert wurden. Aus dieser Definition folgt, dass jede Klassifizierung von Spiel 2 aus Tabelle 3.2 mit $C(s) \neq \heartsuit$ und $C(s) \neq \emptyset$ diesem Fall zugeordnet wird.

Berechnung / Vorhersage	nicht gewinnbar	gewinnbar	
		richtiger Trumpf	falscher Trumpf
nicht gewinnbar	Fall 1 (positiv)	Fall 3 (negativ)	
gewinnbar	Fall 2 (negativ)	Fall 4 (positiv)	Fall 5 (negativ)

Abbildung 3.1: Maß für die Leistungsfähigkeit der Algorithmen

3.2 Der Double Dummy Skat Solver

Bislang blieb lediglich unklar, wie vorberechnete Resultate p der Zielfunktion $V^t(z)$ für Spielen z ermittelt werden, welche sowohl zur Evaluierung als auch zum Training des maschinellen Lernalgorithmus benötigt werden. Für diese Zwecke wird in dieser Studienarbeit der *Double Dummy Skat Solver* (DDSS) von S. Kupferschmid verwendet (Kupferschmid, 2003), dessen Funktionsweise deswegen kurz erläutert werden soll. Das Problem unvollständiger Information wird über einen Monte-Carlo-Ansatz auf Welten mit vollständiger Information abgebildet, die dann separat und für sich optimal durch eine verbesserte Minimax-Suche mit $\alpha\beta$-Pruning berechnet werden. Konkret wurde diese durch Transpositionstabellen, Äquivalenzklassen, Null-Fenster-Suche sowie Memory

Enhanced Test Driver erweitert, außerdem wurde eine sinnvolle Zuganordnung mithilfe einer Heuristik implementiert (Plaat u. a., 1995).

Der Monte-Carlo-Ansatz selbst erstellt eine geeignete, festgelegte Anzahl an, das bisherige Weltwissen nicht verletzenden, Spielzuständen vollständiger Information. Neben dem Wissen über die Karten der eigenen Hand können dabei während des Spiels weitere, ebenfalls gesicherte Informationen gesammelt werden, wie es zum Beispiel beim Nichtbedienen einer ausgespielten Farbe durch einen Spieler geschieht. Konkret erstellt Monte-Carlo also Spiele $z_i = \langle s, c_{1,i}, c_{2,i}, sk_i, t, pos \rangle$, wobei s, t sowie pos eindeutig festgelegt sind und dementsprechen für alle z_i gelten. Die restlichen Karten werden so auf $c_{1,i}$, $c_{2,i}$ und den Skat sk_i verteilt, dass der resultierende Spielzustand mit allen gesammelten Informationen konform ist. Aus allen für ein solches z_i berechneten Strategien gilt es dann, eine geeignete Lösung für das ursprüngliche Problem mit unvollständiger Information abzuleiten. Im DDSS wird dabei diejenige Karte gespielt, die die meisten Spiele gewonnen hat. Haben mehrere die selbe Anzahl an Spielen gewonnen, wird die Karte ausgewählt, die die höchste kumulative Punktzahl in diesen Spielen erreicht hat. S. Kupferschmid schlägt, analog zu M. Ginsberg's GIB, der auf der selben Methode beruht, 100 auf diese Weise konstruierte Spiele für jeden Spielzug vor. Der resultierende Skatspieler schlägt mit dieser Anzahl eine gute Approximation an die optimale Strategie vor. Schon mit nur halb so viel verwendeten Spielzuständen ist die Leistung des Programms aber deutlich schlechter. Dennoch darf nicht unerwähnt bleiben, dass auch mit 100 Samples pro Spielzug lediglich eine Annäherung berechnet wird und keineswegs optimales Spiel garantiert ist.

So werden zum Beispiel niemals solche Züge gemacht werden, die zum Sammeln von Informationen benötigt werden, da durch die Reduktion auf Welten vollständiger Information bereits alles Wissen vorhanden ist. Tatsächlich ergeben sich beim Skat aber durchaus Situationen, in denen es optimal wäre, einen Stich zu verlieren, falls dadurch Wissen hinzugewonnen werden kann. Es kann durch den Monte-Carlo-Ansatz auch geschehen, das unterschiedliche Strategien für den Erfolg im Einzelfall verantwortlich waren. Durch das Mischen von diesen kann unter Umständen eine suboptimale Strategie für das ursprüngliche Problem resultieren. Somit kann dieser Ansatz selbst bei erschöpfender Suche zu suboptimalen Zügen führen, wie schon von D. Basin und I. Frank aufgezeigt wurde (Frank und Basin, 1998). Trotz der beschriebenen Kritikpunkte ist der Monte-Carlo-Ansatz aber sehr geeignet, das Problem von Spielen mit unvollständiger Information gut zu lösen. Sowohl M. Ginsbergs GIS als auch S. Kupferschmids DDSS spielen auf einem hohen Niveau.

Die für meine Zwecke erforderliche Trainingsmenge besteht aus 10 000 voneinander verschiedenen, zufällig erstellten Spielen, die für jeden Trumpf t durch den DDSS berechnet wurden. Hinzu kommen 5 000 auf die selbe Weise generierte Spiele als Evaluierungsmenge. Da dadurch insgesamt 75 000 Spiele durch den DDSS berechnet werden mussten, konnte lediglich ein Wert von zehn Spielzuständen pro Spielzug und für jeden Spieler verwendet werden. Eine Berechnung mit 100 Samples hätte auf dem mir

zur Verfügung stehenden Rechner mehrere Wochen benötigt. Ausserdem ist die Verteilung der unbekannten Karten auf die Mitspieler beim Reizen weit weniger wichtig als es beim Ausspielen der Stiche der Fall ist. Dennoch muss ein gewisser propagierter Fehler erwartet werden, da die Zielfunktion $V^t(z)$ durch suboptimales Spiel in einigen Fällen unerwartete Ausreißer haben wird. Für 3 102 der Spiele der Evaluierungsmenge wurde durch den DDSS kein gewinnender Trumpf gefunden, diese bilden die linke Spalte aus Abbildung 3.1. Die restlichen 1 898 sind dementsprechend mit mindestens einem Trumpf gewinnbar. Da alle Spiele ohne Aufnahme des Skats als Handspiele berechnet wurden ist eine Verteilung von circa 60% ungewinnbaren zu 40% gewinnbaren Spielen durchaus realistisch.

4 Der Algorithmus der kleinsten Fehlerquadrate

4.1 Hintergrund

Auch S. Kupferschmid versuchte die Implementierung eines maschinellen Lernalgorithmus für das Reizen und schlug dafür die Implementierung des LMS-Algorithmus vor (Kupferschmid, 2003). Dieser gehört zur Gruppe der *überwachten Lernverfahren*, welchen gemein ist, dass zu jedem Zeitpunkt das gewünschte Ergebnis, also das der Zielfunktion $V^t(z)$, bereits bekannt sein muss, so dass der Lerner sich an dieses annähern kann. Dies ist durch die im vorangegangenen Abschnitt beschriebene Trainingsmenge gegeben, welche ab sofort durch D_1 beschrieben sei. Ziel des Algorithmus ist, den Fehler E zwischen den kalkulierten Trainingswerten, also der Zielfunktion $V^t(z)$, sowie den Werten der Bewertungsfunktion $\widehat{V}^t(s)$ zu minimeren, wobei gilt:

$$E \equiv \frac{1}{2} \sum_{\langle s,t,V^t(z)\rangle \in D_1} \left(V^t(z) - \widehat{V}^t(s) \right)^2 \tag{4.1}$$

Als Bewertungsfunktion kann eine beliebige Funktion gewählt werden, wobei die Wahl eines linearen Ansatzes

$$\widehat{V}^t(s) = w_0 + \sum_i \left(f_i^t(s) \cdot w_i \right) \tag{4.2}$$

üblich ist, bei dem $f_i^t(s)$ die *Merkmale* sind, die eine Anfrage s beschreiben und in Abschnitt 4.2 näher erläutert werden, und w_i die *Gewichte*, deren Wert durch LMS gelernt wird. Versucht wird also, eine geeignete Approximierung der Bewertungsfunktion an die Zielfunktion zu finden. Der Algorithmus beruht dabei auf der Methode des *steilsten Gradientenabstiegs* für einen durch die Gewichte definierten Vektor $\vec{w} = (w_0, w_1, ..., w_n)^T$. Dieser wird beim Start von LMS beliebig festgelegt und dann wiederholt derart modifiziert, dass er sich in kleinen Schritten in die Richtung verändert, die den größten Fehler hervorruft. Dieser Prozess läuft solange, bis ein globales Fehlerminimum erreicht wurde (Mitchell, 1997).

Zur Berechnung des steilsten Abstiegs in Richtung des Fehlers E werden die partiellen Ableitungen von E nach $w_0, w_1, ..., w_n$ benötigt, der sogenannte *Gradient* von E abhängig von $\vec{w}$:

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right] \tag{4.3}$$

Mit Gleichung 4.1 ergibt sich also für jede einzelne Komponente von 4.3:

$$\frac{\partial E}{\partial w_i} = \eta \sum_{\langle s, V_{train}(s)\rangle \in D} \left(V_{train}(s) - \widehat{V}(s) \right) (-f_i) \tag{4.4}$$

wobei η die sogenannte Lernrate darstellt.

Hieraus lässt sich nun die Aktualisierungsregel für alle Gewichte w_i des LMS-Algorithmus ableiten:

$$w_i \leftarrow w_i + \Delta w_i \tag{4.5}$$

wobei

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{4.6}$$

und damit mit Gleichung 4.5:

$$\Delta w_i = \eta \sum_{\langle s, V_{train}(s)\rangle \in D} \left(V_{train}(s) - \widehat{V}(s) \right) f_i \tag{4.7}$$

Somit ergibt sich der LMS-Algorithmus wie folgt:

```
LMS(D,η)
  initialize each weight w_i to some random number
  for each z = ⟨s, c_1, c_2, sk, t, pos⟩ ∈ D do
    use the current weights w_i to calculate V̂^t(s)
    error = V^t(z) − V̂^t(s)
    for each weight w_i do
      w_i ← w_i + (η · error · f_i^t)
```

Abbildung 4.1: LMS-Algorithmus

4.2 Parameter von LMS

Die wichtigsten Parameter des LMS-Algorithmus sind die gewählten Merkmale $f_i^t(s)$. Diese sollen einen Zustand bestmöglichst beschreiben, es ist also wichtig, dass sie sehr aussagekräfig gewählt werden. Dann ist auch eine gute Approximation der Zielfunktion möglich und somit eine gute Vorhersage des Resultats. Dennoch muss auch beachtet werden, dass mehr Trainingsdaten benötigt werden, je ausdruckstärker die Merkmale gewählt werden. Es gilt also, diejenigen Eigenschaften einer Anfrage s zu finden, die für den Gewinn eines Spieles am wichtigsten sind, auf zu detaillierte Beschreibungen von s aber zu verzichten. Im Skatspiel haben Trümpfe und Buben wohl den grössten Einfluss auf den Ausgang eines Spieles und damit auch auf die Höhe des Gebotes beim Reizen und müssen ausreichend in die Merkmale einfliessen, damit für Hände mit vielen dieser Karten eine höhere Bewertung erfolgt. Allerdings gibt es mehrere Möglichkeiten, diese zu repräsentieren: Deren Anzahl mag bereits genügen, es ist aber auch möglich, dass auch Rang und Punktewert einfliessen sollten. Weitere wichtige Karten sind Asse und Zehner. Als Startpunkt für diese Studienarbeit entschied ich mich, die auch von S. Kupferschmid für diesen Ansatz verwendeten und in Tabelle 4.1 dargestellten Merkmale

zu implementieren, bei denen lediglich die Anzahl in die Merkmale einfliesst (Kupferschmid, 2003).

Merkmal	**Beschreibung**	**Wertebereich**
$f_1^t(s)$	Anzahl der Buben in s	0–4
$f_2^t(s)$	Anzahl der Trümpfe in s	0–11(Grand:0–4)
$f_3^t(s)$	Anzahl der Asse und 10er in s	0–8
$f_4^t(s)$	Gesamtpunktzahl der Karten in s	0–92
$f_5^t(s)$	Anzahl verschiedener Farben in s	1–4
$f_6^t(s)$	Anzahl blanker 10er in s	0–3

Tabelle 4.1: Merkmale M_1 nach S. Kupferschmid

Die Merkmale M_1 werden analog zu Tabelle 4.1 extrahiert und dann mithilfe des angegebenen Wertebereichs noch auf das Intervall $[0;1]$ normiert. Auch die verwendete Zielfunktion $V^t(z)$ wird auf diese Intervall normiert und ergibt sich als

$$V^t(z) = \frac{\text{durch DDSS berechnete Punktzahl}}{120} \tag{4.8}$$

Als Bewertungsfunktion $\widehat{V}^t(s)$, durch die $V^t(z)$ approximiert werden soll, wird der einfache, lineare Ansatz aus Gleichung 4.2 verwendet:

$$\widehat{V}^t(s) = \sum_{i=1}^{6} f_i^t(s) \cdot w_i \tag{4.9}$$

Es bleibt noch zu klären, ob für jeden Trumpf t ein eigener Gewichtsvektor $\vec{w}^t$ gelernt wird, oder ob eine gute Approximation auch mit einem einzigen, trumpfunabhängigen Gewichtsvektor $\vec{w}$ möglich ist. Für alle Farbspiele ist dabei ein einzelner Gewichtsvektor $\vec{w}^F$ aussreichend, da die Farben symmetrisch zueinander sind. Sie sind beliebig austauschbar untereinander, sofern beide nicht zur Trumpffarbe gehören bzw. wenn eine zur Trumpffarbe gehört und der Trumpf ebenfalls getauscht wird: Eine Hand ♡A ♡9 wird die selbe Auswirkung beim Reizen auf Trumpf ♡ (oder auch Trumpf ♣) haben wie die Hand ♢A ♢9 auf Trumpf ♢ (bzw. ♣). Dementsprechend wird auch der gelernte Gewichtsvektor $\vec{w}^F$ in beiden Fällen identisch sein. Anders verhält es sich bei den Grandspielen, jedes Merkmal kann einen anderen Einfluss auf den Ausgang eines Spieles haben als in Farbspielen. Es werden also zwei voneinader unabhängige Gewichtsvektoren $\vec{w}^F$ für Farbspiele und $\vec{w}^G$ für Grandspiele gelernt.

Wie in Abbildung 4.1 beschrieben wurden diese zufällig initialisiert, der Algorithmus wurde aber auch auf festgelegte Startgewichte angewendet. Auch die Lernrate η wurde variiert, nach einigen Versuchen erwies sich ein Wert von $\eta = 0,001$ als geeignet. Der

Lerner wurde mehrfach mit Initialgewichten im Intervall $[0;1]$ oder $[-1;1]$ gestartet. Unabhängig von diesem konnte dabei eine Tendenz erkannt werden, es ergaben sich aber bei extremen Startgewichten auch größere Abweichungen. Der Durchschnitt der gelernten Gewichte, der auch ein typisches Ergebnis beschreibt, ist in Tabelle 4.2 abgebildet.

	$f_1^t(s)$	$f_2^t(s)$	$f_3^t(s)$	$f_4^t(s)$	$f_5^t(s)$	$f_6^t(s)$
$\vec{w}^F$	0,30	0,41	0,35	0,22	−0,03	−0,02
$\vec{w}^G$	0,42	0,39	0,34	0,26	−0,06	−0,04

Tabelle 4.2: Mittelwerte der gelernten Gewichte mit Merkmalen M_1

Mit diesen ermittelten Gewichtsvektoren wurden dann die Spiele der Evaluierungsmenge über die im vorigen Kapitel in Gleichung 3.3 beschriebene Funktion $C(s)$ klassifiziert. Auch die Ergebnisse der Evaluation variierten je nach verwendeten Gewichten stark, insbesondere bei den gewinnbaren Spielen ergaben sich Unterschiede von bis zu 10%. Insgesamt lassen die in Abbildung 4.2 dargestellten Resultate noch Platz für Verbesserungen, egal ob feste oder zufällige Gewichtsvektoren zu Beginn gewählt wurden, was sich auch mit S. Kupferschmids Ergebnissen deckt.

Berechnung / Vorhersage	nicht gewinnbar	gewinnbar	
		richtiger Trumpf	falscher Trumpf
nicht gewinnbar	91,90%	58,26%	
gewinnbar	8,10%	26,48%	15,26%

Abbildung 4.2: Ergebnisse von LMS mit Merkmalen M_1

Als nicht gewinnbar berechnete Spiele werden sehr gut erkannt, da allerdings annähernd 80% aller Spiele mit $C(s) = \emptyset$ klassifiziert werden ist ein hoher Wert für diesen Fall auch zu erwarten. Bei den gewinnbaren Spielen dagegen ist die Fehlerrate zu hoch, lediglich jedes vierte solche Spiel wird korrekt eingeordnet. Insgesamt wird über die Bewertungsfunktion $\widehat{V}^t(s)$ des LMS-Reizers 67,07% der Spiele die richtige Klasse zugewiesen. Auf den ersten Blick erscheint dies ordentlich, da aber ein Algorithmus, der alle Spiele als ungewinnbar betrachtet, auch schon mehr als 60% aller Spiele dieser Evaluierungsmenge korrekt klassifiziert, kann dieses Ergebnis nicht zufriedenstellend sein. Auch eine mittlere quadratische Abweichung von 293,65 lässt darauf schliessen.

Eine Möglichkeit, bessere Resultate zu erzielen, sehe ich vor allem in der wichtigen Wahl der Merkmale. Zwar werden durch M_1 viele Aspekte einer Hand abgedeckt, dennoch sollten mit einigen wenigen Änderungen aussagekräftigere Beschreibungen eines Zustandes möglich sein, ohne den Lernaufwand dadurch zu erhöhen.

4.3 Entwicklung der Merkmale M_2

Bei Betrachtung der Beschreibung der Merkmale $f_i^t(s)$ aus Tabelle 4.1 fällt auf, dass ihr Wert für eine Hand s, abgesehen von $f_2^t(s)$, für alle Trümpfe der selbe ist. So gilt zum Beispiel für jede Anfrage s, dass $f_3^G(s) = f_3^{\clubsuit}(s) = f_3^{\spadesuit}(s) = f_3^{\heartsuit}(s) = f_3^{\diamondsuit}(s)$. Die Merkmale sind also zwar geeignet, etwas über die allgemeine Qualität einer Hand auszusagen, nicht aber darüber, welche Farbe die stärkste ist, und auf welches Spiel am ehesten gereizt werden soll. Ändert man die Definition von $f_3^t(s)$ so, dass dabei lediglich die Anzahl der Zehner und Asse gezählt wird, die nicht zur Trumpffarbe gehören, kann dieses Merkmal für verschiedene Trümpfe bereits unterschiedliche Werte annehmen. Ein weiterer Vorteil ist dann, dass Ass und Zehn der Trumpffarbe nicht mehr doppelt gezählt werden — diese sind bereits in $f_2^t(s)$ repräsentiert. Argumentiert man, dass diese Karten einen stärkeren Einfluss auf das Spiel haben, ist das sicherlich richtig. Dies soll aber über die gelernten Gewichte in die Bewertungsfunktion $\widehat{V}^t(s)$ eingehen, und nicht über eine mehrfache Berücksichtigung in den Merkmalen. Dennoch soll hier angemerkt werden, dass auch für die durch diese Änderungen entstehenden Merkmale M_2 Mehrfachrepräsentierungen einzelner Karten bestehen bleiben.

Eine weitere Schwäche von M_1 ist die, dass lediglich die Anzahl der Buben beziehungsweise der Trümpfe für den Gewinn eines Spieles eine Rolle zu spielen scheint. Dies ist für das Skatspiel aber zu allgemein gehalten, wie folgendes Beispiel zweier Anfragehände zeigt:

Hand 1	♣J	♠J	♣K	♣Q	♣9	♠10	♡10	♡8	♡7	♢A
Hand 2	♡J	♢J	♣9	♣8	♣7	♠10	♡10	♡K	♡Q	♢A

Tabelle 4.3: Hände mit identische Merkmalen M_1

Beide Hände werden in Bezug auf Trumpf ♣ durch den selben (noch nicht normierten) Merkmalsvektor $\langle 2; 5; 3; 42; 4; 1\rangle^T$ beschrieben, ein Skatspieler wird aber sofort Hand 1 als die Stärkere bezeichnen. Eine Unterscheidung der beiden Fälle ist dem Reizer mit M_1 aber nicht möglich, weshalb die selbe Punktzahl aus $\widehat{V}^t(s)$ resultieren wird. Wie bereits gesehen entstehen bereits Probleme durch unvollständige Information. In diesem Fall ist aber nicht diese, sondern die nicht ausreichende Ausnutzung des Wissens das Problem. Da der Rang von Buben und Trümpfen eine wichtige Rolle für den Gewinn eines Spieles spielt, werden in M_2 deshalb die zusätzlichen Merkmale $f_7^t(s)$ und $f_8^t(s)$ eingeführt.

Ähnlich lässt sich auch für die Nichttrumpffarben argumentieren. Durch die bereits beschriebene Änderung der Merkmale $f_3^t(s)$ und $f_6^t(s)$ werden die wichtigsten Karten des Beiblattes, nämlich die Asse und Zehner, aber bereits ausreichend stark repräsentiert, und auch $f_4^t(s)$ hängt stark von diesen ab. Um die Komplexität nicht zu hoch werden zu lassen und somit eine größere Trainingsmenge zu benötigen, wird deshalb auf ein solches Merkmal verzichtet.

Zusammenfassend ergeben sich aus diesen Überlegungen und weiteren, kleineren Änderungen, die in Tabelle 4.4 beschriebenen Merkmale M_2. Die Normierung auf das Intervall $[0; 1]$ wurde selbstverständlich beibehalten.

Merkmal	**Beschreibung**	**Wertebereich**
$f_1^t(s)$	Anzahl der Buben in s	0–4
$f_2^t(s)$	Anzahl der Trümpfe in s, ohne Buben	0–7(Grand:0)
$f_3^t(s)$	Anzahl der Beiblatt-Asse sowie der 10er in s. 10er werden nur gezählt, falls das gleichfarbige Ass in s ist.	0–6(Grand:0–8)
$f_4^t(s)$	Gerundete Gesamtpunktzahl der Karten in s (auf %5)	0–90
$f_5^t(s)$	Anzahl der Fehlfarben in s	0–3
$f_6^t(s)$	Anzahl blanker 10er in den Nichttrumpffarben in s	0–3
$f_7^t(s)$	Bewertung der Buben in s	0–10*
$f_8^t(s)$	Bewertung der Trümpfe in s, ohne Buben	0–17(Grand:0)**

* ♣J : 4, ♠J : 3, ♡J : 2, ◇J : 1
** A : 5, 10 : 4, K : 3, Q : 2, 9,8 und 7: 1

Tabelle 4.4: Merkmale M_2

Mit M_2 wurden identische Berechnungen durchgeführt. Die Anzahl der Buben und Trümpfe relativ zu den anderen Gewichten zeigte sich dabei als etwas weniger wichtig, dafür sind diese aber zusätzlich in den Gewichten w_7 und w_8 repräsentiert. Die Ergebnisse der Klassifikation variierten etwas weniger stark als dies mit M_1 der Fall war. Das Resultat mit einem der ermittelten Gewichtsvektoren ist in Abbildung 4.3 dargestellt.

Berechnung / Vorhersage	nicht gewinnbar	gewinnbar	
		richtiger Trumpf	falscher Trumpf
nicht gewinnbar	97,64%	69,76%	
gewinnbar	2,36%	20,60%	9,64%

Abbildung 4.3: Ergebnisse von LMS mit Merkmalen M_2

Zwar werden noch mehr ungewinnbare Spiele auch so klassifiziert, aber auch die Gesamtzahl an als spielbar eingeschätzten Händen ist weiter gesunken: Insgesamt werden nur noch 12,94% aller Spiele nicht durch $C(s) = \emptyset$ beschrieben, obwohl nur drei von fünf Spielen tatsächlich dieser Klasse angehören. Wird ein Spiel aber als gewinnbar erkannt, wird es häufiger auch mit dem richtigen Trumpf klassifiziert. Zumindest zur Einschätzung, welche Farbe die stärkste auf der Hand ist, scheint M_2 also besser geeignet zu sein. Insgesamt ist der Anteil korrekt klassifizierter Spiele aber mit 68,39% nur unwesentlich gestiegen. Wünschenswert wäre insbesondere, gewinnbare Spiele mit einer besseren Rate richtig zu klassifizieren. Lediglich die Betrachtung des durchschnittlichen quadratischen Fehlers deutet eindeutig darauf hin, dass M_2 leistungsstärker ist als M_1: Dieser ist um beinahe 50 Punkte auf 246,67 gesunken.

Eine andere These, dass keine besseren Ergebnisse erzielt werden konnten, ist die, dass die Bewertungsfunktion $\widehat{V}^t(s)$ nicht optimal gewählt wurde. Hierfür wurde eine simple, lineare Funktion gewählt. Ist aber die Zielfunktion $V^t(z)$ zu komplex, wird auch eine komplexe Bewertungsfunktion benötigt, um diese gut zu approximieren. Es ist also durchaus möglich, dass für das Skatspiel keine Bewertungsfunktion existiert, die simpel genug ist und trotzdem Ergebnisse liefert, die denen der Zielfunktion nahe kommen. Auch wenn diese existiert können mögliche Ausreißer durch die im vorigen Kapitel beschriebene Funktionsweise des DDSS die resultierende Zielfunktion zu kompliziert machen. In jedem Fall kann gesagt werden, dass die gewählte lineare Funktion nicht zu einer guten Approximierung geeignet ist. Deshalb soll ein gänzlich anderer Ansatz beschrieben werden, der das Problem der komplexen Zielfunktion komplett umgeht.

5 Der k-Nächste-Nachbar-Algorithmus

5.1 Hintergrund

Der k-Nächste-Nachbar Algorithmus folgt einer völlig anderen Idee als LMS, nämlich der instanzenbasierter Lernalgorithmen. Diesen ist gemein, dass Trainingsbeispiele in einer Wissensbasis gespeichert werden, und Anfragen mit diesen Daten verglichen werden, um dann durch ähnliche, bereits bekannte Instanzen klassifiziert zu werden. Der wichtigste Unterschied zu Verfahren wie LMS ist also, dass nicht versucht wird, eine Zielfunktion $V^t(z)$ durch eine Bewertungsfunktion $\widehat{V}^t(s)$ zu approximieren, wodurch das Aufstellen einer konkreten Bewertungsfunktion umgangen wird. Es ergibt sich aber auch ein Nachteil: Instanzenbasierte Lernalgorithmen nehmen zur Klassifizierung verhältnismäßig viel Zeit in Anspruch, da beinahe alle Berechnungen in Folge auf eine Anfrage stattfinden und nicht, wie zum Beispiel auch bei LMS, während eines separaten Lernvorgangs.

Die einfachste, instanzenbasierte Methode ist der k-Nächste-Nachbar-Algorithmus KNN. Dieser betrachtet alle Instanzen, die durch n Merkmale beschrieben sind, als Punkte im n-dimensionalen Raum $\mathbb{R}^n$. Analog zum vorigen Kapitel wird eine Instanz s in diesem Raum beschrieben durch ihren Merkmalsvektor

$$\left\langle f_1^t(s), f_2^t(s), ..., f_n^t(s) \right\rangle \tag{5.1}$$

Der Abstand zweier Punkte s und z voneinander ist definiert über deren euklidische Distanz $d(s, z)$ in Bezug auf den beschreibenden Merkmalsvektor, wobei

$$d(s, z) = \sqrt{\sum_{i=1}^{n} (f_i^t(s) - f_i^t(z))^2} \tag{5.2}$$

In der Literatur wird eine Anfrage s durch ihre k naheliegendsten Nachbarn $z_1, ..., z_k$ aus einer Wissensbasis klassifiziert. Für meine Belange ist diese Definition allerdings nicht ausreichend: Der Ansatz funktioniert nur, falls die Funktionen $f_i^t(z)$, welche die Merkmale extrahieren, bijektiv und somit eineindeutig definiert sind. Bei keinem der gewählten Merkmalsvektoren ist dies aber der Fall: Wie bereits aufgezeigt wurde existieren zu jedem Merkmalsvektor sehr viele mögliche Hände. Also ist es auch möglich, mehrere Instanzen in einer Wissensbasis zu finden, die die selbe euklidische Distanz zu einer Anfrage haben. Die Definition muss also so erweitert werden, dass eine Anfrage s klassifiziert wird durch k Mengen von nächsten Nachbarn Z_i, wobei alle Elemente $z_{i,j}$ einer solchen Menge den selben euklidischen Abstand $d(s, z_{i,j})$ zu s haben. Hieraus leiten sich weitere Modifikationen ab, die an den entsprechenden Stellen dieser Studienarbeit beschrieben werden.

Wie genau aus den Nachbarn eine Klassifizierung abgeleitet wird ist weitgehend offen. Für einen diskreten Raum gibt es zum Beispiel die Möglichkeit, den am häufigsten ermittelten Wert der Bewertungsfunktion unter den Nachbarn als Klassifizierung zu

verwenden. Dies ist eine Möglichkeit, eine Klassifizierung auf der Bewertungsfunktion $\widehat{V}_0^t(s)$ aus Gleichung 3.1 zu berechnen. Zwar ist auch die Bewertungsfunktion $\widehat{V}^t(s)$ aus Gleichung 3.2 über dem diskreten Raum der natürlichen Zahlen in $[0; 120]$ definiert, zu zählen wie oft ein bestimmter Wert erreicht wird erscheint aber wenig sinnvoll. Die Bewertungsfunktion als den Durchschnitt der Resultate der Zielfunktion $V^t(z_i)$ der verwendeten Nachbarn z_1 aus der Wissensbasis zu berechnen und dann über $C(s)$ aus Gleichung 3.3 zu klassifizieren erscheint einleuchtender. Eine andere Möglichkeit wäre es, lediglich für jeden Trumpf zu zählen, wie oft das Spiel gewinnbar ist, und dann denjenigen zu wählen, der am häufigsten gewinnt. Allerdings werden so nicht alle möglichen Informationen genutzt. Im nächsten Abschnitt werden unter anderem vier unterschiedliche Bewertungsfunktionen miteinander verglichen.

Anschaulich wird der KNN-Algorithmus in Abbildung 5.1 dargestellt. Links wird die 5-Nächste-Nachbar Klassifizierung einer Anfrage x_q im $\mathbb{R}^2$ mit den Klassen $+$ und $-$ illustriert. Dabei würde x_q als $-$ klassifiziert werden, da die meisten der 5-nächsten Nachbarn ebenfalls dieser Klasse angehören. Eine Schwäche von KNN ist hier bereits ersichtlich: Ein 1-Nächster-Nachbar würde x_q als $+$ einordnen, da der nächste aller Nachbarn dieser Klasse angehört. Einen guten Wert für die Anzahl der verwendeten Nachbarn zu ermitteln ist ebenfalls Teil des nächsten Abschnittes. Auf der rechten Seite ist die Aufteilung des Raumes $\mathbb{R}^2$ in Klassen für einen 1-Nächsten-Nachbar-Algorithmus abgebildet, ein sogenanntes Voronoi-Diagramm. Anfragen werden hier immer genauso klassifiziert, wie die durch einen Punkt dargestellte, bereits berechnete Instanz innerhalb des Polygons, in dem auch die Anfrageinstanz liegt.

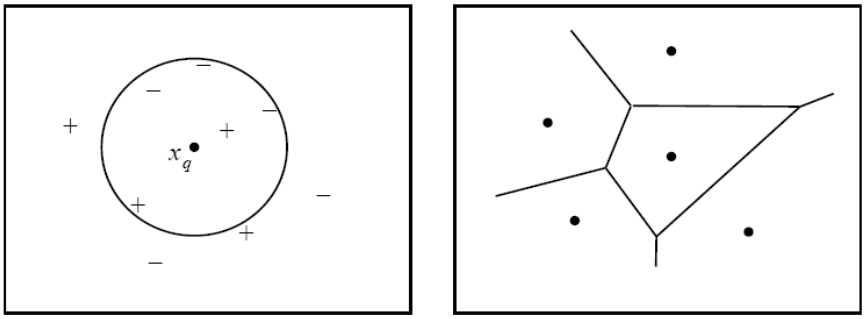

Abbildung 5.1: KNN aus: Mitchell 1997

Der KNN-Algorithmus zur Klassifizierung einer Anfrage s auf einer Wissensbasis D, bestehend aus Trainingsinstanzen z_i, ergibt sich somit wie folgt:

```
KNN(s,D,k)
  let Z_1,...,Z_k be the k sets of nearest neighbors to s in D
  classificate s based on Z_1,...,Z_k by use of
        any evaluation function V̂^t(s,Z_1,...,Z_k)
```

Abbildung 5.2: KNN-Algorithmus

Durch diese Definition wird lediglich der grundsätzliche Ansatz von KNN dargestellt, beide Teile des Algorithmus bieten Spielraum für Variationen, die in den folgenden Abschnitten und Kapitel 6 näher untersucht werden. Als Wissensbasis werden zur besseren Vergleichbarkeit mit LMS die selben Spiele benutzt, auf denen dieser trainiert wurde, also D_1.

5.2 Vergleich unterschiedlicher Parameter und Methoden

In diesem Abschnitt sollen vier grundlegende Erkenntnisse über den KNN-Algorithmus gewonnen werden: Erstens ein genereller Überblick über die Qualität des Algorithmus, insbesondere im Vergleich zu LMS. Zweitens soll untersucht werden, ob ein Wertebereich für die Anzahl der relevanten Nachbarmengen k exisitiert, in dem die beste Klassifizierungsleistung erreicht wird. Da durch den LMS-Algorithmus keine Klarheit über die Qualität der Merkmalsvektoren M_1 und M_2 gewonnen werden konnte, sollen drittens diese beiden erneut gegenübergestellt werden. Und schliesslich sollen vier unterschiedliche Bewertungsfunktionen $\widehat{V}^t(s, Z_1, ..., Z_k)$ miteinander verglichen werden, mit denen aus den Nachbarn eine Klassifizierung für eine Anfrage s hergeleitet werden kann. Trotz ihrer Abhängigkeit von $Z_1, ..., Z_k$ seien diese weiterhin durch $\widehat{V}^t(s)$ beschrieben.

1. **Count** ($\widehat{V}^t_{CO}(s)$) Jedem Trumpf t wird die Anzahl der Nachbarn aus $\{Z_1, ..., Z_k\}$ zugewiesen, die ihr Spiel $z_{i,j}$ gewinnen. $\widehat{V}^{\emptyset}_{CO}(s)$ ist gleich der Anzahl an Spielen, die von keinem Trumpf gewonnen werden. Für diese Bewertungsfunktion wird auch eine alternative Klassifizierungsmethode $C_{CO}(s)$ benötigt: Eine Anfrage s wird durch den Trumpf t oder durch $\emptyset$ klassifiziert, der den höchsten Wert in der Bewertungsfunktion erzielt.

2. **Average** ($\widehat{V}^t_{AV}(s)$) Jedem Trumpf t wird der Durchschnitt aller Ergebnisse $V^t(z_{i,j})$ in $\{Z_1, ..., Z_k\}$ zugewiesen. Zur Klassifizierung wird $C(s)$ aus Gleichung 3.3 verwendet.

3. **Median** ($\widehat{V}^t_{ME}(s)$) Jedem Trumpf t wird der Median der Ergebnisse $V^t(z_{i,j})$ in $\{Z_1, ..., Z_k\}$ zugewiesen. Zur Klassifizierung wird $C(s)$ aus Gleichung 3.3 verwendet.

4. **Average of Median** ($\widehat{V}^t_{AM}(s)$) Dies ist eine Mischung der beiden vorherigen Bewertungsfunktionen $\widehat{V}^t_{AV}(s)$ und $\widehat{V}^t_{ME}(s)$. Es wird der Durchschnitt aus der mittlere Hälfte aller sortierten Resultate $V^t(z_{i,j})$ für jeden Trumpf t gebildet und der entsprechenden Bewertungsfunktion zugewiesen. Zur Klassifizierung wird $C(s)$ aus Gleichung 3.3 verwendet.

Allen Methoden ist dabei gemein, dass kein Unterschied gemacht wird, welchem Z_i eine Instanz $z_{i,j}$ angehört. Die beschriebene Bewertungsfunktion wird über allen Instanzen aller Nachbarsmengen berechnet.

Im Folgenden wird die genaue Beschreibung einer Klassifizierung durch $KNN(m, c, k)$ angegeben, wobei $m \in \{M_1, M_2\}$ für die verwendeten Merkmale steht, wie sie in Kapitel 4 definiert wurden, $c \in \{CO; AV; ME; AM\}$ für die verwendete Bewertungsfunktion $\widehat{V}^t_c(s)$ und k für die Anzahl der in das Ergebnis einfliessenden Nachbarmengen. Der Reizer wurde auf allen Kombinationen aus den angegebenen m, c sowie für $k \in \{1; 3; 5; 8; 15; 30; 50\}$ auf der Evaluierungsmenge getestet.

Da insgesamt 56 verschiedene Klassifizierungen miteinander verglichen werden, können nicht alle Ergebnisse kompakt und anschaulich dargestellt werden. Auf eine Darstellung der detaillierten Resultate der verschiedenen Bewertungsfunktionen wird deswegen verzichtet. Eine Analyse der Ergebnisse zeigt, dass der Einfluss dieser Funktionen auf das Resultat vernachlässigbar ist. Keine der verwendeten Methoden schneidet nennenswert besser oder schlechter ab als die anderen, die Unterschiede liegen unter einem Prozent. Insbesondere ist keine Tendenz erkennbar, dass eine Methode einer anderen in bestimmten Wertebereichen für k überlegen ist. Bei folgenden Klassifizierungen soll deswegen immer $\widehat{V}^t_{AV}(s)$ verwendet werden. Der Durchschnitt ist etwas schneller zu berechnen als $\widehat{V}^t_{ME}(s)$ und $\widehat{V}^t_{AM}(s)$, da für diese die Nachbarmengen geordnet werden müssen, und im Gegensatz zu $\widehat{V}^t_{CO}(s)$ ist der mittlere quadratische Fehler definiert, was der Analyse von Ergebnissen ein zusätzliches Werkzeug gibt. Der bedeutendste Nachteil des Durchschnitts gegnüber dem Median, dass einzelne Ausreisser nach oben oder unten das Ergebnis verzerren, scheint zudem keine Rolle zu spielen. Eine Analyse der Varianz der Ergebnisse in den Nachbarmengen Z_i bestätigt dies. Im folgenden wird der Index AV bei der Bezeichnung der Bewertungsfunktion deswegen wieder fallengelassen.

In Abbildung 5.3 sind die Ergebnisse der verbleibenden 14 Klassifizierungen veranschaulicht. Der erste Teil jedes Balkens zeigt den Anteil korrekt klassifizierter, ungewinnbarer Spiele, der zweite den korrekt klassifizierter, gewinnbarer Spiele. Diese sind bereits analog der Verteilung von gewinnbaren zu ungewinnbaren Spielen in der Evaluierungsmenge gewichtet, die Summe dieser Abschnitte zeigt also den Anteil korrekt klassifizierter Spiele insgesamt. Die restlichen Teile eines jeden Balken stellen von links nach rechts die Fälle 3, 5 und 2 aus Abbildung 3.1 dar.

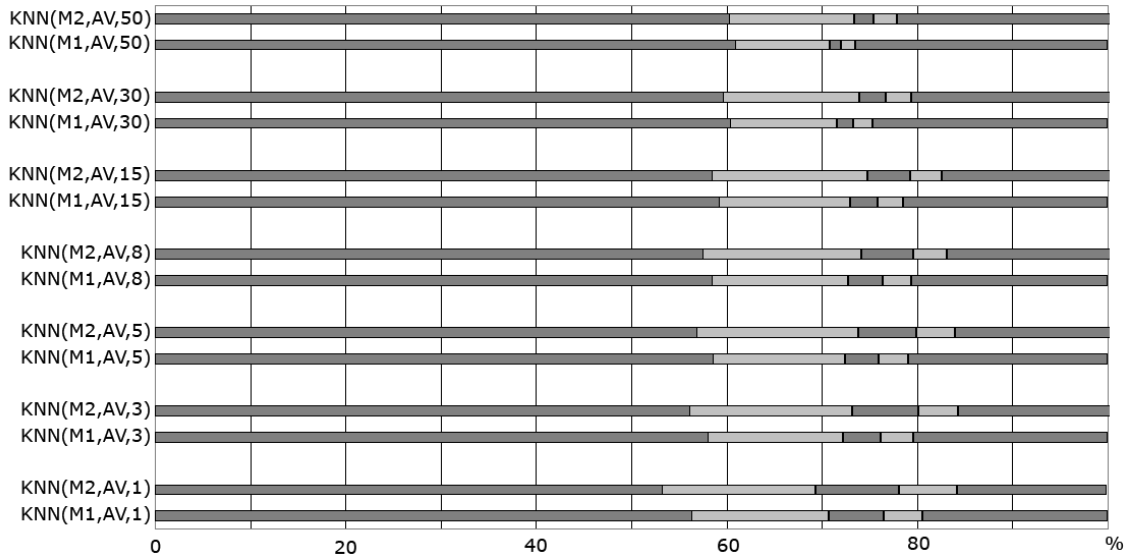

Abbildung 5.3: KNN-Ergebnisse ($m \in \{M_1; M_2\}$, $c = AV$, $k \in \{1; 3; 5; 8; 15; 30; 50\}$)

Zwei aufeinander folgene Balken bilden Paare mit identischer Anzahl verwendeter Nachbarn k. Der untere zeigt jeweils die durch Verwendung der Merkmale M_1 gewonnenen Resultate, der obere diejenigen von M_2. Ausser für $k = 1$ ist das Gesamtergebnis in allen Fällen unter Verwendung von M_2 das Bessere. Zwar ist M_1 geeigneter, ungewinnbare Spiele korrekt zu klassifizieren, die Ergebnisse in diesem Fall gleichen sich bei steigendem k aber an. In der Klassifizierung gewinnbarer Spiele dagegen werden mit M_2 mindestens 6% bessere Ergebnisse für alle k erzielt, woraus die besseren Gesamtergebnisse resultieren. Ein Blick auf die mittleren quadratischen Fehler in Abbildung 5.4 bestätigt die besseren Resultate von M_2 zusätzlich.

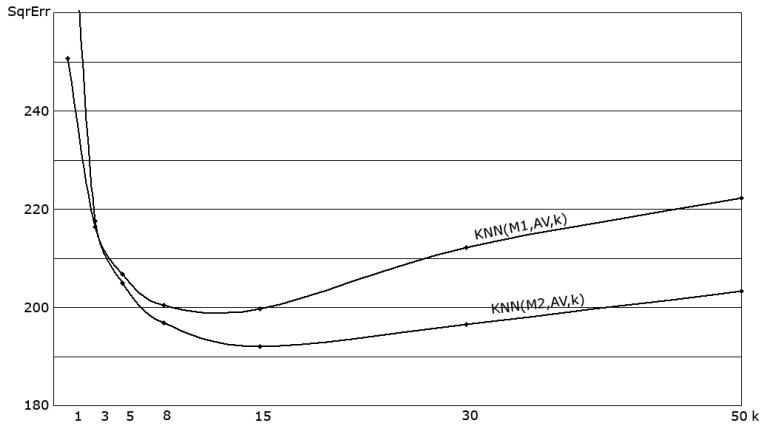

Abbildung 5.4: Mittlerer quadratischer Fehler von KNN für M_1 und M_2

Aus den Abbildungen 5.3 und 5.4 kann eine weitere wichtige Erkenntnis über KNN gezogen werden: Es scheint ein Optimum oder zumindest ein optimaler Wertebereich für die Anzahl verwendeter Nachbarn k zu existieren. Im Intervall $]8;30[$ kann sowohl der geringste mittlere quadratische Fehler gemessen als auch das beste Gesamtergebnis erzielt werden. Zwar ist die Anzahl korrekt klassifizierter, ungewinnbarer Spiele für größere k höher und die korrekt klassifizierter, gewinnbarer Spiele für kleinere k, dies liegt aber daran, dass der Reizer bei wenigen verwendeten Nachbarn risikoreicher reizt und bei steigender Anzahl vorsichtiger wird.

Insgesamt ist im Vergleich zu LMS eine deutliche Verbesserung zu erkennen. Selbst das schlechteste Gesamtergebnis bei $KNN(M_2, AV, 1)$ liegt mit 69,98% noch knapp über dem Ergebnis aus den beiden Implementierungen aus Kapitel 4. Das beste Gesamtergebnis des KNN-Algorithmus liegt mit 74,2% richtig klassifizierten Spielen bei $KNN(M_2, AV, 15)$ sogar schon deutlich darüber. Auch die Teilergebnisse sprechen dafür: Bei ungewinnbaren Spielen sind die Ergebnisse ähnlich, Gewinnbare werden mit KNN beinahe doppelt so häufig korrekt eingeordnet. Ebenso deutlich ist die Verbesserung des mittleren quadratischen Fehlers auf 192,07 im besten Fall. Eine tiefgehendere Untersuchung instanzenbasierter Algorithmen erscheint also vielversprechend. Dafür werden im Folgenden, aufgrund der gewonnenen Erkenntnisse, Klassifizierungen mit den Parametern $c = AV$, $m = M_2$ sowie $k \in \{10; 15; 20; 25\}$ verwendet. Zur besseren Vergleichbarkeit mit anderen Implementierungen sind in Abbildung 5.5 die Ergebnisse von $KNN(M_2, AV, 15)$ detailliert dargestellt.

Berechnung / Vorhersage	nicht gewinnbar	gewinnbar	
		richtiger Trumpf	falscher Trumpf
nicht gewinnbar	94,22%	50,00%	
gewinnbar	5,78%	41,47%	8,53%

Abbildung 5.5: Ergebnisse von $KNN(M_2, AV, 15)$

Eine Schwäche von LMS bestand darin, dass insgesamt zu viele Spiele als ungewinnbar betrachtet werden, und auch wenn KNN etwas seltener mit $C(s) = \emptyset$ klassifiziert, werden noch immer drei von vier Spielen so eingeordnet. In Kapitel 3 wurde zur Klassifizierung ein Schwellenwert th eingeführt, der bislang mit $th = 60$ nicht zufällig gewählt wurde: Wird beim Ausspielen der Stiche mindestens diese Punktzahl erreicht, ist das Spiel gewonnen. Da die fehlerfreie Vorhersage dieser Punktzahl aufgrund der unvollständigen Information aber nicht möglich ist, könnte eine Verschiebung dieses Wertes die Resultate verbessern, falls die Vorhersage zu Über- oder Unterschätzung neigt. Die Ergebnisse durch Klassifizierung mit variierendem Schwellenwert $th \in [55; 62]$ für $KNN(M_2, AV, 15)$ sind Abbildung 5.6 zu entnehmen.

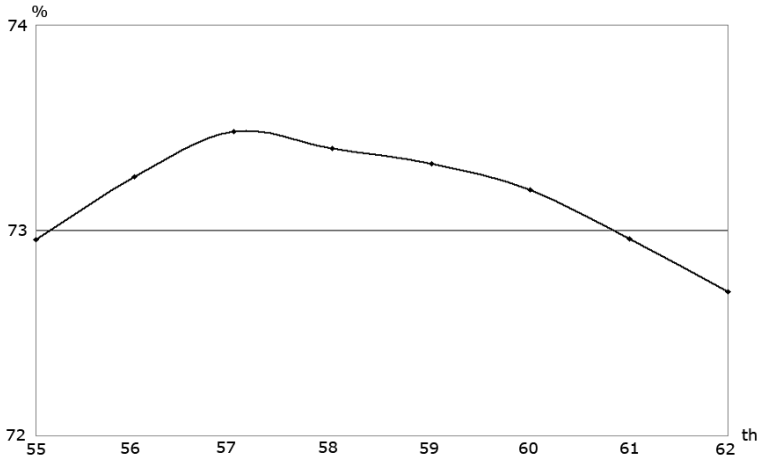

Abbildung 5.6: Ergebnisse für $KNN(M_2, AV, 15)$ und verschiedene Schwellenwerte th

Das Optimum richtig klassifizierter Spiele liegt tatsächlich nicht bei $th = 60$ sondern mit $th = 57$ etwas darunter, was die These, dass auch KNN zu Unterschätzung neigt, bestätigt. Auch wenn der Unterschied im Optimum des Gesamtresultats klein erscheint — mit 74,48% liegt dieses lediglich 0,28% über dem Ergebnis mit dem anfänglichen Schwellenwert — lohnt eine genauere Betrachtung. Ein kleinerer Schwellenwert führt dazu, dass weniger Spiele als ungewinnbar und mehr als gewinnbar klassifiziert werden. Es werden sich also nur Verbesserungen bei den gewinnbaren Spielen ergeben. Da der Anteil dieser in der Evaluierungsmenge kleiner ist, steigt der Anteil der korrekt klassifizierten gewinnbaren Spiele proportional höher als der jetzt falsch klassifizierter Ungewinnbarer fällt. Bislang werden aber nur Handspiele betrachtet, bei denen der Skat unbeachtet bleibt. Wenn später in dieser Arbeit der Algorithmus aber darauf ausgeweitet wird, dass der Skat auch aufgenommen wird und die Hand dadurch in vielen Fällen stärker wird, wird der Anteil gewinnbarer Spiele in der Evaluierungsmenge stark ansteigen. Mehr Spiele als gewinnbar zu klassifizieren wird dann, sofern eine gute Trumpfwahl getroffen wurde, von Vorteil sein.

Denoch werde ich mich weiter auf eine Optimierung der Klassifizierung mit $th = 60$ konzentrieren. Der Drückalgorithmus, der in Kapitel 7 vorgestellt wird, wird nämlich auf einer guten Einschätzung, wieviele Punkte mit einer Hand ohne Skat möglich sind, aufbauen. Im nächsten Abschnitt wird dafür eine weitere Eigenschaft der Bewertungsfunktion $\widehat{V}^t(s)$ untersucht: Bislang folgt die Klassifizierung über die Nachbarinstanzen $z_{i,j}$ so, dass kein Unterschied gemacht wird, welcher Menge Z_i diese angehört. Da aber nähere Nachbarn eine Anfrage meist besser beschreiben können wird das Konzept der *Distanzgewichte* eingeführt.

5.3 Distanzgewichte

Eine grundlegende Erweiterung von KNN ist die Einführung von Gewichten, die nähere Nachbarn stärker in die Klassifizierung einfliessen lassen als weiter von der Anfrage entfernte. Dieser Ansatz wird *distanzgewichteter k-nächster-Nachbar-Algorithmus* genannt. Jeder Nachbar z_i, geht dabei mit dem Gewicht

$$w_i = \frac{1}{d(s, z_i)^2} \tag{5.3}$$

in die Klassifizierung ein. Die Bewertungsfunktion ergibt sich damit als

$$\widehat{V}^t(s) = \frac{\sum_{i=1}^{k} w_i V^t(z_i)}{\sum_{i=1}^{k} w_i} \tag{5.4}$$

für einen Trumpf t. Auch hier ist die bereits in Abschnitt 5.1 erforderliche Erweiterung auf Mengen von Nachbarn gleicher Distanz nötig. Hierfür wurde ein Ansatz gewählt, der aus den Elementen $z_{i,j}$ einer Menge Z_i eine repräsentierende Zielfunktion $V^t(Z_i)$ mithilfe des Durchschnitts berechnet, also

$$V^t(Z_i) = \frac{\sum_{j=1}^{k} V^t(z_{i,j})}{|Z_i|} \tag{5.5}$$

Diese wird anstelle von $V^t(z_i)$ in der Bewertungsfunktion 5.4 verwendet. Gleichung 5.3 kann dagegen weitestgehend beibehalten werden, z_i beschreibe lediglich ein beliebiges Element $z_{i,j}$ der entsprechenden Menge. Der Abstand $d(s, z_{i,j})$ ist für alle Elemente einer solchen Menge Z_i identisch.

Zwei kleinere Probleme ergeben sich durch diese Erweiterung: Zum einen gilt bei einer bijektiven Funktion zur Merkmalsextraktion, dass $s = z$ falls ein Nachbar z existiert mit $d(s, z) = 0$. Dementsprechend wird s dann einfach genauso klassifiziert wie z. Im hier betrachteten Problem ist das aber nicht der Fall, gleiche Merkmale können wie bereits gezeigt unterschiedliche Instanzen beschreiben. Eine Analyse der Nachbarmengen aus Abschnitt 5.2 zeigte sogar, dass häufig mehrere Nachbarn selben Abstandes gefunden werden. Der Anfrage s den Durchschnitt der Elemente gleichen Abstands zuzuweisen würde dann Ergebnisse ähnlich zu $KNN(M_2, AV, 1)$ ergeben. Diese waren aber deutlich schlechter als solche mit höherem k. Es bietet sich also an, diese Fälle genauso zu behandeln wie solche, in denen keine Nachbarn mit identischen Merkmalen gefunden werden. Dazu muss Gleichung 5.3 doch durch eine kleine Konstante γ erweitert werden zu

$$w_i = \frac{1}{d(s, x_i)^2 + \gamma} \tag{5.6}$$

Gleichung 5.6 entspricht dem ersten implementierten Distanzgewicht DW_1, für die Konstante wurde dabei ein Wert von $\gamma = 0,0001$ festgelegt.

Das zweite durch die Erweiterung entstandene Problem ist, dass einzelne Nachbarn durch die Größe der Nachbarsmenge, in der sie enthalten sind, zusätzlich gewichtet werden: Elemente aus kleinen Mengen haben einen stärkeren Einfluss auf das Gesamtergebnis als solche aus großen Mengen, da sie die repräsentierende Zielfunktion $V^t(Z_i)$ stärker beeinflussen. Gewichtet man diese zusätzlich mit der Anzahl der in einer Menge enthaltenen Elemente wird dies ausgeglichen. So wurde

$$w_i = \frac{|X_i|}{d(s, x_i) + \gamma} \tag{5.7}$$

als Distanzgewicht DW_3 implementiert.

Der genaue Einfluss von γ auf das Ergebnis kann nur schwer abgeschätzt werden. Es ist möglich, dass alle Nachbarn $z_{1,j}$ mit $d(s, z_{1,j}) = 0$ dadurch zu stark in das Ergebnis einfliessen. Der Algorithmus wird dann ähnliche Ergebnisse liefern wie $KNN(M_2, AV, 1)$. Deshalb soll ein alternativer Ansatz, basierend auf dem Rang der Nachbarschaft $i \in \{1, ..., k\}$, zum Vergleich implementiert werden. Die Gewichte DW_2 (5.8) und DW_4 (5.9) sind definiert durch:

$$w_i = \frac{1}{i^2} \tag{5.8}$$

$$w_i = \frac{|X_i|}{i^2} \tag{5.9}$$

Eine Klassifizierung durch einen distanzgewichteten KNN sei im Folgenden beschrieben als $DKNN(m, c, k, w)$, wobei m, c und k analog der Beschreibung von KNN definiert sind und $w \in \{DW_1; DW_2; DW_3; DW_4\}$ das verwendete Distanzgewicht angibt.

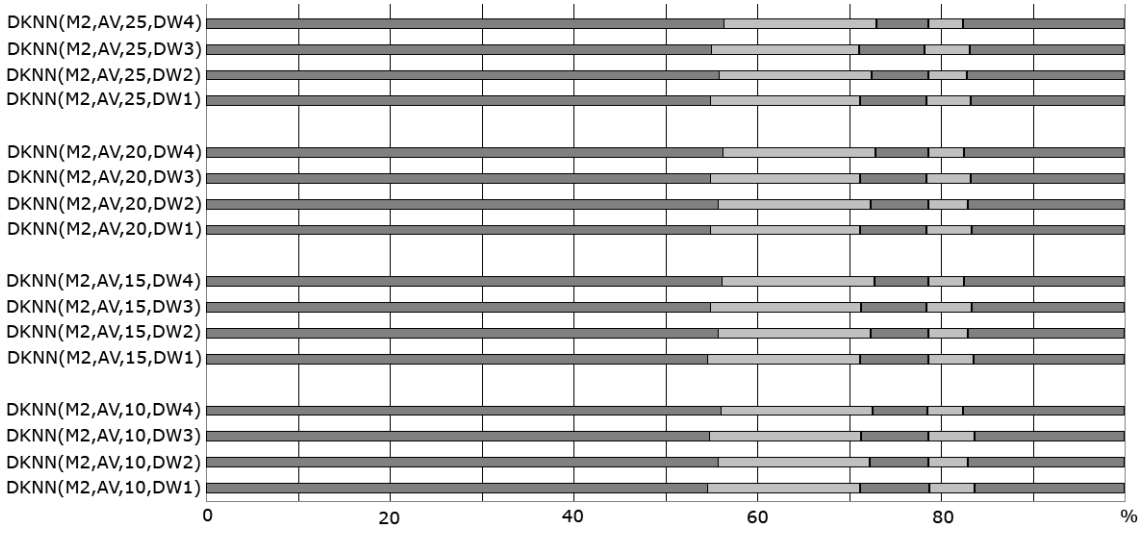

Abbildung 5.7: DKNN-Ergebnisse ($k \in \{10; 15; 20; 25\}$, $w \in \{DW_1; DW_2; DW_3; DW_4\}$)

Im Vergleich der unterschiedlichen Distanzgewichte untereinander schneidet DW_4 in beiden Teilresultaten für richtig klassifizierte Spiele für alle k am besten ab, und damit auch im Gesamtergebnis. Auch der mittlere quadratische Fehler ist unter Verwendung von DW_4 am geringsten. Allerdings sind die Resultate mit DW_2 nur wenig schlechter. Das schlechtere Abschneiden der beiden Fälle, in denen der Abstand das Gewicht definiert, kann wie angedeutet an der notwendigen Einführung von γ liegen. Ist dieser größer als der 'übliche' Abstand zwischen den Nachbarsmengen Z_1 und Z_2 wird erstere zu stark in das Ergebnis einfliessen. Variationen des Wertes zeigten aber auch keine Resultate, die an diejenigen von DW_2 und DW_4 heranreichten.

Eine andere Erklärung ist, dass der Einfluss von Nachbarn mit steigendem Rang der Nachbarschaft in DW_1 und DW_3 schneller abnimmt als dies bei DW_2 und DW_4 der Fall ist, und damit zu wenig Instanzen mit genügend Gewicht zur Klassifizierung verwendet werden. Eine Erhöhung von k, um dem gegenzusteuern, ändert die Ergebnisse nur unwesentlich. Bereits in dem untersuchten Intervall sind kaum Unterschiede auszumachen, da Nachbarn Z_i mit großem i durch ihr vergleichsweise geringes Gewicht kaum mehr Einfluss auf das Resultat haben.

Auch der Vergleich mit den Ergebnissen des vorigen Abschnitts deutet darauf hin — für alle implementierten Distanzgewichte. Mit identischen Parametern ist der mittlere quadratische Fehler für DKNN immer um mehr als 25 Punkte höher - im besten Fall liegt er bei 218,44 für $DKNN(M_2, AV, 25, DW_4)$. Auch das beste berechnete Gesamtergebnis von 72,94% ist denen aus dem vorigen Abschnitt unterlegen. Dies lässt darauf schliessen, dass Nachbarn bis zu einem gewissen Rang der Nachbarschaft ein Skatspiel in etwa gleich gut beschreiben können. Ein Beispiel hierfür ist in Tabelle 5.1 gegeben.

	Hand									**Skat**	
Anfrage s	♣J	♠J	♣10	♣K	♣9	♠A	♠10	♡9	♡8	♢9	♢8	♢Q
Instanz z	♠J	♣A	♣10	♣K	♣9	♠10	♡K	♡Q	♢K	♢7	♣J	♠A

Tabelle 5.1: Anfrage und Instanz mit ähnlichen Gewinnwahrscheinlichkeiten

Für Trumpf ♣ wird s durch den unnormierten Merkmalsvektor $\langle 2; 3; 2; 39; 0; 0; 7; 8\rangle$ beschrieben, die Instanz z durch $\langle 1; 4; 0; 48; 0; 1; 3; 14\rangle$. Der euklidische Abstand beträgt also $d(s, z) \approx 0,79$. Trotz des relativ großen Abstands ist z eine gute Beschreibung von s. Befindet sich z zum Beispiel in der Nachbarschaftsmenge Z_{10}, wird es (vorausgesetzt $k \geq 10$) bei einer KNN-Klassifizierung genauso stark in das Resultat eingehen wie jedes Spiel aus Z_1, während bei DKNN der Einfluss kaum noch wahrzunehmen ist. Unter diesen Voraussetzungen ist ein distanzgewichteter KNN-Algorithmus zur Klassifizierung von Skatspielen also weniger geeignet als der ungewichtete Ansatz.

Verantwortlich für die Diskrepanz zwischen Ähnlichkeit der Merkmale und Ähnlichkeit der Gewinnwahrscheinlichkeiten ist der Skat der Instanz z. Dieser lässt die sicher verlorenen Punkte der blanken ♠10 zu einem relativ sicheren Punktgewinn in dieser Farbe werden. Ausserdem wird ♠J zur höchsten Karte im Spiel und ein bislang bei der Gegenpartei vermuteter hoher Trumpf liegt tatsächlich im Skat. Über Instanzen der Wissensbasis D_1 ist dem Reizer die Verteilung der Karten auch zugänglich, sie werden aber wie Anfragen verwertet obwohl vollständige Information über die Spiele vorliegt. Der folgende Abschnitt soll deswegen untersuchen, in wie weit Änderungen an der Wissensbasis die Leistung des Reizers erhöhen können und ob es möglich ist, die zu erwartenden Verbesserung einer Hand durch den Skat abzuschätzen.

5.4 Entwicklung verschiedener Wissensbasen

Das Problem aus Tabelle 5.1 resultiert aus dem zu hohen Abstand zwischen der Anfrage s und der Instanz z der, solange der Skat unbeachtet bleibt, auch gerechtfertigt ist. Wird dieser in die Betrachtung des Spieles aber mit einbezogen, zeigt sich, dass er einen erheblichen Einfluss auf den Ausgang des Spieles hat, auch wenn er nicht aufgenommen wird. Allein die Tatsache, dass sich die beiden Skatkarten nicht im Spiel befinden, bewirkt eine Stärkung der Hand einer solchen Instanz. Dementsprechend sollte diese auch eher in der Nachbarmenge von Anfragen mit einer vermeintlich stärkeren Hand liegen. In einem ersten Schritt gilt es also, die Wissensbasis so zu ändern, dass verfügbare Informationen über den Skat in die Merkmalsvektoren der gespeicherten Instanzen einfliessen — schliesslich handelt es sich bei den Einträgen der Datenbank um Spiele, bei denen vollständige Information vorliegt. Bislang werden diese aber wie Anfragen als Spiele mit unvollständigem Wissen verarbeitet. Hierfür wird die Datenbank D_2 erstellt, die allen Instanzen den Merkmalsvektor zuweist, der sich durch Berechnung von M_2 mit folgenden Änderungen ergibt:

1. Eine 10 ist auch dann Teil der Asse und 10er, wenn das zugehörige Ass im Skat liegt ($f_3^t(z)$)
2. Eine 10 ist nur dann blank, wenn zusätzlich das zugehörige Ass nicht im Skat liegt ($f_6^t(z)$)
3. Bei der Normierung werden im Skat liegende Karten berücksichtigt. Der Divisor wird dementsprechend kleiner (alle Merkmale)

Für Anfragen s werden bereits alle verfügbaren Informationen bei der Merkmalsextraktion genutzt, deren beschreibende Merkmalsvektoren bleiben also identisch. Die Instanz aus Tabelle 5.1 ist in D_2 mit dem unnormierten Merkmalsvektor $\langle 1, 4, 1, 48, 0, 0, 3, 14\rangle$ beschrieben. Zusätzlich haben auch die Normierungsdivisoren bei $f_1^{\clubsuit}(z)$, $f_4^{\clubsuit}(z)$ und $f_7^{\clubsuit}(z)$ veränderte Werte. Die euklidische Distanz zu s verringert sich dadurch auf $d(s, z) \approx 0,26$ und damit auf weniger als ein Drittel des ursprünglichen Wertes.

Durch diesen ersten Schritt werden Auswirkungen des Skats auf den Spielwert der Karten einer berechneten Instanz so korrigiert, dass für eine Anfrage ein durchschnittlich spielstarker Skat erwartet wird. Aber auch der Punktewert des Skats hat erheblichen Einfluss auf das Resultat einer berechneten Instanz. Da die Punkte des Skats zu denen des Alleinspielers addiert werden, wird das Endresultat einer Instanz mit einem Skat aus einer Kombination von Assen und Zehnern um einige Punkte über demjenigen liegen, das mit der entsprechenden Hand zu erwarten wäre. Rechnet man diese Punkte aus dem Resultat der Spiele in der Datenbank heraus bleibt eine Wissensbasis, in der allen Händen genau die Punkte zugewiesen werden, die durch Stiche erzielt wurden. Kombiniert mit den Änderungen an den Funktionen zur Merkmalsextraktion entsteht so ein genaueres Bild, wieviele Punkte mit einer Hand erzielt werden können. Dann wird allerdings angenommen, dass der Skat gar keine Punkte enthält, was zu einer Unterschätzung führen wird. Da der Erwartungswert der Punkte im Skat für eine Anfrage s einfach über die sich bereits auf der Hand befindenden Karten berechnet werden kann, wird der Algorithmus so verändert, dass dieser Wert zum Resultat der Bewertungsfunktion $\widehat{V}^t(s)$ hinzuaddiert wird. Zusammengenommen entsteht so eine Datenbank, die eine Klassifizierung mit einem durchschnittlich zu erwartenden Skat für eine Anfrage vornimmt. Es sei dabei angemerkt, dass weiterhin die selben Vorraussetzungen wie bei allen vorherigen Implementierungen verwendet werden. Es wird lediglich versucht, mehr Wissen aus der selben Trainingsmenge zu ziehen.

Die Ergebnisse auf D_2 zeigen vor allem ein noch größere Unterschätzung des erreichbaren Resultats. Da Spiele aus der Wissensbasis durch die Änderungen durch stärkere Merkmalsvektoren beschrieben werden, und damit Anfragen durch schwächere Spiele klassifiziert werden ist dies auch zu erwarten. Zwar kann die Unterschätzung durch eine Senkung des Schwellenwertes ausgeglichen werden, da der mittlere quadratische Fehler mit ≈ 257 aber deutlich höher ausfällt wird die Leistung des Reizers durch diese Änderungen nicht erhöht. Die Annahme eines durchschnittlichen Skat scheint also keinen positiven Einfluss bei der Klassifizierung zu haben. Eine Erklärung hierfür ist, dass dieser durch die Änderungen 'manuell' konstruiert wird. Auf D_1 dagegen wird über die große Menge an verwendeten Nachbarn ein tatsächlich zu erwartender Durchschnitt besser gebildet. Es sei hier angemerkt, dass keine der beiden Änderungen in separaten Imlpementierungen zu besseren Ergebnissen führt, ein Ansatz, der lediglich den zweiten Schritt verwendet aber zumindest vergleichbar gute Fehlerraten wie das bisherige Optimum erreicht. Dies liegt daran, dass der erste Schritt durch die Veränderung der Divisoren zur Normierung eine Vergrößerung der Anzahl der möglichen Merkmalsvektoren bewirkt, und damit weniger Nachbarn in jeder Nachbarmenge Z_i liegen. Dies führt dazu, dass bei der selben Anzahl verwendeter Nachbarmengen k weniger Instanzen zur Klassifizierung verwendet werden. Eine — nicht implementierte — Lösung dieses Problems wäre eine geeignete Rundung der extrahierten Merkmale.

Nachdem Veränderungen an der Wissensbasis D_1 bislang nicht die erhoffte Auswirkung auf den Reizer hatten soll untersucht werden, ob eine gesteuert aufgebaute Datenbank der zufällig generierten überlegen ist. Anforderungen an diese sind vor allem

durch die folgenden beiden Eigenschaften beschrieben, die durch zufällige Generierung der Instanzen nicht gegeben sind:

1. Die Instanzen sollten eine gute Repräsentation möglicher Anfragen darstellen
2. Die Instanzen sollten möglichst gleichmäßig über den Merkmalsraum verteilt sein

Der Schritt von D_1 nach D_2 ist letztlich ein Versuch, die erste Eigenschaft zu verbessern. Die Instanz im gewählten Beispiel, bei der ♣J und Trumpfass im Skat liegen, ist keine typische Beschreibung für eine Anfrage s' mit identischer Hand. Auf D_1 wird KNN diese aber als Nachbar mit Distanz 0 zurückgeben, und der Reizer wird damit dazu gebracht, auf einen solchen Skat zu 'hoffen'. Dies stellt aber keine optimale Strategie dar — die Wahrscheinlichkeit, so gute Karten aus dem Skat zu erhalten ist äußerst gering. Auf D_2 wird dieses Spiel daher eher einer stärkeren Hand wie s zugeordnet, die mit einem durchschnittlichen Skat tatsächlich ähnliche Ergebnisse erzielen sollte. Da aber auch auf D_2 keine Leistungssteigerung beobachtet werden konnte, soll dieser Punkt bereits beim Aufbau einer neuen Wissensbasis berücksichtigt werden. Die zweite Eigenschaft dagegen wurde bislang noch gar nicht beachtet. Durch die sehr große Zahl möglicher Merkmalsvektoren entstehen in einer zufälligen Wissensbasis mit großer Wahrscheinlichkeit 'Löcher', bei denen Anfragen nur weit entfernte Nachbarn und damit unzureichende Annäherungen finden. Eine gleichmäßige Verteilung der Instanzen über den Merkmalsraum ist nicht gegeben. Auch dies soll beim Aufbau der Wissensbasis D_3 berücksichtigt werden. Der in Abbildung 5.8 gezeigte Algorithmus wurde zu deren Generierung verwendet:

```
BuildKnowledgeBase()
  for each possible combination of jacks
    add those jacks to cards of player1
    for each i between 1 and 7
      add i trumps (♣) to cards of player 1
      fill up cards of player 1 with random cards
            that are no jack nor a trump
      add one random card to the skat that is no jack
      add one random card to the skat that is no jack nor a trump
      distribute the remaining cards randomly among the opponents
      add the resulting game to the knowledgebase
```

Abbildung 5.8: Aufbau der Wissensbasis D_3

Aufgrund der Symmetrie der Farben zueinander reicht es aus, alle Spiele mit den Trümpfen ♣ und Grand in die Datanbank aufzunehmen. Ausserdem werden weniger Spiele mit 1, 2, 6 oder 7 Trümpfen der Wissensbasis hinzugefügt als Spiele mit 3, 4 oder 5 Trümpfen. Letztere erzielen im Allgemeinen eher Resultate nahe an 60, während

erstere meist einen eindeutigen Ausgang haben. Dadurch wird sichergestellt, dass solche knappen Spiele bei gleichem k sehr viel mehr Nachbarn zur Klassifizierung verwenden als eindeutige, da jede Menge von Nachbarn gleicher Distanz X_i mehr Instanzen enthält. Nicht nur dadurch ist die Verteilung der Instanzen aber nicht ganz gleichmäßig: Lediglich $f_1^t(s)$, $f_2^t(s)$ und $f_7^t(s)$ werden nicht zufällig über den Merkmalsraum verteilt. Da Buben und Trümpfe aber den größten Einfluss auf die Qualität einer Hand haben ist der Aufbau mit dem Algorithmus aus 5.8 den Ansprüchen genügend. Außerdem ist so garantiert, dass mehrere Instanzen mit den exakt gleichen Werten bei diesen Merkmalen in der Wissensbasis vorhanden sind. Eine Datenbank, die gleichmäßig über allen Merkmalen verteilt ist, wird wegen der großen Anzahl möglicher Merkmalsvektoren entweder zu dünn besetzt oder aber zu umfangreich sein.

Um die erste Eigenschaft zu gewährleisten wird lediglich ausgeschlossen, einen Skat mit Buben oder genau zwei Trümpfen zu generieren. Ein menschlicher Spieler würde auf keine der beiden Alternativen beim Reizen hoffen. So ergibt sich die Datenbank D_3 mit 64 000 Einträgen. Die mittleren quadratischen Fehlerkurven der Klassifizierung mit $KNN(M_2, AV, k)$ und $DKNN(M_2, AV, k, DW_4)$ auf D_3 sowie zum Vergleich $KNN(M_2, AV, k)$ auf D_1 sind Abbildung 5.9 zu entnehmen.

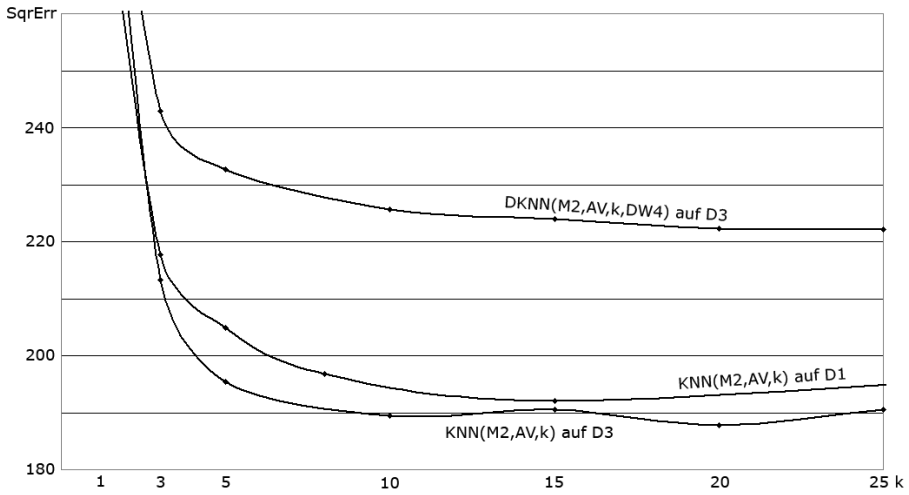

Abbildung 5.9: Mittlerer quadratischer Fehler auf D_3

Abermals zeigt sich die Überlegenheit des ungewichteten KNN gegenüber demjenigen mit Distanzgewichten deutlich, auf eine nähere Analyse von DKNN wird deswegen im Folgenden verzichtet. Der Vergleich mit der Wissensbasis D_1 ergibt für $KNN(M_2, AV, k)$ geringfügig verbesserte Ergebnisse im mittleren quadratischen Fehler, aber ein etwas schlechteres Gesamtresultat: Auf D_3 werden etwas mehr ungewinnbare Spiele korrekt eingeordnet, dafür weniger Gewinnbare. Eine Verschiebung des Schwellenwertes brachte dementsprechend auch ein etwas tieferes Optimum mit $th = 55$. Dennoch deutet auch

dieses Ergebnis an, dass die zufällig generierte Datenbank keine Schwäche des Algorithmus darstellt, die Unterschiede sind dafür zu gering. Die Untersuchung des Einflusses der gewählten Datenbank soll deswegen hiermit auch abgeschlossen werden und ein kurzer Blick auf die zur Klassifizierung einer Anfrage benötigte Zeit durch KNN geworfen werden.

5.5 Analyse der Laufzeit

Zu Beginn dieses Kapitels wird bereits ein bedeutender Nachteil instanzenbasierter Lernalgorithmen angesprochen: Im Unterschied zu vielen anderen Methoden fällt der Großteil der Berechnungen während der Klassifizierung an. Während LMS einen rechenintensiven Lernvorgang benötigt, in dem die verwendeten Gewichte ermittelt werden, ist zur Klassifizierung lediglich die Berechnung der Bewertungsfunktion mit diesen Gewichten nötig, was vernachlässigbar wenig Zeit in Anspruch nimmt. KNN dagegen hat keinen solchen separaten Lernvorgang: Die Klassifizierung der Anfrage wird komplett zur Laufzeit vorgenommen. Schon das Einlesen der Wissensbasis in den Speicher benötigt Zeit, was allerdings nur einmalig zum Start des Programmes geschieht. Auf dem mir zur Verfügung stehenden Rechner wird für das Einlesen von D_1 etwas weniger als eine halbe Sekunde benötigt, für D_3 etwas mehr. Die Zeit, die zur Klassifizierung benötigt wird, beträgt weniger als $0,08$ Sekunden für alle Wissensbasen und den gesamten Wertebereich für die Anzahl der verwendeten Nachbarn k, in dem die besten Ergebnisse erzielt werden. Es zeigt sich also, dass KNN in einer auch für den Einsatz in einem Skatprogramm annehmbaren Zeit in der Lage ist zu klassifizieren.

Damit sollen die Untersuchungen des KNN-Algorithmus abgeschlossen werden. Gegenüber dem LMS-Algorithmus konnte eine starke Verbesserung festgestellt werden, annähernd drei von vier Spielen werden durch einen KNN-Reizer korrekt und schnell klassifiziert. Auch ein mittlerer quadratischer Fehler von unter 190 Punkten ist beachtlich, wenn die Fülle unvollständiger Information in diese Überlegung einbezogen wird. Besser veranschaulicht wird dies noch durch den mittleren Fehler, der im besten Fall bei lediglich 10,93 Punkten liegt. Im nächsten Kapitel soll mit einer Verbindung der beiden bislang verwendeten Algorithmen versucht werden, noch etwas genauere Vorhersagen treffen zu können, bevor ich mich dann in Kapitel 7 dem Drücken zuwenden möchte.

6 KNN und LMS

Am Ende des Kapitels über den LMS-Algorithmus komme ich zu dem Schluß, dass eine einfache Bewertungsfunktion zur Approximation der komplexen Zielfunktion $V^t(s)$ im Skat vermutlich nicht existiert. Dennoch darf keineswegs davon ausgegangen werden, dass die Ergebnisse von LMS auf einer solchen Funktion komplett zufällig sind, der Algorithmus ist durchaus in der Lage, eine Hand zu bewerten. Ein genaues Resultat kann dadurch zwar nicht gut genug abgeschätzt werden, doch ein wichtiges Teilresultat kann aus den Ergebnissen gezogen werden: Der Einfluss eines einzelnen Merkmals auf die Gewinnwahrscheinlichkeit einer Hand. Diese Information kann auch für KNN verwendet werden.

Die Distanzfunktion in KNN wird dafür um Gewichte erweitert, was zu

$$d(s,z) = \sqrt{\sum_{i=1}^{n} w_i \cdot (f_i^t(s) - f_i^t(z))^2} \tag{6.1}$$

für zwei durch einen Merkmalsvektor beschrieben Spiele s und z führt. Anschaulich führt diese Methode zu einer Streckung oder Stauchung der Achsen im Raum der Merkmalsvektoren um den Faktor w_i, wodurch Instanzen, die in den als aussagekräftiger erachteten Merkmalen der Anfrage ähnlicher sind, näher an diese gerückt werden. Zur Bestimmung der Gewichte wurden diese sowohl auf der Trainingsmenge ermittelt, als auch auf der jeweils benutzten Wissensbasis.

Eine detaillierte Angabe der Resultate würde keine zusätzlichen Erkenntnisse bringen, durch die Verwendung von Gewichten in der Distanzfunktion konnten keine besseren erreicht werden. Die Ergebnisse ähneln denen des vorigen Kapitels aber stark. Der Grund dafür ist, dass lediglich Nachbarmengen Z_i untereinander getauscht werden. Durch die sehr unterschiedlichen Normierungsdivisoren entstehen nur selten solche Mengen, in denen Instanzen des selben Abstandes aber unterschiedlicher Merkmale beinhaltet sind. Damit wird der Abstand alle Elemente einer solchen Menge meistens auch weiterhin der selbe bleiben. Von den seltenen Fällen abgesehen werden also nur Mengen, die ohne die Gewichte gerade noch einen Rang der Nachbarschaft von etwa k hatten, durch bislang nicht in die Klassifikation einfliessend Nachbarmengen ersetzt. Da aber sehr viele Spiele zur Klassifikation benutzt werden ist deren Einfluss sehr gering. Bestätigt wird dies dadurch, dass sowohl für kleine k die Ergebnisse stärker abweichen als auch für distanzgewichtete Klassifikationen. Letztere erzielen sogar annähernd genauso gute Ergebnisse wie Implementationen mit ungewichtetem KNN, was darauf schliessen lässt, dass die neue Reihenfolge der Nachbarmengen Anfragen meist besser beschreibt als zuvor. Leider ist der Einfluß auf die Ergebnisse gerade in dem Intervall, in dem KNN die besten Resultate erzielt, unmerklich. Abermals könnte eine Lösung für das Problem eine Rundung der Merkmale derart sein, dass in Mengen gleicher Nachbarschaft Instanzen mit sehr unterschiedlichen Merkmalsvektoren liegen. Dieser Ansatz wurde aber nicht weiterverfolgt.

Stattdessen wurde versucht, eine Möglichkeit zu finden, LMS direkt durch die Bewertungsfunktion von KNN $\widehat{V}^t(s)$ lernen zu lassen. Daraus resultierten aber Probleme, die nicht zufriedenstellend gelöst werden konnten: Auch hier ist eine Minimierung des Fehlers zwischen $\widehat{V}^t(s)$ und der Zielfunktion aus den Trainingsbeispielen $V^t(s)$ notwendig. Es ist aber keine gewichtsabhängige Bewertungsfunktion analog zu Gleichung 4.2 definiert, sondern eine, die aus gespeicherten Instanzen die Bewertung ableitet. Lediglich die Distanz zwischen Anfrage und Instanzen der Wissensbasis ist von den Gewichten abhängig, lediglich diese kann also kleiner oder größer werden. Dafür müsste eine geeignete Menge an Nachbarn z^* aus der Datenbank ermittelt werden, für die es wünschenswert wäre, die Anfrage s zu beschreiben, die dann über Streckung und Stauchung des Raumes möglichst nahe an s gerückt werden. Eine solche Funktion zufriedenstellend aufzustellen ist leider nicht gelungen. Lediglich solche Instanzen z^* dafür auszuwählen, die ein ähnliches Resultat p in der Zielfunktion $V^t(s)$ aufzeigen wird keinen Erfolg bringen, da sehr viele, sehr unterschiedliche Hände ähnliche Ergebnisse p hervorbringen. Ein Ansatz, diese z^* über einen maximal erlaubten Abstand ohne LMS-Gewichte zu ermitteln widerspricht aber dem Sinn dieses Kapitels.

Auch in der Literatur konnte keine Methode gefunden werden, Gewichte für KNN zu lernen. Deswegen soll nun ein Algorithmus für das Drücken implementiert werden. Zwar ist derzeit lediglich eine Vorhersage von durchschnittlich ± 11 Punkten im Vergleich zu den berechneten Resultaten möglich, doch dies ist ausreichend für unsere Zwecke. Eindeutige Spiele werden mit KNN sehr gut als solche erkannt, in diesen Fällen wird der Reizer also korrekt entscheiden, ob gereizt werden soll. Knappe Spiele dagegen werden sich in einem bislang nicht untersuchten Maße durch die Aufnahme des Skats verbessern, was dazu führt, dass eine genauere Einschätzung als der bislang erzielte Fehler gar nicht notwendig sein muss. Desweiteren ist der Reizer sehr gut in der Lage, den stärksten möglichen Trumpf der Hand zu erkennen, wodurch vor allem ein Überreizen durch falsche Trumpfwahl nur sehr selten vorkommt. Ob der momentan verwendete Schwellenwert von $th = 60$ noch gesenkt werden sollte, um bei mehr Spielen auch das höchste Gebot abzugeben wird ebenfalls Gegenstand der Untersuchungen des nächsten Kapitels sein.

7 Drücken

Dieses Kapitel widmet sich nicht nur dem Vorgang des *Drückens* selbst sondern auch der eng damit verbundenen *Trumpfansage*. Zwar wird ein Spieler schon während des Reizens einen gewünschten Trumpf ins Auge gefasst haben, dennoch können diese Pläne aber durch die beiden zur Hand hinzukommenden Karten verändert werden. Letztlich ist für die Trumpfansage die Höhe des letzten Gebots beim Reizen ein entscheidender, limitierender Faktor. Im ersten Abschnitt dieses Kapitels wird von einem Szenario ausgegangen, in dem der Alleinspieler das Reizen mit einem Gebot von 18 gewinnen konnte, ihm stehen also alle Trümpfe zur Verfügung. Der zweite Abschnitt beschäftigt sich nochmals kurz mit dem Reizen, da durch das Drücken zusätzliche, neue Optionen ermöglicht werden. Im dritten Abschnitt sollen Reizen und Drücken dann miteinander verbunden werden.

7.1 Drücken mit KNN

Der Ansatz für das Drücken mit KNN ist einfach: Für alle $\binom{12}{2} = 66$ Möglichkeiten, zwei Karten zu drücken wird für alle möglichen Trümpfe über KNN ermittelt, wieviele Punkte zu erwarten sind. Der höchste Wert bestimmt dann den Trumpf sowie die zu drückenden Karten. Leider ist eine Evaluation des optimalen Drückens auf vielen Spielen damit unmöglich: Für jedes Spiel müssten $5 \cdot 66 = 330$ mögliche Kombinationen berechnet werden. Alle 5 000 Spiele der bislang verwendeten Evaluationsmenge so zu kalkulieren (also $5000 \cdot 330 = 1\,650\,000$ Spiele) würde mit dem DDSS mehrere Wochen benötigen. Aus diesem Grund muss die Evaluierung auf wenigen Beispielen vorgenommen werden, für die alle möglichen Kombinationen an resultierenden Händen durchgerechnet wurden. Drei dieser Spiele sind in Tabelle 7.1 dargestellt. Die letzte Zeile beschreibt dabei die Hand, die die höchste durch den DDSS mit 10 Samples für jeden Spieler berechnete Punktzahl erzielt.

Es wird mit zwei alternativen Ansätzen gedrückt: Der erste (DR_1) verwendet mit $KNN(M_2, AV, 15)$ diejenigen Parameter, die die bislang besten Ergebnisse auf D_1 hervorgebracht hat. Auch die zweite (DR_2) ist ein $KNN(M_2, AV, 15)$, allerdings wird die Wissensbasis ähnlich wie in Kapitel 5.4 modifiziert: Die Punkte des Skat sind aus dem Resultat herausgerechnet. Die dort verwendete Abschätzung der Punkte im Skat über den Erwartungswert erwies sich als zu ungenau, beim Drücken muss dieser aber nicht geschätzt werden. Der Karten, die nach dem Drücken im Skat liegen, sind schliesslich bekannt. Hier sollte dieser Ansatz also sehr viel besere Abschätzungen erzielen.

Spiel 1 (Trumpf: ♡)

	Hand										Skat	
Starthand	♣J	♠J	♣7	♠10	♠Q	♡10	♡Q	♡9	♡8	♢Q	♢J	♢A
DR_1	♣J	♠J	♢J	♠10	♠Q	♡10	♡Q	♡9	♡8	♢A	♣7	♢Q
DR_2	♣J	♠J	♢J	♣7	♡10	♡Q	♡9	♡8	♢A	♢Q	♠10	♠Q
Optimum	♣J	♠J	♢J	♣7	♡10	♡Q	♡9	♡8	♢A	♢Q	♠10	♠Q

Spiel 2 (Trumpf: ♣)

	Hand										Skat	
Starthand	♡J	♢J	♣A	♣9	♠Q	♠8	♡A	♡Q	♢K	♢9	♠K	♢A
DR_1	♡J	♢J	♣A	♣9	♠K	♠Q	♠8	♡A	♢A	♢K	♡Q	♢9
DR_2	♡J	♢J	♣A	♣9	♠K	♠Q	♠8	♡A	♡Q	♢9	♢A	♢K
Optimum	♡J	♢J	♣A	♣9	♠K	♠Q	♠8	♢A	♢K	♢9	♡A	♡Q

Spiel 3 (Trumpf: ♠)

	Hand										Skat	
Starthand	♠J	♢J	♣Q	♣7	♠Q	♠9	♠7	♢10	♢K	♢9	♡Q	♢A
DR_1	♠J	♢J	♣Q	♣7	♠Q	♠9	♠7	♢A	♢10	♢9	♡Q	♢K
DR_2	♠J	♢J	♣Q	♣7	♠Q	♠9	♠7	♢A	♢10	♢9	♡Q	♢K
Optimum	♠J	♢J	♣7	♠Q	♠9	♠7	♡Q	♢A	♢K	♢9	♣Q	♢10

Tabelle 7.1: Drei Beispielsspiele für das Drücken mit KNN

Alle Spiele sind Teil der ursprünglichen Evaluierungsmenge und werden von einem $KNN(M_2, AV, 15)$-Reizer vor der Skataufnahme als gewinnbar mit dem angegebenen Trumpf klassifiziert. In Spiel 1 drückt der zweite Ansatz exakt die beiden Karten, die auch als optimal berechnet werden. Aber auch DR_1 drückt so, dass das Spiel gewonnen wird, und erkennt, dass eine Farbe (in diesem Fall ♣) wegzudrücken von Vorteil ist. Im zweiten Spiel drückt keiner der beiden Ansätze so, wie es als Optimum berechnet wurde. Dennoch wird beidesmal so gedrückt, dass beide Spiele gewonnen werden. Tatsächlich handelt es sich dabei nach den Berechnungen um den dritt- beziehungsweise viertbesten Weg. Insbesondere das Verhalten von DR_1 halte ich aber für vergleichbar gut: Da sich nur zwei ♡ unter den zwölf bekannten Karten befinden ist es unwahrscheinlich, dass die Verteilung der restlichen Karten so ist, dass ♡A keinen Stich erzielt. Dieses nicht zu drücken kann also durchaus die richtige Entscheidung sein. Diese Variante, mit ♢K statt ♢9 gedrückt, wird auch als zweitbester Weg berechnet. Es muss aber noch einmal darauf hingewiesen werden, dass der DDSS selbst mit 100 Samples lediglich eine Approximierung an das Optimum berechnet. Mit nur 10 Samples können die berechneten

Spielzüge schon erheblich abweichen. Diese Problematik zeigt sich auch im dritten Spiel: Die am besten zum Drücken geeigneten Karten sind sicher nicht diejenigen, die als optimal berechnet wurden. Wenn überhaupt eine Karte der Farbe ♢ gedrückt wird, muss dies das ♢A sein, da damit ein sicherer Punkt mehr erzielt wird und die verbleibenden Karten mit der selben Wahrscheinlichkeit Stiche gewinnen. Auch die ♣Q erscheint mir eher zufällig, die blanke ♡Q, die von DR_1 und DR_2 gedrückt wird, ist sicher die bessere Alternative. Lediglich der ♢K sollte aus oben genannten Gründen durch das ♢A ersetzt werden. Dies ist aber schwer für den Drücker zu erkennen, da hohe Beiblattkarten vor allem über $f_3^t(s)$ repräsentiert werden, und Könige nicht in dieses Merkmal einfliessen.

Die manuelle Auswertung der Ergebnisse des Drückens, bei der insgesamt 20 Spiele betrachtet wurden, zeigte durchweg gute Ergebnisse. Besonders erfreulich ist, dass die gute Strategie, eine Farbe komplett wegzudrücken, auch meistens so erkannt wird. In diesem Punkt zeigt DR_1 ein etwas besseres Verhalten, wie es in Spiel 2 zu sehen ist. Können dagegen von mehreren Farben alle Karten gedrückt werden, entscheidet sich DR_2 häufiger für die bessere, wie es in Spiel 1 deutlich wird. Insgesamt sind beide Ansätze aber so leistungsstark, dass keine weiteren Verbesserungen nötig sind.

Dennoch soll noch eine größere Evaluation des Drückers erfolgen: Da nicht alle Möglichkeiten für alle 5 000 Spiele berechnet werden können sollen alle Spiele gedrückt werden. Dann wird überprüft, wie viele Spiele der Evaluierungsmenge mit dem gewählten Trumpf t gewonnen werden. Die Berechnungen des DDSS ergaben, dass 2492 der 5 000 Spiele mit DR_2 gewonnen werden, mit DR_1 2435. Dies zeigt, dass beide Ansätze vergleichbare Ergebnisse auf einer großen Menge erzielen. Die tatsächliche Anzahl gewinnbarer Spiele dürfte zwar etwas höher liegen, aber nicht viel. Ich schätze, dass zwei von drei, maximal drei von vier Spielen mit optimalem Drücken und optimalem Spiel bei vollständiger Information des Alleinspieler und unvollständiger der Gegenpartei auch tatsächlich gewinnbar sind. Hat kein Spieler vollständige Information, wie es sich in einer Skatrunde verhält, halte ich eine Wert von etwa 60% für realistisch, beide Drücker erreichen etwa 50%. Erschwerend kommt hinzu, dass durch nicht-optimales Spiel des DDSS zusätzliche, eigentlich korrekt gedrückte Spiele verloren gehen. Die Leistung des Drückers ist also sehr gut, durch den Informationsgewinn über die Karten im Skat liegt sie noch über der des Reizers. Im Vergleich beider Ansätze zeigt sich, dass DR_2 etwas besser Ergebnisse liefert. Bei einem ähnlichen Vergleich für das Reizen schnitt die unveränderte Datenbank dagegen noch besser ab. Im nächsten Abschnitt soll deswegen untersucht werden, ob durch eine genauere Übertragung dieses Ansatzes auf das Reizen auch dort noch einmal Fortschritte erzielt werden können.

7.2 Reizen auf gedrückten Spielen

Ein in Kapitel 5 behandeltes Problem war die Unsicherheit über die Karten im Skat. Dort wurde ein Ansatz vorgestellt, der einen Erwartungswert für Spiel- und Punktewert des Skats berechnet. Durch die Implementierung eines Drückers bietet sich die Möglichkeit, den Wert des Skats auch für die Spiele der Wissensbasis genauer abzuschätzen. Alle

Spiele aus D_1 wurden dafür gedrückt, und die gedrückten Spiele dann durch den DDSS erneut berechnet und als Wissensbasis D_4 abgespeichert. Da die daraus resultierenden Spiele keine typischen Hände vor der Skataufnahme darstellen, sondern eben eine typische Hand, die bereits durch die Aufnahme des Skat verbessert wurde, wurde die Hand des Alleinspielers und damit auch die Merkmale nicht auf die Werte nach dem Drücken gesetzt. Letztlich beinhaltet D_4 also exakt die selben Spiele wie D_1, lediglich die Zielfunktion $V^t(s)$ wurde geändert. Dadurch ergeben sich Spiele, die eine Hand während des Reizens betrachten und ihr die Punkte zuweisen, die diese im Durchschnitt nach dem Drücken erreichen wird. Die Ergebnisse des Reizens auf D_4 mit $KNN(M_2, AV, 15)$ sind in Abbildung 7.1 abgebildet.

Berechnung / Vorhersage	nicht gewinnbar	gewinnbar	
		richtiger Trumpf	falscher Trumpf
nicht gewinnbar	87,82%	36,82%	
gewinnbar	12,18%	51,07%	12,11%

Abbildung 7.1: Ergebnisse von $KNN(M_2, AV, 15)$ auf D_4

Insgesamt werden durch diese Klassifizierung 73,94% aller Spiele richtig eingeordnet, das Ergebnis ist also vergleichbar mit demjenigen auf D_1, die Ergebnisse zeigen aber eine starke Verschiebung unter den korrekt klassifizierten Spiele hin zu den Gewinnbaren. Zum ersten Mal wird bei einem Schwellenwert von $th = 60$ ein Wert von mehr als 50% erreicht. Durch eine Verschiebung dieses Wertes kann aber auch auf D_1 das Ergebnis des vierten Falls erhöht werden. Beide Klassifizierer sollen deswegen im nächsten Abschnitt auf einen guten Schwellenwert untersucht werden.

7.3 Der Schwellenwert th und Evaluation ganzer Spiele

Der Schwellenwert wird letztlich der Parameter des Reizers sein, mit dem bestimmt werden kann, bei wieviel Prozent der Spiele versucht werden soll, das Reizen zu gewinnen und damit das Spiel als Alleinspieler zu spielen. Eng damit verbunden ist die Risikobereitschaft des resultierenden Spielers: Wird häufiger gereizt werden auch häufiger Spiele gespielt, die nicht gewonnen werden. Wird dagegen zu selten gereizt besteht die Gefahr, dass zu viele mögliche Punktgewinne ausgelassen werden. Für diese Auswertung wurden die 5 000 Spiele der Evaluationsmenge komplett durchgespielt. Das eben beschrieben Szenario wird dafür folgendermaßen abgeändert: Der $KNN(M_2, AV, 15)$-Reizer muss bis zum höchsten notwendigen Gebot reizen, mit diesem darf er dann den Skat aufnehmen. Dementsprechend stehen dem Drücker als Trumpf dann nur die Farben offen, die den selben oder einen höheren Reizkoeffizienten haben wie der vom Reizer ermittelte Trumpf, sowie immer der Grand. Ermittelt also zum Beispiel der Reizer ein Spiel ♠ mit den

ersten beiden Buben wird angenommen, dass bis 33 gereizt wurde. Dementsprechend hat der Drücker die Möglichkeit, ♠, ♣ oder G als Trumpf zu wählen. Überreizen durch mit dem Skat aufgenommene Buben wird dabei nicht berücksichtigt. Die resultierenden Spiele wurden dann vom DDSS mit 10 Samples für alle Spieler berechnet. Abgebildet sind in Abbildung 7.2 die Spiele, auf die korrekterweise gereizt beziehungsweise nicht gereizt wurde.

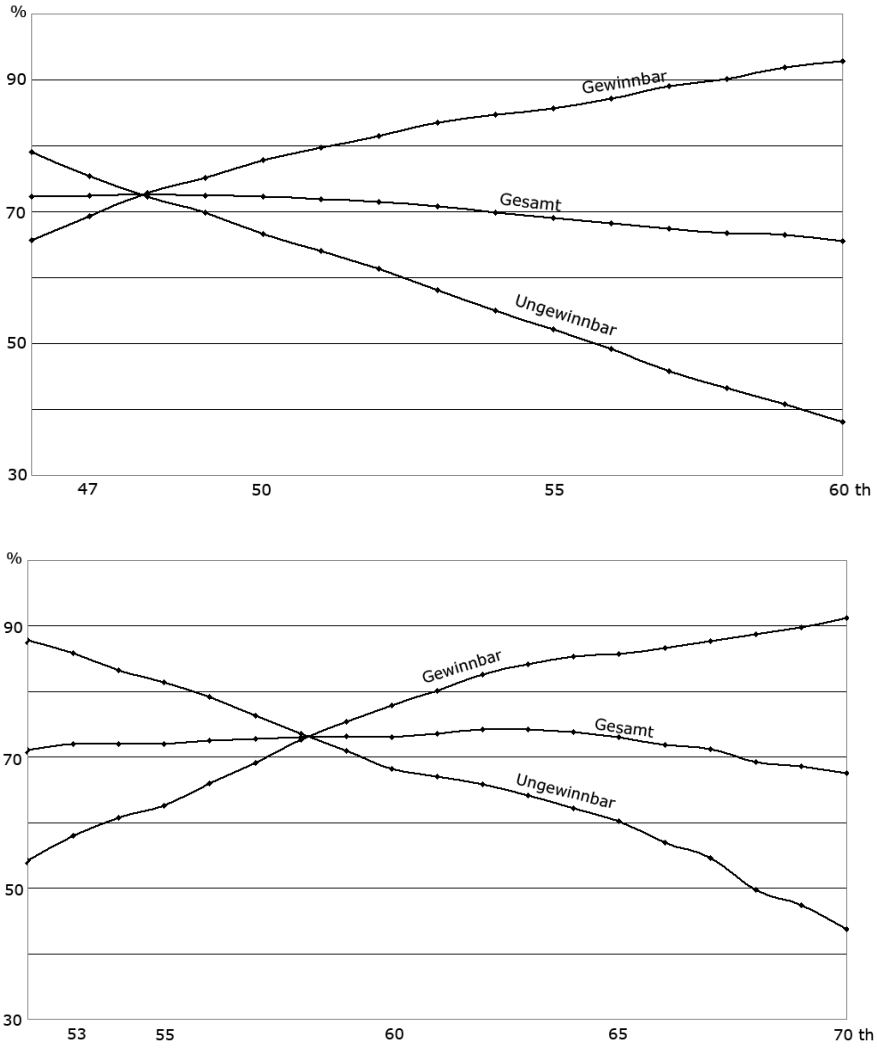

Abbildung 7.2: Anteil korrekt gereizter Spiele mit DR_2 auf D_1 (oben) und D_4 (unten)

Es ist zu erkennen, das beide Drücker im Optimum einen Gesamtwert von etwa 75% erreichen. Allerdings setzt sich dieser Wert aus sehr unterschiedlichen Teilergebnissen zusammen: Das Optimum des Reizens auf D_1 liegt bei $th = 48$, dort werden sowohl 73% der gewinnbaren als auch der ungewinnbaren Spiele korrekt klassifiziert. Das Gesamtoptimum auf D_4 dagegen liegt zwischen $th = 61$ und $th = 64$ mit 74%. In diesem Bereich

werden 80-85% aller gewinnbaren Spiele richtig eingeordnet, dafür lediglich 63-67% der ungewinnbaren. Nicht nur aufgrund des etwas besseren Gesamtergebnisses ist das Reizen auf D_4 aber überlegen. Dadurch, dass der DDSS nur wenige Samples verwendet wird jedes als ungewinnbar klassifizierte Spiel, das durch einen stärkeren Spieler (also zum Beispiel einen DDSS mit 100 Samples) gespielt wird, unter Umständen doch gewonnen werden. Bereits jetzt als gewinnbar klassifizierte Spiele werden aber weiterhin gewonnen. Es ist also zu erwarten, dass einige der in dieser Statistik als ungewinnbar auftauchenden Spiele tatsächlich gewinnbar sind. Davon wird D_4 in einem stärkeren Maße profitieren, und dann auch ein deutlich höheres Gesamtergebnis erzielen.

Ein weiterer Vorteil ist die leichtere Implementierung eines mit unterschiedlichem Risiko reizenden Spieler, da der optimale Bereich sehr viel breiter ist und der Schwellenwert zwischen $th = 55$ und $th = 65$ überall in beiden Teilergebnisse akzeptable Resultate liefert. Denkbar wäre hier zum Beispiel eine Strategie, die weniger risikofreudig reizt, wenn der Spieler im Gesamtergebnis in Führung liegt.

Insgesamt stellt dies ein sehr zufriedenstellendes Ergebnis dar, insbesondere unter dem Gesichtspunkt der Fülle unvollständiger Information beim Skat. Selbst der beste menschliche Spieler wird Spiele annehmen, die dann durch einen schlechten Skat verloren gehen, und in noch größerem Maße wird er auf Spiele nicht reizen, obwohl die Aufnahme des Skats diese gewinnbar gemacht hätte. Der Teil dagegen, in dem die fehlende Information nicht so sehr benötigt wird, nämlich das Drücken, liefert meist nahezu optimale Ergebnisse. Ich denke, ein viel besseres Ergebnis ist für einen maschinellen Reizer und Drücker nicht möglich. Damit soll die Untersuchung des k-Nächsten-Nachbar Algorithmus und dessen Anwendung auf das Skatspiel abgeschlossen werden.

8 Zusammenfassung und Ausblick

Durch die in dieser Studienarbeit implementierten Methoden ist es gelungen, einen leistungsfähigen Reizer und Drücker mit Methoden des maschinellen Lernens zu entwickeln. Gegenüber dem LMS-Algorithmus zeigte sich der KNN-Algorithmus als großer Fortschritt. Insbesondere die Komplexität der beim Skat vorliegenden Zielfunktion machen einen solchen instanzenbasierten Lernalgorithmus zu einer optimalen Wahl für die Lösung beider Probleme. KNN zeigte sich als optimaler Algorithmus zur Implementierung einer Vielzahl von Methoden. So wurden unter anderem verschiedene Merkmale, unterschiedliche Bewertungsfunktionen, Distanzgewichte, Gewichte zur Bewertung der Merkmale oder unterschiedlich aufgebaute Wissensbasen implementiert und analysiert.

Der resultierende Reizer ist in der Lage, drei von vier Spielen trotz unvollständiger Information die korrekte Klassifizierung zuzuweisen. Dieser Wert könnte durch die beschriebenen Defizite des verwendeten DDSS sogar noch höher liegen. Auch der Drücker zeigt hervorragende Ergebnisse. Leider konnten einige Methoden aufgrund der Vielzahl an implementierten Ansätzen nicht für alle Weiterentwicklungen getestet werden. So zeigte zum Beispiel die gesteuert generierte Wissensbasis aus Kapitel 5.4 gute Resultate, konnte aber für das Drücken durch den enormen Zeitaufwand, der für Berechnungen durch den DDSS benötigt wird, nicht implementiert werden.

Sicherlich existieren noch weitere Möglichkeiten für Verbesserungen, die den Rahmen dieser Studienarbeit aber gesprengt hätten. Offensichtlich ist zum Beispiel, analog zum Reizen auch für das Drücken eine Wissensbasis aus Spielen zu verwenden, die bereits gedrückt wurden. Dadurch steht dem Drücker eine Datenbank zur Verfügung, die sehr viel mehr Spiele beinhaltet, die eine reale Situation nach dem Drücken darstellen. Ein weiterer, vielsversprechender Ansatz ist die Implementierung eines Monte-Carlo Ansatzes auf KNN. Für diesen müssen lediglich die Merkmale so erweitert werden, dass auch die Karten der Gegenpartei sowie des Skat darin berücksichtigt sind. Da KNN ein äußerst effizient zu berechnender Algorithmus ist wird auch der Mehraufwand an Rechenzeit akzeptabel bleiben.

Auch kleine Feinheiten, die während einer Skatrunde zur Geltung kommen, wurden angedacht, hier fehlte aber die Umgebung, in der tatsächlich gegen das Programm gespielt wird. Der angesprochene mehr oder weniger risikofreudige Reizer ist ein Beispiel hierfür. Auch für das Lernen während dem Spielen ist KNN sehr geeignet. Ein Ansatz, der tatsächlich gespielte Spiele gegen diejenige Instanz der Wissensbasis austauscht, die vom tatsächlichen Ergebnis am weitesten entfernt ist, wäre denkbar. Eine solche Umgebung, in welcher die in dieser Studienarbeit entwickelten Reizer und Drücker zusammen mit dem DDSS gegen einen menschlichen Gegner spielen, ist deswegen auch angedacht.

Literatur

[Frank und Basin 1998] Frank, Ian ; Basin, David A.: Search in games with incomplete information: A case study using bridge card play. In: *Artificial Intelligence* 100(1-2) (1998), S. 87–123

[Gerhardt 2007] Gerhardt, Gunter: *XSkat*. 2007. – URL http://www.xskat.de/

[Ginsberg 1999] Ginsberg, Matthew L.: *GIB: Steps toward an expert-level bridge-playing program*. 1999. – 584–589 S

[Hsu 2002] Hsu, Feng-Hsiung: *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002

[Kupferschmid 2003] Kupferschmid, Sebastian: *Entwicklung eines Double-Dummy Skat Solvers mit einer Anwendung für verdeckte Skatspiele*, Albert-Ludwigs-Universität Freiburg, Diplomarbeit, 2003. – URL http://www.informatik.uni-freiburg.de/~ki/theses.html

[Mitchell 1997] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill, 1997

[Plaat u. a. 1995] Plaat, Aske ; Schaeffer, Jonathan ; Pijls, Wim ; Bruin, Arie de: Best-first fixed-depth game-tree search in practice. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)* (1995), S. 273–279

[Russell und Norvig 2004] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence — A modern approach*. Prentice Hall, 2004

[Schäfer 2005] Schäfer, Jan: *Monte-Carlo-Simulation im Skat*, Otto-von-Guericke-Universität Magdeburg, Dissertation, Dezember 2005. – URL http://www.b0n541.net/study/jan_schaefer_monte_carlo_simulation_bei_skat.pdf

[Skatordnung 1998] Skatordnung: *Internationale Skatordnung*. 1998. – URL http://www.skat.com/skatordnung.html