TECHNISCHE UNIVERSITÄT DORTMUND
FAKULTÄT FÜR INFORMATIK

Diplomarbeit

Methoden der Computational Intelligence zur
Gegnermodellierung beim Pokerspiel

René Goebels
rene.goebels@tu-dortmund.de
20. Oktober 2008

Gutachter:
Prof. Dr. Günter Rudolph
Dipl.-Inform. Nicola Beume

Lehrstuhl Informatik 11
Algorithm Engineering
der Technischen Universität Dortmund

Erklärung

Hiermit bestätige ich, die vorliegende Diplomarbeit selbstständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst zu haben.

Ich bin damit einverstanden, dass Exemplare dieser Arbeit in den Bibliotheken der TU Dortmund ausgestellt werden.

Dortmund, 20. Oktober 2008

René Goebels

Inhaltsverzeichnis

1 Einleitung

„Poker ist ein Strategiespiel mit einer Glückskomponente“ [Sch07]

So beschreibt die derzeit beste deutsche Pokerspielerin Katja Thater das Pokerspiel. Genauso sehen das auch die meisten ihrer Pokerkollegen weltweit. Ein guter Pokerspieler braucht natürlich Glück, aber in erster Linie ist eine gute Strategie erforderlich, um erfolgreich zu pokern. Eine gute Strategie zu entwickeln ist für einen Pokerspieler jedoch eine schwierige Aufgabe. Er muss in der Lage sein, seine eigenen Karten, die Gegner und die Spielsituation richtig einzuschätzen und zu beurteilen. Daraus muss er Schlüsse ziehen, die seine Strategiewahl und damit seine Spielaktionen beeinflussen. Da dies für einen Menschen bereits ein komplexes Problem darstellt, liegt es nahe, dass sich auch die Informatik mit dem Thema Poker beschäftigt. Computerprogramme zu schreiben, die Spielstrategien entwickeln und selbstständig gegen menschliche Gegner spielen, ist seit jeher dazu genutzt worden, um die Leistungsfähigkeit von Computern zu veranschaulichen in einem Gebiet, das Menschen gemeinhin mit Intelligenz assoziieren. Besondere Popularität hat in diesem Zusammenhang die Untersuchung des Schachspiels erlangt. Mittlerweile sind die Schachprogramme so ausgereift, dass sie wiederholt die besten menschlichen Gegner schlagen. Seitdem hat Schach an Reiz für die Forschung eingebüßt und das Pokerspiel ist immer mehr in den Fokus der Forscher gerückt, welches aufgund der unvollständigen Informationen über die Karten der einzelnen Spieler und die Gemeinschaftskarten sehr viel komplexer als das Schachspiel ist. In Kanada gibt es eine eigene Forschungsgruppe, die sich ausschließlich mit Poker aus wissenschaftlicher Sicht beschäftigt.

Die Modellierung des Gegners ist ein zentraler Bestandteil für ein Pokerprogramm. Ein wichtiger Teil dieses Modells befasst sich damit, die Aktion eines Gegners vorherzusagen, die er in einer Spielsituation ausführen wird. Diese Vorhersage basiert auf den Aktionen, die der Gegner in der Vergangenheit in ähnlichen Situationen ausgeführt hat. Die vorliegenden Arbeit beschäftigt sich mit dieser Aktionsvorhersage, wobei künstliche neuronale Netze und Support-Vector-Machines angewendet werden.

1.1 Zielsetzung

In der vorliegenden Arbeit wird untersucht, ob sich künstliche neuronale Netze und Support-Vector-Machines dazu eignen, Pokerdaten zu klassifizieren, welche die Spielsituationen des Pokerspiels darstellen. Die Aktionsvorhersage wird durch die Klassifikation der Daten in die Klassen Fold, Call und Raise realisiert. Es wird zudem getestet, ob eine verbesserte Klassifikation dadurch erreicht wird, dass zwischen vier Aktionen Fold, Check, Call und Raise unterschieden wird. Durch die Unterscheidung von Check und Call, die jeweils ein „Mitgehen“ bedeuten (vergleiche Abschnitt 2.1.7), ist eventuell eine bessere Abgrenzung zu den Aktionen Fold und Raise möglich. Das Pokerspiel ist rundenbasiert. Daher wird außerdem untersucht,

ob die Klassifikation verbessert werden kann, wenn die Pokerdaten für jede einzelne Spielrunde gesondert klassifiziert werden, anstatt für alle Spielrunden zusammen. Die Ergebnisse der Klassifikation werden untereinander und mit denen eines simplen Klassifikators verglichen, der immer die Aktion vorhersagt, die der Spieler in einer Spielrunde am häufigsten durchführt.

1.2 Aufbau der Arbeit

Kapitel 2 gibt eine Einführung in das Pokerspiel, wobei der Spielablauf, Strategieansätze und die Notwenigkeit der Gegnermodellierung erläutert werden. Das Grundlagenkapitel 3 führt in die verwendeten Techniken und Methoden ein, die für das Verständnis dieser Arbeit nötig sind. Ein allgemeiner Einblick in die Mustererkennung wird gegeben, ebenso wie die Beschreibung der künstlichen neuronalen Netze und der Support-Vektor-Machines, zwei Methoden zur Lösung von Klassifikationsproblemen. Ein Einblick in verwandte Arbeiten, die sich ebenfalls aus wissenschaftlicher Sicht mit Poker beschäftigen, wird in Kapitel 4 gegeben. Zudem erfolgt eine Abgrenzung zu dieser Arbeit. Kapitel 5 beschreibt die Evaluation. Sie befasst sich mit der Beschreibung des Ablaufs und der Durchführung der Tests. Die Ergebnisse der Klassifikation werden dargestellt, Beobachtungen beschrieben und diskutiert. Ebenfalls werden Schlussfolgerungen aus den Ergebnissen gezogen und deren Anwendbarkeit analysiert. Im abschließenden Kapitel 6 wird die Arbeit zusammengefasst und ein Ausblick auf mögliche Erweiterungen gegeben. Jedes Kapitel wird mit einer kurzen Zusammenfassung des Kapitelinhaltes abgeschlossen. Im Anhang befinden sich Tabellen mit Ergebnissen und Plots, auf die in der Arbeit nicht vollständig eingegangen wird. Es sei bemerkt, dass reelle Werte in dieser Arbeit durch einen Punkt als Dezimaltrennzeichen und nicht durch ein Komma getrennt sind. Attributnamen und Funktionen besitzen englische Bezeichnungen bzw. Abkürzungen.

2 Poker

Poker ist der Oberbegriff für eine Kartenspiel-Familie. Besonders in der letzten Zeit hat dieses Spiel sehr an Beliebtheit gewonnen. Diese Tendenz ist an den wachsenden Mitgliederzahlen in den Online-Kasinos im Internet und an der steigenden Zahl der Turniere, die mittlerweile sogar im Fernsehen übertragen werden, zu erkennen. Auch im privaten Umfeld ist dieser Trend zu verfolgen. Das Gewinnen eines Spiels hängt von einem nicht unerheblichen Teil Glück ab, weshalb in den meisten Ländern Poker rechtlich ein Glücksspiel ist. Pokerspiele sind allerdings besonders von Taktik geprägt, da sich die Spieler die Unwissenheit der Gegner über die eigenen Handkarten zu Nutze machen können und ihre Karten als stärker darstellen können, als sie tatsächlich sind. Viele Spieler bezeichnen Poker als Strategiespiel mit Glückskomponente. Wie ist es sonst zu erklären, dass an den Final-Tischen, also den letzten Runden eines Turniers, im Fernsehen meist die gleichen Spieler sitzen.

Dieses Kapitel gibt einen Überblick über die verwendeten Begrifflichkeiten und stellt die wichtigsten Regeln sowie die verschiedenen Pokerarten und -varianten vor. Für die in dieser Arbeit betrachtete Variante Texas-Hold'em wird der Spielablauf im Weiteren näher erläutert. Die Ausführungen orientieren sich an dem Buch von Achenbach [Ach07]. Da die meisten Poker-Begriffe aus dem Englischen stammen, werden Fachbegriffe, wie in diesem Buch verwendet, eingeführt.

2.1 Grundlagen

Es gibt viele verschiedene Arten des Pokerspiels. Allen gemein ist, dass sie mit einem Kartenspiel (**Deck**) gespielt werden, das aus 52 Karten mit vier verschiedenen Spielfarben Pik (♠), Herz (♡), Karo (♢) und Kreuz (♣) besteht. Die Spielkarten weisen pro Spielfarbe 13 verschiedenen Wertigkeiten von 2 bis 9, 10 (**Ten**), Bube (**Jack**), Dame (**Queen**), König (**King**) und Ass (**Ace**) auf. Für Ten bis Ace werden nur die Anfangsbuchstaben genannt, um den Wert der jeweilige Karte zu bezeichnen, beispielsweise steht T♡ für die Karte 10 in Herz oder A♢ für ein Karo-Ass. Die Höhe der Wertigkeit der einzelnen Karten hängt jedoch nicht von der Spielfarbe ab, sondern nur von der Ordnung $2 < 3 < \ldots < 9 < T < J < Q < K < A$. Eine 8♢ hat somit die gleiche Wertigkeit wie eine 8♠.

Das Ziel des Pokerspiels ist es, aus den gegebenen Karten eine möglichst gute **Hand** zu bilden. Eine Hand bezeichnet eine Kombination aus fünf dem Spieler zur Verfügung stehenden Karten. Welche Karten einem Spieler zur Verfügung stehen, wird in Abschnitt 2.1.6 beschrieben. Hände haben verschiedene Wertigkeiten, die in Tabelle 2.1 aufgelistet sind. Dabei gilt, dass der **Royal-Flush** die höchstmögliche Wertigkeit und die Hand, in der nur die höchste Karte zählt, die **High-Card**, die niedrigste Wertigkeit besitzt. Die Ordnung beruht auf der Wahrscheinlichkeit der Hände. Hände, die unwahrscheinlicher zu bilden sind, besitzen eine höhere Wertigkeit als solche, die wahrscheinlicher zu bilden sind.

Tab. 2.1: *Handwertigkeiten*

A♡	K♡	Q♡	J♡	10♡	Royal-Flush	Straight-Flush bis zum Ass
7♠	8♠	9♠	10♠	J♠	Straight-Flush	Straight in gleicher Spielfarbe
J♠	J♡	J♢	J♣	K♠	Vierling	Vier Karten gleicher Wertigkeit
Q♠	Q♡	Q♢	7♣	7♡	Full-House	Ein Drilling und ein Paar
K♣	5♣	J♣	2♣	10♣	Flush	Fünf Karten gleicher Spielfarbe
8♣	9♢	10♣	J♡	Q♡	Straight	Fünf aufeinanderfolgende Karten
10♡	10♢	10♣	5♡	Q♢	Drilling	Drei Karten gleicher Wertigkeit
K♢	K♣	9♠	9♢	J♣	Zwei Paar	Zweimal zwei Karten gleicher Wertigkeit
9♢	9♠	5♣	3♣	K♢	Paar	Zwei Karten gleicher Wertigkeit
Q♠	2♡	7♢	5♣	9♡	High-Card	Höchste Karte

Beim Vergleich zweier Hände spielt die Spielfarbe keine Rolle. Bildet ein Spieler ein Paar mit 3♡ 3♠ und ein anderer Spieler eines mit 3♢ 3♣, dann haben die Paare die identische Wertigkeit. Wenn jedoch zwei Spieler jeweils einen Drilling haben, der eine mit 3♡ 3♣ 3♠ und der andere Spieler mit 4♡ 4♢ 4♠, dann wird der Vierer-Drilling besser bewertet als der Dreier-Drilling, da die Wertigkeit der 4 höher ist als die der 3. Bei einer **Straight**, bei der fünf Karten aufeinander folgen, ist beim Vergleich zweier Hände die höchste Karte ausschlaggebend. Eine Straight 2♡ 3♣ 4♣ 5♣ 6♢ ist weniger Wert als eine Straight mit T♣ J♢ Q♢ K♠ A♣. Eine Straight darf mit einem Ass beginnen und bis zur 5 gehen (A♢ 2♣ 3♢ 4♢ 5♣) oder sie kann mit einem Ass enden (T♣ J♣ Q♡ K♢ A♡). Jedoch ist eine Straight über das Ass hinaus nicht erlaubt, beispielsweise Q♣ K♡ A♣ 2♢ 3♡. Bei einem **Flush**, also fünf Karten in der gleichen Spielfarbe, entscheidet ebenfalls die höchste Karte. Der Flush mit 2♡ 4♡ 5♡ 9♡ K♡ verliert gegen einen Flush mit 2♣ 4♣ 5♣ 9♣ A♣, jedoch nicht wegen der Farbe, sondern aufgrund der höheren Wertigkeit des Ass gegenüber des Königs. Wenn die höchste Karte den gleichen Wert hat, dann entscheidet die zweithöchste usw. Wenn zwei Spieler die gleiche Wertigkeit einer Hand haben, entscheidet der so genannte **Kicker**, die Beikarte. Dieser ist die nächste, höchste Einzel-Karte, die nicht Bestandteil beispielsweise eines Paares ist. Ein Spieler, der das Paar 3♡ 3♠ und den Kicker A♣ hält, schlägt damit die Hand des Gegners mit 3♢ 3♣ und Kicker K♠. Auch hier gilt, dass bei gleichwertigem Kicker die nächsthöchste Karte in Betracht gezogen wird usw. Für den Fall, dass zwei Spieler exakt die gleiche Wertigkeit einer Hand besitzen, ist dieses Spiel ein Unentschieden, und die Spieler teilen sich den **Pot** (vergleiche Abschnitt 2.1.1). Daher wird dieses Unentschieden auch **Split-Pot** genannt.

2.1.1 Einsatz

Die Spieler erhalten zu Beginn eines Pokerspiels eine festgelegte Menge an Spiel-Chips. Diese Chips können von den Spielern eingesetzt werden, wenn sie an der Reihe sind. Wenn die Spieler einen Einsatz bringen, dann zahlen sie diese Chips in den **Pot** ein. In diesem werden die Chips gesammelt und am Ende eines Spiels an den Spieler ausgeschüttet, der noch im Spiel ist und die beste Hand hält. Auf das Setzverhalten wird im Zuge der Erläuterung des Ablaufs der Spielvariante **Hold'em-Poker** in Abschnitt 2.1.7 näher eingegangen.

2.1.2 Spielarten

Es gibt drei verschiede Grundarten der Pokerspiele: **Hold'em-Poker**, **Draw-Poker** und **Stud-Poker**. Diese unterscheiden sich in erster Linie durch die Art und Weise, auf welche die Spieler ihre Karten bekommen. Beim Hold'em-Poker bekommt jeder Spieler verdeckte Handkarten, die so genannten **Hole-Cards**, die nur der Spieler selbst kennt. Zudem werden während des Spiels öffentliche Gemeinschaftskarten, die **Community-Cards** aufgedeckt. Diese Community-Cards können von jedem Spieler benutzt werden, um eine Hand zu bilden. Die Gemeinschaftskarten existieren beim Draw-Poker nicht. Hier bekommt ein Spieler nur verdeckte Karten auf die Hand. Beim Stud-Poker wiederum bekommt ein Spieler zusätzlich zu seinen verdeckten Hole-Cards noch private Karten, die offen direkt vor ihm aufgedeckt werden, jedoch nur dem jeweiligen Spieler zur Bildung einer Hand zur Verfügung stehen.

2.1.3 Texas Hold'em

Texas Hold'em ist die zur Zeit meistverbreitete Pokervariante. Sie wird typischerweise mit zwei bis zwölf Spielern gespielt. Texas Hold'em ist auch die Grundlage dieser Arbeit und wird nun genauer erläutert. In Abbildung 2.1 ist die Anordnung der Spieler am Tisch zu sehen.

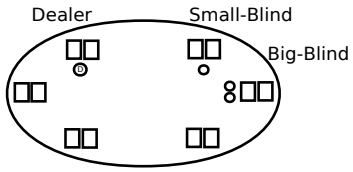

Abb. 2.1: *Positionen am Pokertisch*

2.1.4 Dealer

Zunächst wird einer der Spieler als **Dealer** bestimmt. Dieser ist in dem aktuellen Spiel der Kartengeber, der die Karten mischt und verteilt. Der so geannte **Dealer-Button** ist ein Chip, der vor den Dealer gelegt wird und ihn dadurch als solchen kennzeichnet. Nach einem Spiel wird der links neben dem aktuellen Dealer sitzenden Spieler neuer Dealer und der Dealer-Button dorthin weitergegeben. Der erster Dealer wird vor Beginn des ersten Pokerspiels ermittelt, indem jeder Spieler eine verdeckte Karte zieht und derjenige Spieler ausgewählt wird, dessen Karte die höchste Wertigkeit hat. Nur bei dieser Festsetzung ist die Spielfarbe wichtig. Bei gleicher Wertigkeit wird der Spieler erster Dealer, dessen Spielfarbe der Karte in der Ordnung $\spadesuit > \heartsuit > \diamondsuit > \clubsuit$ am höchsten ist.

2.1.5 Blinds

Blinds sind Zwangseinsätze, die gewisse Spieler in den Pot zahlen müssen, bevor die Karten ausgeteilt werden. Damit soll erreicht werden, dass zu Beginn eines Spiels der Pot nicht leer ist und es somit einen Anreiz für die Spieler gibt, in diesen zu investieren, um die ersten Gemeinschaftskarten zu sehen. Der Begriff Blind ist darauf zurückzuführen, dass die Spieler den Einsatz bringen müssen, ohne ihre Hole-Cards gesehen zu haben. Der Spieler links neben dem Dealer muss den so genannten **Small-Blind** setzen, der einen bestimmten Wert hat. Der Spieler links neben dem Small-Blind muss den doppelt so hohen Einsatz des **Big-Blind** setzen.

Bei Pokerspielen in Turnierform steigen die Blinds jeweils nach einem vorher festgelegten Zeitintervall an. Bei der in dieser Arbeit behandelten Variante sind die Blinds jedoch zu jedem Zeitpunkt identisch. Dies ist das so genannte **Cash-Game**.

2.1.6 Spielablauf

Anfangs werden die Blinds gesetzt und daraufhin jedem Spieler vom Dealer zwei verdeckte Karten aus dem Deck zugeteilt. Der Spielablauf teilt sich in vier Spielrunden auf. In der ersten Spielrunde gibt es noch keine Gemeinschaftskarten. Diese Spielrunde wird **Preflop** genannt. Nun werden die Einsätze der Spieler gebracht, wie in Abschnitt 2.1.7 beschrieben wird. Die nächste Spielrunde ist der so genannte **Flop**. Dafür legt der Dealer drei Karten des verbliebenen Decks auf das **Board**, den Spieltisch. Diese Karten bilden die Gemeinschaftskarten, die jeder Spieler zur Bildung einer Hand nutzen kann. Die nächste Spielrunde ist der **Turn**, wozu eine weitere Karte aufgedeckt wird. Die letzte Spielrunde ist der **River**. Dazu wird eine fünfte und damit letzte Gemeinschaftskarte aufgedeckt. Nachdem in dieser Spielrunde die Einsätze gebracht worden sind, kommt es zum so genannten **Show-Down**. Dabei werden die Handwertigkeiten der sich noch im Spiel befindlichen Spieler verglichen und der Inhalt des Pots an den jeweiligen Gewinner des Spiels ausgezahlt. Die Spielrunden sind in Abbildung 2.2 dargestellt. Zu Beginn des folgenden Spiels wird der Dealer-Button eine Position im Uhrzeigersinn weitergeschoben, genauso wie die Position der Blinds. Demnach wechselt der Zwangseinsatz nach jedem Spiel im Kreis, so dass jeder Spieler zum Dealer wird und die Zwangseinsätze bringen muss.

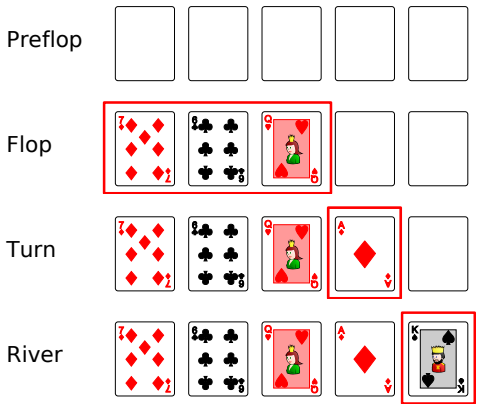

Abb. 2.2: *Die verschiedenen Spielrunden*

2.1.7 Spieleraktionen

Jeder Spieler, der sich noch im Spiel befindet, hat folgende Möglichkeiten zu agieren, wenn er an der Reihe ist:

- Der Spieler kann aus dem Spiel aussteigen, indem er seine Karten ablegt und dem Dealer verdeckt zurückgibt. Diese Karten kommen in dieser Spielrunde nicht mehr zum Einsatz. Diese Aktion wird **Fold** genannt. Damit verliert er alle eventuell bis zu dem Zeitpunkt schon in den Pot eingesetzte Chips. Er kann jedoch in diesem Spiel keine weiteren Chips verlieren.

- Wenn zuvor in der Spielrunde keine Erhöhung gesetzt worden ist und demnach der aktuelle Spieler bereits genau so viele Chips in den Pot einbezahlt hat, wie alle anderen noch aktiven, also nicht aus der Spielrunde ausgestiegenen Spieler, dann kann dieser Spieler *schieben* und der nächste Spieler ist an der Reihe. Demnach verbleibt der Spieler im Spiel, ohne weitere Chips in den Pot einbezahlen zu müssen. Dies wird als **Check** bezeichnet.

- Falls aber ein vorheriger Spieler eine Erhöhung getätigt hat und der aktuelle Spieler im Spiel verbleiben möchte, muss er die Differenz zwischen seinem bisher getätigten Einsatz und dem Gesamteinsatz des erhöhenden Spielers nachbezahlen. **Call** ist die Bezeichnung für diese Aktion. Check und Call stellen ein so gennantes „Mitgehen“ dar, da ein Spieler ohne eigene Erhöhung im Spiel verbleibt.

- Der aktuelle Spieler kann die zuvor von einem Spieler getätigte Erhöhung noch weiter erhöhen. Dazu zahlt er die Differenz zwischen seinem bisherigen Einsatz und dem Einsatz des erhöhenden Spielers und zusätzlich einen Betrag, der mindestens so hoch sein muss, wie die vorangegangene Erhöhung. Diese Aktion wird **Raise** genannt. Wenn zuvor noch nicht erhöht worden ist, der aktuelle Spieler also die erste Erhöhung in dieser Spielrunde macht, ist die Bezeichnung für die Erhöhung **Bet**. Falls ein Spieler alle seine ihm noch verbleibenden Chips einsetzt, wird dies **All-In** genannt. Der erlaubte Betrag der Erhöhung kann sich je nach Poker-Variante unterscheiden. Es gibt so genanntes **Limit**-Texas-Hold'em, bei dem eine Erhöhung einen vorher festgesetzten Grenz-Betrag nicht überschreiten darf. Hierbei ist also ein All-In nur in dem Fall möglich, in dem der Restbestand der Chips eines Spielers dieses Limit unterschreitet. In allen anderen Fällen ist eine Erhöhung maximal bis zu dieser Grenze möglich. Bei der in dieser Arbeit betrachteten Variante ist keine solche Grenze festgelet. Sie wird mit **No-Limit**-Texas-Holdem bezeichnet. Im Weiteren werden Raise, Bet und All-In unter dem Begriff Raise zusammengefasst.

Durch ein Raise eines Spielers werden alle nachfolgenden aktiven Spieler unter Aktionszwang gesetzt. Das gilt auch für die Spieler, die bereits in dieser Spielrunde eine Aktion ausgeführt haben. Sie müssen sich entscheiden, ob sie **folden**, die Erhöhung **callen** oder aber eine weitere Erhöhung ansetzen, also **raisen** wollen. **Checken** ist nun keine Alternative mehr, da noch weitere Chips nachbezahlt werden müssen. Wenn nun alle aktiven Spieler in einer Spielrunde den gleichen Betrag an Chips in den Pot bezahlt haben, ist diese Spielrunde beendet und die nächsten Karten werden aufgedeckt, damit die folgende Spielrunde beginnen kann.

Nachdem die River-Spielrunde beendet ist, zeigt zunächst der Spieler, der als erstes links neben dem Dealer sitzt, seine Karten, bzw. der Spieler, der zuletzt eine Erhöhung gemacht hat. Im Uhrzeigersinn werden dann die weiteren Hole-Cards der Spieler aufgedeckt, wenn die Spieler über eine bessere beliebige Fünf-Karten-Kombination aus den insgesamt sieben Karten (fünf Community- plus zwei Hole-Cards) verfügen. Für den Fall, dass die Kombination eine schlechtere Wertigkeit besitzt als eine bereits aufgedeckte Hand, müssen die Spieler ihre Karten nicht zeigen. Somit kann der Spieler seinen Gegnern die Information weiter vorenthalten, mit welchen Karten er gespielt hat. Der Spieler, der die höchstwertige Hand zeigt, bekommt alle Chips aus dem Pot. Bei genau gleichwertigen Händen wird der Pot geteilt und es kommt zum bereits erwähnten Split-Pot. Ein Spiel kann auch bereits beendet sein, bevor die letzte Spielrunde durchgeführt worden ist. Dieser Fall tritt dann ein, wenn alle Spieler außer ei-

nem gefoldet haben. Damit erhält der letzte aktive Spieler alle Chips aus dem Pot. Wenn ein Spieler alle seine Chips verloren hat, scheidet er aus und kann nicht mehr am Spielgeschehen teilnehmen. In Cash-Games hat der Spieler die Möglichkeit, sich im nächsten Spiel wieder neu einzukaufen und weiter ins Spielgeschehen einzugreifen.

2.2 Bietstrategie

In welcher Situation soll der Spieler nun welche Aktion ausführen? Und wie hoch soll der Einsatz sein? Genau das sind die schwierigen Fragen beim Pokerspiel, weil ein Spieler nur seine Hole-Cards und die Gemeinschaftskarten kennt. Die aus den Karten zu bildende Hand kann dahingehend bewertet werden, ob sie schon eine hohe Wertigkeit besitzt und wie gut die Chancen dafür stehen, dass sich die Hand durch Karten verbessert, die auf dem Board noch aufgedeckt werden. Dementsprechend kann eine Aktion ausgewählt und ein Einsatz erbracht werden. Bei einer Hand mit einer bereits hohen Wertigkeit wird beispielsweise stark erhöht, bei einer Hand mit hohem Potential dazu, dass sie sich verbessert, wird ein wenig erhöht und bei einer schlechten Hand mit wenig Aussichten auf eine Verbesserung wird gefoldet.

Die aktuelle Wertigkeit einer Hand wird mit **Hand-Value** bezeichnet und die Möglichkeit auf Verbesserung nennt sich **Hand-Potential**. Eine Hand mit beispielsweise A♡ 3♡ 4♡ 5♣ 7♡ hat eine sehr geringe Wertigkeit, da im Moment nur das Herz Ass als High-Card die Stärke der Hand repräsentiert. Jedoch hält sie ein hohes Potential, dass sie sich verbessert. Es fehlt nur noch eine 2 oder 6 zu einer Straight oder eine beliebige Karte der Spielfarbe Herz, um einen Flush zu bilden.

Es gibt auch die Möglichkeit des **Bluffs** und des **Slow-Plays**. Beim Slow-Play hält der Spieler bereits eine Hand mit hoher Wertigkeit, spielt aber so, als wären seine Karten nicht so stark. Er erhöht also nicht, sondern callt nur die Erhöhungen der Gegner, übt also selbst keinen Druck aus. Damit vermittelt er bei seinen Gegnern den Eindruck, dass ihre Hände stärker seien und verleitet sie zu Aktionen, welche die Höhe des Pots immer weiter ansteigen lässt. Am Ende werden die Gegner überrascht, dass der Spieler doch eine sehr gute Hand hält und den Pot gewinnt. Diese Spielweise birgt jedoch die Gefahr, dass sich die Hände der Gegner ebenfalls noch so verbessern, dass sie die eigene, bisher als stark angesehene Hand, schlagen.

Der Bluff stellt den umgekehrten Fall da. Hierbei repräsentiert der Spieler eine sehr starke Hand, obwohl er eine Hand hält, die eine geringe Wertigkeit besitzt und wenig Potential auf eine Verbesserung hat. Er erhöht und erzeugt so Druck auf die Gegner, die entscheiden müssen, ob es sich für sie lohnt, noch weiter im Spiel zu verbleiben oder doch besser auszusteigen und damit nicht noch mehr Chips zu investieren und möglicherweise zu verlieren. Damit kann also auch mit einer schlechten Hand ein Spiel gewonnen werden. Hier besteht die Gefahr darin, dass ein Gegner jedoch eine sehr starke Hand hält und gar nicht dazu verleitet wird zu folden. Dann verliert der bluffende Spieler einen großen Anteil seiner Chips. Weitere Ausführungen zu Bluffs und Slow-Play sind in Sklanskys Buch [Skl04] nachzulesen.

Das Bietverhalten vor dem Flop und nach dem Flop unterscheidet sich nach Billings et al. [BDS02] sehr. Beim **Preflop** sind sehr wenige Information für die Spieler vorhanden. Sie kennen nur ihre eigenen Hole-Cards, die ihre Starthand ausmachen. Für diese Starthände gibt

es Bewertungen, wie stark sie im Vergleich zu anderen Starthänden sind. Diese Bewertung wird **Income-Rate** genannt. Für genauere Ausführungen zu den Preflop-Strategien sei auf das Buch von Sklansky [Skl99] verwiesen. Beim **Postflop**-Spiel, also in den Spielrunden Flop, Turn und River, sind mehr Informationen vorhanden, da schon Gemeinschaftskarten offen liegen und die Aktionen der Gegner bis zu dem Zeitpunkt in die Entscheidungsfindung einfließen können.

2.3 Gegnermodellierung

Beim Pokerspiel ist es auch wichtig, nicht nur die eigenen Hole- und die Community-Cards zu beachten, sondern im Besonderen auch das Verhalten der Gegner. Da ein Spieler die Karten der Gegner nicht kennt, muss er Vermutungen über diese anstellen. Solche Vermutungen beruhen darauf, wie das Setzverhalten eines Gegners in der Vergangenheit gewesen ist und wie es sich verändert, wenn eine bestimmte weitere Karte aufgedeckt wird.

Spieler lassen sich grob in vier Kategorien einteilen. Zunächst gibt es Spieler, die sich sehr oft den Flop anschauen. Sie callen also im Preflop auch mit einer schlechten Starthand jede Erhöhung, um sich den Flop anzuschauen und dann zu bewerten zu können, wie hoch die Handwertigkeit und das -potential sind. Dafür muss er jedoch immer wieder Chips investieren. Dieses Verhalten wird **loose** genannt. Im Gegensatz dazu gibt es das Spielverhalten, welches als **tight** bezeichnet wird und bei dem Spieler eher zurückhaltend spielen. Dabei werden nur die Starthände gespielt, die eine hohe Income-Rate versprechen. Alle anderen Hände werden gefoldet. Eine weitere Unterscheidung erfolgt nach der Art und Weise, wie erhöht wird. **Passive** Spieler erhöhen nur selten. Sie callen zwar die Erhöhungen der Gegner, bauen jedoch selbst keine Druck auf, indem sie Erhöhungen ansetzen. Im Gegensatz dazu raisen **aggressive** Spieler sehr häufig, auch wenn sie eine nicht so starke Hand besitzen. Diese Eigenschaften kombiniert ergeben die Spielertypen **loose-passive**, **loose-aggressive**, **tight-passive** und **tight-aggressive**.

Es kommt zusätzlich darauf an, wie ein Spieler in bestimmten Spielsituation reagiert, also welche Aktion Fold, Call, Check oder Raise er ausführen wird. Dafür kann entscheidend sein, wie oft vor dem Spieler schon geraiset worden ist, ob er in der vorherigen Spielrunde bereits erhöht hat, wie viele aktive Spieler sich noch im Spiel befinden und an welcher Position der Spieler sitzt. Eine genauere Beschreibung der Spielsituation ist in Unterkapitel 5.2 zu finden. Bei der Gegnermodellierung, **Opponent-Modeling**, soll also ein Modell zu einem Spieler erstellt werden, in dem aus dem Spielkontext eine Situation erkannt wird und daraufhin Rückschlüsse auf die Aktion des Spielers zu machen sind, also ob er folden, checken, callen oder raisen wird. Dieses Modell wird dann in die Entscheidungsfindung der eigenen Strategie mit einbezogen. Wenn ein Spieler weiß, welche Aktion ein Gegner wahrscheinlich ausführen wird, kann dies die Wahl der eigene Aktion beeinflussen. Wenn beispielsweise der Gegner mit großer Sicherheit callen oder sogar erhöhen wird, dann ist es nun ein falscher Zeitpunkt, mit einer schlechten Hand einen Bluff zu versuchen. In diesem Fall ist Fold die bessere Wahl. Eine anderes Beispiel ist das so genannte **Check-Raise**, welches stark mit dem Slow-Play zusammenhängt. Wenn ein Spieler eine starke Hand hält und ein Gegner bei einer Erhöhung sehr wahrscheinlich folden wird, aber bei einem Call des Spielers wahrscheinlich selbst eine Erhöhung ansetzt, dann sollte in diesem Fall eher gecheckt werden. Die Erhöhung des Gegners

wird dann, wenn der Spieler wieder an der Reihe ist, abermals erhöht, damit die Größe des Pots ansteigt, um die Auszahlung für den Spieler im Endeffekt zu erhöhen.

Das Modell muss adaptiv sein, damit es eventuelle Strategieänderungen des Gegners in zukünftigen Entscheidungen berücksichtigen kann. Auf verschiedene Modelle von Gegnermodellierungen wird in Kapitel 4 näher eingegangen.

Auch das Verhalten des Spielers an sich, wie schnell er eine Aktion ausführt, ob er zittert beim Erhöhen, ob er während der einen Spielsituation viel und in der anderen wenig mit seinen Gegnern kommuniziert, ob er angespannt oder locker wirkt usw., kann in die Vermutungen einfließen, welche Hand ein Gegner im Moment hält und die kommende Aktion beeinflussen. Diese physischen Zeichen werden **Tells** genannt. Ausführungen dazu sind in dem Buch von Caro [Car03] zu finden, worauf in der vorliegenden Arbeit jedoch nicht weiter eingegangen wird.

Durch Beobachtung des Gegners wird versucht, dessen Strategie herauszufinden, um darauf angemessen zu reagieren, also zu bestimmten Zeitpunkten zu erhöhen oder doch eher die Karten zu verwerfen. Daran wird das eigene Setzverhalten, also die Bietstrategie angepasst. Desweiteren wird bei der Gegnermodellierung versucht, durch die bisherigen Aktionen in der aktuellen Spielrunde eine Wahrscheinlichkeitsverteilung für die Hole-Cards der Gegner zu erstellen, welche wiederum in die Entscheidungsfindung einfließt. Dies ist jedoch nicht Bestandteil dieser Arbeit und wird daher nicht weiter beschrieben. Es sei auf die Ausführungen von Billings et al. [BDS02] verwiesen.

Es ist das Ziel eines jeden Pokerspielers, möglichst unberechenbar zu sein, also seiner Bietstrategie keine Regelmäßigkeit zu unterlegen, die erkannt und ausgenutzt werden kann. Diese Regelmäßigkeiten werden bei schwachen Spielern und Anfängern eher vorkommen als bei starken Spielern, die ihre Strategie oft wechseln. Eventuell sind solche Gesetzmäßigkeiten aber auch unbewusst vorhanden und durch einen Spieler nicht steuerbar.

2.4 Spieltheoretische Einordnung

Das Pokerspiel gehört zu den Spielen mit **unvollständiger Information**. Die Informationen, die ein Spieler besitzt, sind insofern unvollständig, dass er nur die eigenen Karten und die bereits aufgedeckten Community-Cards kennt. Über die Karten, welche die Gegner halten, kann ein Spieler nur Vermutungen anstellen. Auch die noch aufzudeckenden Karten beinhalten Informationen, welche die Spieler nicht kennen. Im Gegensatz dazu gehört z. B. Schach zu den Spielen mit **vollständiger Information**, da hier zu jedem Zeitpunkt allen Spielern die gleichen Informationen vorliegen und keine Zufallskomponente Einfluss hat. Die unvollständige Information ist ein Grund dafür, warum das Pokerspiel mittlerweile Gegenstand vieler wissenschaftlichen Arbeiten ist und es in Kanada eine eigene Arbeitsgruppe gibt, die sich mit dem Thema Poker aus wissenschaftlicher Sicht beschäftigt, vergleiche Kapitel 4.

2.5 Zusammenfassung

In diesem Kapitel sind die Grundlagen des Pokers vorgestellt worden. Die Regeln und der Ablauf der Poker-Variante Texas-Hold'em sind dargestellt worden. Außerdem sind die Bietstrategien angesprochen worden, die durch die Gegnermodellierung unterstützt werden.

3 Grundlagen

Beim Pokerspiel besteht ein Klassifikationsproblem darin, Spielsituationen so voneinander zu trennen, dass sie auf die Aktion (Fold, Check/Call, Raise) schließen lassen, die ein Spieler ausführen wird. Dazu werden in diesem Kapitel die Grundlagen erklärt, die für das Verständnis von Klassifikation und Mustererkennung notwendig sind. Zudem werden die Klassifikationsverfahren künstliche neuronale Netze und Support-Vector-Machines vorgestellt. Für genauere Ausführungen an einigen Stellen sei für die künstlichen neuronalen Netze auf das Buch von Nauck et al. [NKK94] und für die Support-Vector-Machines auf das Buch von Scholkopf und Smola [SS01] verwiesen.

3.1 Mustererkennung und Klassifikation

Als **Klassifikation** wird der Vorgang bezeichnet, ein Objekt einer Klasse zuzuweisen. Die Zuweisung zu einer Klasse soll darauf basieren, dass Objekte innerhalb einer Klasse Ähnlichkeiten aufweisen. Nauck et al. [NKK94] definieren diese Objekte als Muster folgendermaßen.

Definition 3.1.0.1 *Gegeben seien n Merkmale (Attribute) $\xi_1, \ldots, \xi_n$ mit ihren korrespondierenden Merkmalsmengen (Attributmengen) $X_1, \ldots, X_n$. Das kartesische Produkt $X = X_1 \times \ldots \times X_n$ heißt Merkmalsraum (Universum, Diskurswelt). Ein Element $\boldsymbol{x} \in X$ wird Muster genannt und durch den Merkmalsvektor $(x_1, \ldots, x_n)$ repräsentiert. Die Werte $x_i \in X_i$ $(i \in \{1, \ldots, n\})$ stellen die (aktuellen) Ausprägungen der korrespondierenden Merkmale ξ_i $(i \in \{1, \ldots, n\})$ dar.*

Um Ähnlichkeiten zwischen Objekten bestimmen zu können, wird eine Ähnlichkeitsfunktion $k : X \times X \to \mathbb{R}$ mit $\mathbf{x} \in X$ benötigt. Das Skalarprodukt ist ein solches Ähnlichkeitsmaß und ist bei gegebenen Vektoren $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$, $n \in \mathbb{N}$ definiert als

$$\langle \mathbf{x}, \mathbf{x}' \rangle := \sum_{i=1}^{n} x_i x_i'$$

mit x_i als i-ten Eintrag des Vektors $\mathbf{x}$.

Bei der Klassifikation wird eine Entscheidungsfunktion $f : X \to C,\ C \subseteq \mathbb{Z},\ |C| = l$ gesucht, welche die Muster in die l verschiedenen Klassen einteilt. Das Verfahren, welches die Klassifikation durchführt, wird **Klassifikationsverfahren**, **Klassifikator** oder auch **Klassifizierer** genannt.

Es wird zwischen unüberwachten und überwachten Klassifikationsverfahren unterschieden. Beim unüberwachten Klassifizieren sind im Vorhinein keine Klassen bekannt. Methoden zum unüberwachten Klassifizieren, wie beispielsweise Clustering, partitionieren die Muster so, dass Muster innerhalb einer Klasse einander möglichst ähnlich sind, Mustern aus anderen Klassen

jedoch möglichst unähnlich. Im Gegensatz dazu sind bei überwachten Klassifikationsverfahren Paare von Mustern und Klassenzuordnungen gegeben, aus denen das Verfahren eine Funktion entwickeln soll, die jedem Muster die richtige Klasse zuordnet.

3.1.1 Lineare Separierbarkeit

In einem n-dimensionalen Vektorraum heißen Vektormengen **linear separierbar** oder **linear trennbar**, wenn eine die Mengen trennende **Hyperebene** in diesen Vektorraum gelegt werden kann. Eine solche Ebene hat die Dimension $n-1$. Im allgemeinen Fall sind l so genannt Musterpaare oder Beispiele gegeben, die aus Merkmalsvektor und zugehöriger Klasse $(\mathbf{x}_i, y_i),\ i = 1, \ldots, l,\ \mathbf{x}_i \in X,\ y_i \in \{-1, +1\}$ bestehen. y_i gibt die Klasse an, in welche ein Muster $\mathbf{x}_i$ fällt. Wenn es nun eine Hyperebene gibt, welche die Muster der Klasse *-1* von denen der Klasse *+1* trennt, dann wird das Klassifizierungsproblem als linear separierbar bezeichnet. Die Hyperebene wird durch den zugehörigen Normalenvektor $\mathbf{w}$ und den so genannten **Bias** oder **Schwellenwert** beschrieben. Diese bestimmen die Lage der Hyperebene. $\mathbf{w}$ steht senkrecht auf der Hyperebene. Der Abstand eines Punktes von der Hyperebene wird berechnet als $\frac{|\langle \mathbf{x}_i, \mathbf{w}\rangle + \theta|}{\|\mathbf{w}\|}$, wobei $\|\mathbf{w}\|$ die euklidische Norm und damit die Länge des Normalenvektors ist. Die Entfernung der Hyperebene zum Ursprung beträgt somit $\frac{|\langle \mathbf{0}, \mathbf{w}\rangle + \theta|}{\|\mathbf{w}\|} = \frac{|\theta|}{\|\mathbf{w}\|}$.
Für die Datenpunkte $\mathbf{x}_i$, die auf der Hyperebene $HE(\mathbf{w}, \theta) = \{\mathbf{x}_i | \langle \mathbf{x}_i, \mathbf{w}\rangle = -\theta\}$ liegen, gilt $\langle \mathbf{x}_i, \mathbf{w}\rangle + \theta = 0$. Damit gilt

$$\langle \mathbf{x}_i, \mathbf{w}\rangle + \theta \geq 0, \quad \text{für } y_i = +1$$

und

$$\langle \mathbf{x}_i, \mathbf{w}\rangle + \theta < 0, \quad \text{für } y_i = -1.$$

Es kann viele verschiedene Hyperebenen geben, die dies leisten, wie in Abbildung 3.2 zu sehen [SS01].

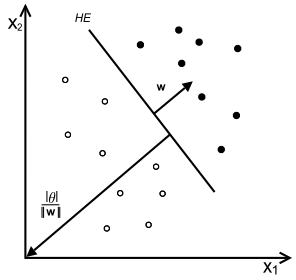

Abb. 3.1: *Trennende Hyperebene*

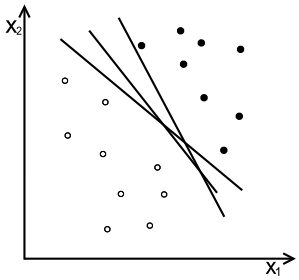

Abb. 3.2: *Mögliche trennende Hyperebenen*

3.1.2 Klassifikationsrate

Das Ziel eines Klassifikationsverfahrens ist es, einen Zusammenhang zwischen den Merkmalen der Muster, also der Eingabe, und der zugehörigen Klasse, also der Ausgabe, zu erkennen. Die Erkennungsphase wird Lernprozess genannt. Um beurteilen zu können, wie gut es dazu in der Lage ist, wird die Klassifikationsrate des Verfahrens ermittelt. Dabei gibt die Klassifikationsrate $CR \in [0, 1] \subseteq \mathbb{R}$ das Verhältnis der richtig klassifizierten zu der Gesamtmenge der klassifizierten Muster an. Je höher die Klassifikationsrate ist, desto besser ist das Verfahren in der Lage, einer Eingabe die korrekte Ausgabe, also die richtige Klasse, zuzuordnen.

3.1.3 Generalisierungsfähigkeit

Die zur Verfügung stehenden Muster werden in eine Trainingsmenge und eine dazu disjunkte Testmenge aufgeteilt. An den Mustern aus der Trainingsmenge wird das Klassifikationsverfahren trainiert. Der Klassifikator soll **generalisieren** können, das heißt er soll in der Lage sein, nach dem Training auch die Muster aus der Testmenge in die richtige Klasse einzuteilen. Diese sind dem Klassifikator bisher unbekannt, da sie nicht zum Training genutzt worden sind. Zur Überprüfung der Generalisierungsfähigkeit wird jedes Muster aus der Testmenge dem Klassifikator präsentiert und eine Ausgabe erzeugt, die einer der l Klassen entspricht. Diese Entscheidung wird mit der tatsächlichen Klasse verglichen und zeigt an, ob eine korrekte Klassifikation des Musters stattgefunden hat.

Die Klassifikationsrate der Trainingsmuster ist meist besser, als die der Testmuster. Dies liegt im Besonderen daran, dass an ihnen gelernt wird und die Muster der Testmenge unbekannt sind. Das Ziel ist es jedoch, nicht primär die Klassifikationsrate der Trainingsmenge zu erhöhen, sondern insbesondere zu generalisieren. Es besteht das Problem der **Überanpassung**. Überanpassung bedeutet, dass die Trainingsdaten „auswendig gelernt“ worden sind und eine Generalisierung nicht eingetreten ist. Das Verfahren passt sich also zu sehr den Trainingsdaten an. Dabei erzeugt der Klassifikator nur für die Trainingsmuster eine gute Klassifikationsrate, ist aber nicht in der Lage, die unbekannten Muster aus der Testmenge richtig zu klassifizieren. Zwei Methoden für die überwachte Klassifikation sind künstliche neuronale Netze und Support-Vector-Machines, die im Folgenden näher beschrieben werden.

3.2 Künstliche neuronale Netze

Dieses Unterkapitel gibt einen Überblick über das Klassifikationsverfahren künstliche neuronale Netze. Für weiterführende Erklärungen sei auf das Buch von Nauck et al. [NKK94] verwiesen. Die künstlichen neuronalen Netze (KNN) sind neben den evolutionären Algorithmen und der Fuzzy-Logik eines der drei Hauptgebiete der Computational Intelligence, die sich mit Algorithmen beschäftigt und solche entwickelt, bei denen Prozesse aus der Natur als Vorbild dienen. Die KNN sind ebenfalls biologisch motiviert. Es sind Systeme, die dem Aufbau und den Abläufen des menschlichen Gehirns nachempfunden sind. Elementarer Bestandteil ist eine Menge so genannter (künstlicher) Neuronen oder auch Verarbeitungseinheiten, die mit gewichteten Verbindungen vernetzt sind. Über diese Verbindungen findet ein Informationsfluss statt. Das KNN soll mittels Lernverfahren einen Zusammenhang zwischen der Eingabe und der Ausgabe erkennen und so nach einem Lernprozess unbekannte Muster den Klassen zuordnen, also das Erlernte anwenden. Wie gut ein KNN gelernt hat, lässt sich mit Hilfe der Klassifikationsrate ausdrücken. Die Lernalgorithmen, die in diesem Unterkapitel vorgestellten werden, verändern die Gewichte im KNN. Das Erlernte ist somit in den Gewichten zwischen den Neuronen kodiert. Diese Information ist nicht in expliziter Form aus dem KNN zu extrahieren, weswegen vom so genannten Black-Box-Verhalten gesprochen wird. Zunächst wird nun das biologische Vorbild kurz erläutert.

3.2.1 Biologisches Vorbild

Das menschliche Gehirn enthält ca. 100 Millionen Nervenzellen, die Neuronen. Diese bestehen aus einem Zellkörper, Dendriten, Synapsen und einem Axon. Im Zellkörper befindet sich ein

Zellkern. Der Aufbau eines Neurons in einem menschlichen Gehirn ist in Abbildung 3.3 zu sehen. Im Ruhezustand ist an der Membran des Neurons ein Ruhepotential von -70 Millivolt vorhanden, da die Natrium-Ionen-Konzentration im Zellkern nur einem Zehntel der Konzentration außerhalb der Zelle entspricht. Wenn ein Neuron erregt wird, also ein elektrisches Signal ankommt, wird die Membran durchlässig und viele Natrium-Ionen gelangen so in den Zellkern. Dabei strömt eine geringe Menge an Kalium-Ionen heraus. Bei der Überschreitung einer bestimmten Konzentration, dem Schwellenwert, baut sich am Axonhügel ein so genanntes Aktionspotential auf. Für eine Millisekunde beträgt das Potential an der Membran des Neurons +40 Millivolt, bevor es wieder in den Ruhezustand zurückkehrt. Mit Hilfe einer Ionen-Pumpe werden die Natrium-Ionen wieder aus dem Zellinneren entfernt. Wenn in hoher Frequenz ein Aktionspotential erzeugt wird, wird das Neuron als *aktiv* bezeichnet, es *feuert*. Über das Axon gelangt das Aktionspotential bis zu den Synapsen. Je nach Stärke der Synapse werden Neurotransmitter ausgeschüttet, die zu den Dendriten des anschließenden Neurons hinüberdiffundieren und sich dort an spezifische Rezeptoren heften. Transmitter können hemmend oder erregend wirken, je nach Beschaffenheit der Rezeptoren. Die Signale aller eingehenden Dendriten werden akkumuliert und an den Zellkörper weitergeleitet, was bei Überschreitung des Schwellenwertes zur Aktivierung dieses Neurons führt [NKK94].

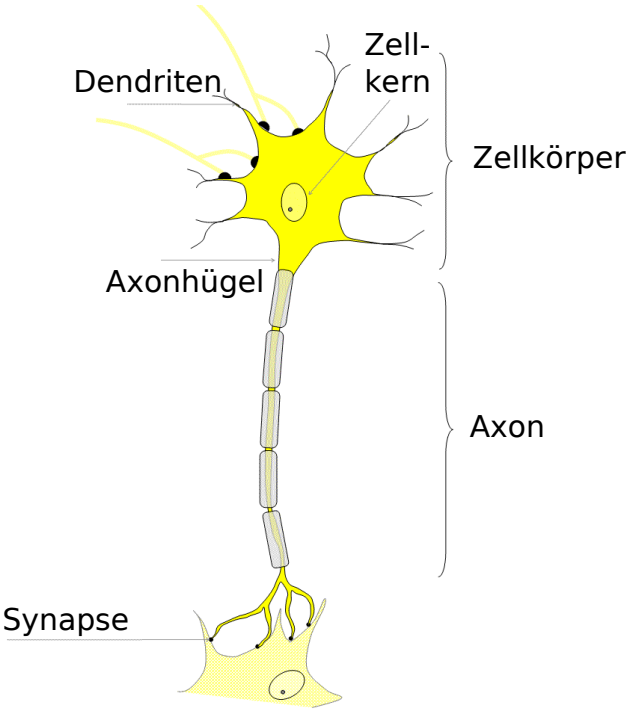

Abb. 3.3: *Aufbau eines Neurons im menschlichen Gehirn*

3.2.2 Neuronen-Modell

Das erste Modell eines künstlichen Neurons haben McCulloch und Pitts 1943 [MP43] vorgestellt. Das Modell ist als McCulloch-Pitts-Neuron bekannt und gilt als Grundlage für die weiteren Entwicklungen der künstlichen Neuronen und der KNN. Dieses ist in der Lage, logische Funktionen wie AND und OR zu lösen. Es ist jedoch nicht in der Lage zu lernen, da es

nicht-veränderbare Gewichte besitzt. Hebb hat 1949 die so gennante **Hebbsche Lernregel** aufgestellt [Heb49]. Diese besagt, wenn zwei miteinander verbundene Neuronen gleichzeitig aktiv sind, dann soll die Verbindung zwischen diesen Neuronen verstärkt werden. Dieser Ansatz wird von vielen Lernverfahren verwendet. Minsky und Papert haben das Perceptron vorgestellt [MP69]. Dieses Modell ist ein Mc-Culloch-Pitts-Neuron mit veränderbaren Gewichten. Das Perceptron wird in Unterkapitel 3.2.5 näher beschrieben.

3.2.3 Struktur des KNN

Hier wird der Aufbau eines KNN erklärt, wozu zunächst die Definitionen für ein KNN gegeben werden.

Definition 3.2.3.1 *Ein KNN ist ein Tupel* $(U,\ W,\ A,\ O,\ \text{NET},\ \text{ex})$ *mit:*

- $U = \{v_1, \ldots, v_m\}$*, Menge aller* m *Neuronen im KNN.*
- $W \subseteq U \times U$*, Netzwerkstruktur.*
- A*, Abbildung, die jedem* $v \in U$ *eine Aktivierungsfunktion* $A_v : \mathbb{R} \to \mathbb{R}$ *zuordnet.*
- O*, Abbildung, die jedem* $v \in U$ *eine Ausgabefunktion* $O_v : \mathbb{R} \to \mathbb{R}$ *zuordnet.*
- NET*, Abbildung, die jedem* $v \in U$ *eine Propagierungsfunktion* $\text{NET}_v : (\mathbb{R} \times \mathbb{R})^{|U|} \to \mathbb{R}$ *zuordnet.*
- ex*, externe Eingabefunktion* $\text{ex} : U \to \mathbb{R}$*, die einem* $v \in U$ *eine externe Eingabe in Form einer reellen Zahl* $\text{ex}_v = \text{ex}(v) \in \mathbb{R}$ *zuordnet.*

Ein künstliches Neuron ist ein stark vereinfachtes Modell seines natürlichen Vorbildes und wird durch die folgenden Elemente beschrieben.

- $\mathbf{x}_v = (x_{1v}, \ldots, x_{nv}), x_{iv} \in \mathbb{R}$, Werte der Eingänge des Neurons, n $\in \mathbb{N}$, welche entweder die externe Eingabe oder die Ausgabe von anderen Neuronen sind.
- $\mathbf{w}_v = (W(u_1, v), \ldots, W(u_n, v)) = (w_{v1}, \ldots, w_{vn})$, u_i ist Vorgängerneuron von v, $W(u_i, v) = w_{vi} \in \mathbb{R}$, Gewichtswerte des Eingänge.
- A_v, Aktivierungsfunktion mit einem Schwellenwert/Bias θ_v.
- O_v, Ausgabefunktion.
- NET_v, Propagierungsfunktion.

Der Aufbau eines künstlichen Neurons ist in Abbildung 3.4 zu sehen.

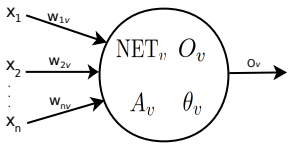

Abb. 3.4: *Aufbau eines künstlichen Neurons*

Die Eingänge entsprechen den Dendriten im biologischen Vorbild. Die Gewichte drücken die Stärke der Signale aus. Der Schwellenwert des Neurons ist vergleichbar mit dem Schwellenwert des menschlichen Neurons. Die Propagierungsfunktion (auch Übertragungsfunktion) entspricht der Akkumulation der Signale, die von den Dendriten an den Zellkern gelangen. Beim KNN ist dies die gewichtete Summe der Eingänge.

Netzwerkstruktur W ist eine Konnektivitäts- oder Gewichtsmatrix. $W(u, v)$ ist ein Gewicht, welches der Kante zwischen u und v zugeordnet ist. Durch die Gewichtung kann ein Neuron ein ankommendes Signal stärker oder weniger stark berücksichtigen. $W(u, v) = 0$ bedeutet, dass diese Kante nicht existiert. $W(u, v) > 0$ wird excitatorisch oder anregend, $W(u, v) < 0$ inhibitorisch oder hemmend genannt. Ein KNN ist in verschiedenen Schichten aufgebaut. Der Aufbau eines KNN ist in der Abbildung 3.6 des Multilayer-Perceptrons zu sehen. Es wird zwischen der Eingabeschicht, den inneren Schichten (versteckte Schichten) und der Ausgabeschicht unterschieden. In der Abbildung 3.6 stellt U_1 die Eingabe-, U_t die Ausgabe- und U_2 eine innere Schicht dar. Die Neuronen auf der Eingabeschicht repräsentieren die Eingabe für das KNN, die Neuronen auf der Ausgabeschicht die Ausgabe des KNN. Neuronen der inneren Schichten verarbeiten die Ausgaben ihrer Vorgängerneuronen und leiten ihre Ausgabe an die Neuronen auf der nächsten Schicht weiter.

Externe Eingabefunktion Die externe Eingabefunktion ex stellt die Verbindung des neuronalen Netzes mit der Umgebung da. Neuronen auf der Eingabeschicht werden mithilfe der externen Eingabefunktion auf die Eingabewerte für das Netz initialisiert und stellen so die Gesamteingabe für das KNN dar. Diese initialen Werte entsprechen den Merkmalswerten des Musters, welches dem Netz präsentiert wird. Die externe Eingabefunktion ist nur für die Neuronen der Eingabeschicht definiert.

Eingänge Die Werte der Eingänge des Neurons werden entweder durch die Ausgaben der Vorgängerneuronen bestimmt oder von der externen Eingabefunktion gesetzt, je nachdem, auf welcher Schicht sich das Neuron befindet.

Aktivierungsfunktion A_v bestimmt den aktuellen Aktivierungszustand a_v des Neurons v. Er berechnet sich zu $A_v(\text{net}_v) = a_v$. Mögliche Aktivierungsfunktionen sind die folgenden, wobei die logistische Funktion zur Menge der sigmoiden Funktionen gehört.

- Lineare Schwellenwertfunktion $a_v = \begin{cases} 1 & \text{falls } \text{net}_v > \theta_v \\ 0 & \text{sonst} \end{cases}$
- Lineare Funktion $a_v = c_v \cdot \text{net}_v$, mit $c_v \in \mathbb{R}$ als Konstante.
- Logistische Funktion $a_v = \frac{1}{1+e^{-\text{net}_v}}$

Ausgabefunktion Die Ausgabefunktion $O_v(a_v) = o_v$ bildet die Aktivierung eines Neurons auf eine Ausgabe ab. Meist ist die Aktivierung identisch zur Ausgabe und daher wird die Identitätsfunktion genutzt. Die Identitätsfunktion gibt die Eingabe unverändert aus, was zu $o_v = a_v$ führt.

Propagierungsfunktion Die Propagierungsfunktion (auch Netzeingabefunktion) berechnet die gewichtete Summe der Eingänge eines Neurons $v \in U_i$ mit

$$\text{net}_v = \sum_{u \in U_{i-1}} W(u, v) \cdot o_u.$$

3.2.4 Lernprozess des KNN

Das Lernen des KNN findet durch die Veränderung der Gewichte $W(u, v)$ in der Gewichtsmatrix W und der Schwellenwerte statt. Ziel des Lernens ist es, die Gewichte und Schwellenwerte so zu modifizieren, dass beim Anlegen einer bestimmten Eingabe die passende Ausgabe erzeugt wird und somit nach dem Training auf neue, unbekannte Eingaben angemessen reagiert werden kann. Die Trainingsmuster werden dem KNN in so genannten **Epochen** präsentiert. In einer solchen Epoche wird jedes einzelne Trainingsmuster genau einmal als Eingabe angelegt. Nach jeder Epoche wird die produzierte Ausgabe mit der tatsächlichen Ausgabe verglichen und W und θ entsprechend angepasst. Diese Anpassung nehmen die Lernalgorithmen vor, die in diesem Kapitel beschrieben werden. So nähert sich die produzierte Ausgabe der tatsächlichen Ausgabe immer mehr an. Für die Anpassung wird ein Fehlermaß benötigt, welches in Definition 3.2.4.3 beschrieben ist. Der Fehler des Klassifikators für die Trainingsmenge sinkt nach jeder Epoche. Auch der Fehler der Testmenge verringert sich zunächst. Ab einem bestimmten Zeitpunkt wird sich bei immer noch sinkendem Fehler der Trainingsmenge der Fehler der Testmenge wieder erhöhen. An diesem Zeitpunkt soll das Training abgebrochen werden, da nun die Generalisierungsfähigkeit des Klassifikators abnimmt und die so genannte Überanpassung einsetzt. Ein typischer Fehlerverlauf ist in Abbildung 3.5 zu sehen. An der Abszisse sind die Epochen abgetragen und an der Ordinate der jeweilige Fehler der Epoche (Mean-Squared-Error, vergleiche Abschnitt 3.2.4).

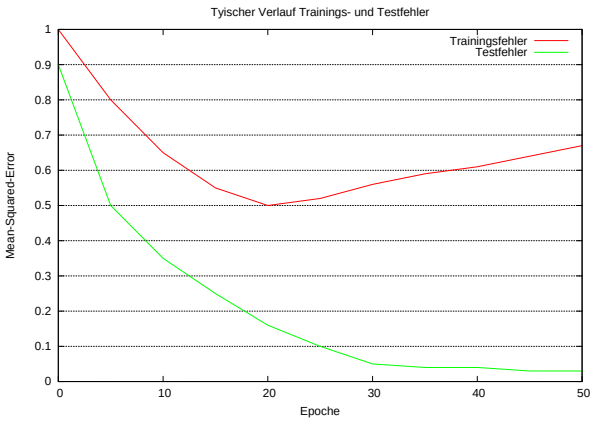

Abb. 3.5: *Verlauf der Klassifikationsfehler der Trainings- und Testmustermenge über die Epochen*

Im Folgenden werden Begriffe in Anlehnung an Nauck et al. [NKK94] erläutert, auf die in späteren Beschreibungen zurückgegriffen wird. Die Eingabe und Ausgabe eines neuronalen

Netzes sind wie folgt beschrieben.

Definition 3.2.4.1 *Sei NN ein beliebiges neuronales Netz mit einer Menge von Verarbeitungseinheiten U, der externen Eingabefunktion* ex *und der für eine Ausgabeeinheit v gültigen Ausgabefunktion O_v. $U_I = \{u_1, \ldots, u_n\}$ sei die Menge der Eingabeeinheiten und $U_O = \{v_1, \ldots, v_m\}$ die Menge der Ausgabeeinheiten. Weiterhin bezeichne dom den Bildbereich (Domäne) einer Funktion.*

- *Eine Eingabe (Eingabemuster) i von NN ist ein Element der Menge $I = \{(x_1, \ldots, x_n) | x_i \in dom(\text{ex})\} \subseteq \mathbb{R}^{|U_I|}$.*
- *Eine Ausgabe (Ausgabemuster) t von NN ist ein Element der Menge $T = \{(y_{v_1}, \ldots, y_{v_m}) | y_{v_i} \in dom(O_{v_i})\} \subseteq \mathbb{R}^{|U_O|}$.*

Eine feste Lernaufgabe (überwachte Klassifikation) für ein neuronales Netz ist wie folgt definiert.

Definition 3.2.4.2 *Eine feste Lernaufgabe $L \subseteq I \times T$ eines neuronalen Netzes ist eine Menge von Ein-/Ausgabepaaren. Das neuronale Netz soll auf jede Eingabe i eines Paares $(i, t) \in L$ mit der dazugehörigen Ausgabe t reagieren. Ein neuronales Netz erfüllt die feste Lernaufgabe, wenn es zu jeder Eingabe die laut Lernaufgabe dazugehörige Ausgabe erzeugt.*

Ein Fehlermaß drückt den Fehler aus, den ein Lernverfahren produziert und ist folgendermaßen definiert.

Definition 3.2.4.3 *Sei L eine feste Lernaufgabe und $p \in L$ ein zu lernendes Musterpaar.*

- *Ein Fehlermaß für einen überwachten Lernalgorithmus ist eine Abbildung*

$$e : \mathbb{R}^{|U_O|} \times \mathbb{R}^{|U_O|} \to \mathbb{R}_{\geq 0},$$

  so dass $\forall a, b \in \mathbb{R}^{|U_O|}$ gilt : $e(a, b) = 0 \Leftrightarrow a = b$. Der Fehler $e^{(p)}$, den das neuronale Netz bei der Verarbeitung des Musterpaares $p \in L$ macht, ist durch $e^{(p)} = e(t^{(p)}, o^{(p)})$ gegeben. Dabei ist $o^{(p)}$ die Ausgabe des Netzes auf das Eingabemuster $i^{(p)}$ und $t^{(p)}$ die erwünschte, durch die Lernaufgabe L vorgegebene Ausgabe (target).

- *Der Fehler $e_v^{(p)}$ einer Ausgabeeinheit $v \in U_o$ bei der Verarbeitung des Musterpaares $p \in L$ ist die Differenz zwischen dem durch die Lernaufgabe vorgegebenen Ausgabewert $t_v^{(p)}$ und dem tatsächlichen Ausgabewert $o_v^{(p)}$:*

$$e_v^{(p)} = t_v^{(p)} - o_v^{(p)}.$$

Das Fehlermaß $E = E_{SSE} \in \mathbb{R}$ (Sum-Squared-Error) berechnet die Summe über die quadrierten Einzelfehler der Ausgabeeinheiten aller Musterpaare. Für ein Musterpaar $p \in L$ berechnet sich der Fehler durch

$$e^{(p)} = \sum_{v \in U_O} (e_v^{(p)})^2 = \sum_{v \in U_O} (t_v^{(p)} - o_v^{(p)})^2$$

und der Fehler über alle Musterpaare durch

$$E_{SSE} = \sum_{p \in L} e^{(p)} = \sum_{p \in L} \sum_{v \in U_O} (t_v^{(p)} - o_v^{(p)})^2.$$

Dieses Fehlermaß wird auch gemittelt über die Anzahl der Trainingsbeispiele angegeben. Es wird als $E = E_{MSE}$ (Mean-Squared-Error) bezeichnet und berechnet durch

$$E_{MSE} = \frac{1}{|L|} \sum_{p \in L} \sum_{v \in U_O} (t_v^{(p)} - o_v^{(p)})^2.$$

Die Quadrierung des Fehlers ist deshalb wichtig, damit die Fehler in positive und negative Richtung sich nicht gegenseitig aufheben. Zudem werden große Abweichungen verhältnismäßig wesentlich negativer bewertet als kleine Abweichungen. Ein Fehlermaß kann als Abbruchkriterium für ein Lernverfahren dienen. Nach jeder Epoche wird dieser Fehler mit einem vorher bestimmten Wert verglichen. Wenn der Gesamtfehler des Netzes diesen Wert unterschreitet, wird das Training abgebrochen. Ein anderes Abbruchkriterium kann eine maximale Anzahl an Epochen sein, nach deren Erreichen das Lernverfahren stoppt.

Ein typischer Ablauf eines Lernverfahrens sieht die folgenden Schritte vor.

1. Präsentiere dem neuronalen Netz die Eingabe eines Musterpaares $p \in L$.
2. Berechne die Ausgabe des KNN.
3. Vergleiche Ist- und Sollausgabe und addiere den Fehler dieses Musterpaares zum Gesamtfehler.
4. Wenn der Fehler ungleich 0 ist, also die Ausgabe des Netzes nicht der vorgegebenen Ausgabe entspricht, verändere die Gewichte in W entsprechend.
5. Wiederhole die Schritte 1-4 für alle Musterpaare in L.
6. Wenn der Fehler den vorher bestimmten maximalen Fehler unterschreitet oder die maximale Anzahl an Epochen erreicht wird, wird abgebrochen, sonst wird mit 1 fortgefahren.

Der Lernvorgang soll zum einen die Lernaufgabe erfolgreich lösen. Zum anderen soll jedoch eine möglichst große Generalisierungsfähigkeit erreicht werden, also im Besonderen sollen neue Muster, die am Lernvorgang nicht teilgenommen haben, richtig klassifiziert werden. Es sollen also zu ähnlichen Eingaben auch ähnliche Ausgaben hervorgerufen werden, Muster also wiedererkannt werden. Ein Modell, welches in der Lage ist zu lernen, wird im Folgenden vorgestellt.

3.2.5 Perceptron

Das Perceptron ist die Grundlage für das Multilayer-Perceptron, welches in Abschnitt 3.2.6 vorgestellt wird. Es besteht nur aus einem Neuron und dessen Eingaben gewichtet sind. Das Neuron summiert diese gewichtete Eingabe. Wenn diese Summe den Schwellenwert überschreitet, wird es aktiv, ansonsten bleibt es inaktiv. Aktiv bedeutet, dass die Ausgabe 1 ist, ansonsten 0. Es verarbeitet nur binäre Eingabemuster und erzeugt eine binäre Ausgabe. Für die nun folgende Definition wird das Perceptron in ein KNN überführt [NKK94]. Es besteht aus einer Eingabeschicht mit so vielen Einheiten, wie das Perceptron Eingänge besitzt, wobei die Neuronen auf dieser Schicht keine Verarbeitungen durchführen, sondern ihre Ausgabe lediglich den Werten des Eingabemusters entspricht.

Definition 3.2.5.1 *Ein Perceptron ist ein künstliches neuronales Netz* $P = (U, W, A, O, \mathrm{NET}, \mathrm{ex})$*, wobei gilt:*

- $U = U_I \cup U_O$ *ist eine Menge von Verarbeitungseinheiten, mit* $U_I \neq \emptyset, U_I \cap U_O = \emptyset$ *und* $U_O = \{v\}$*, wobei* U_I *Eingabeschicht und* U_O *(einelementige) Ausgabeschicht genannt wird.*

- *Die Netzwerkstruktur ist* $W : U_I \times U_O \to \mathbb{R}$*, wobei lediglich Verbindungen* $W(u, v)$ *von der Eingabe- zur Ausgabeschicht existieren* ($u \in U_I,\ v \in U_O$)*.*

- *A ordnet jeder Einheit* $u \in U$ *eine Aktivierungsfunktion* $A_u : \mathbb{R} \to \{0, 1\}$ *zur Berechnung der Aktivierung* a_u *zu, wobei* $a_u = A_u(\mathrm{ex}(u)) = \mathrm{ex}(u)\ \forall u \in U_I$ *gilt, und für* $v \in U_O$ *eine lineare Schwellenwertfunktion zur Bestimmung der Aktivierung*

$$a_v = A_v(\mathrm{net}_v) = \begin{cases} 1, & \textit{falls } \mathrm{net}_v \leq \theta_v \\ 0, & \textit{falls } \mathrm{net}_v > \theta_v \end{cases}$$

  verwendet wird. $\theta_v \in \mathbb{R}$ *ist dabei ein Schwellenwert.*

- *O ordnet jeder Einheit* $u \in U$ *eine Ausgabefunktion* $O_u : \mathbb{R} \to \{0, 1\}$ *zur Berechnung der Ausgabe* o_u *zu, so dass* $o_u = O_u(a_u) = a_u\ \forall u \in U$ *gilt*

- NET *ordnet der Ausgabeeinheit* $v \in U_O$ *eine Propagierungsfunktion* $\mathrm{NET}_v : \mathbb{R}^{|U_I|} \times \mathbb{R}^{|U_I| \times |U_O|} \to \mathbb{R}$ *zur Berechnung der Netzeingabe* net_v *zu. Sie berechnet sich zu einer gewichteten Summe über die Ausgaben der Eingabeeinheiten:* $\mathrm{net}_v = \sum_{u \in U_I} W(u, v) \cdot o_u$*.*

- $\mathrm{ex} : U_I \to \{0, 1\}$ *ordnet jeder Eingabeeinheit* $u \in U_I$ *ihre externe Eingabe* $\mathrm{ex}_u = \mathrm{ex}(u)$ *zu.*

Damit ein Perceptron eine Lernaufgabe lösen kann, muss diese *linear separierbar* sein. Ein Perceptron kann eine feste Lernaufgabe mit Hilfe eines überwachten Lernalgorithmus bewältigen. Die Lernregel, die das Perceptron benutzt, ist die so genannte Delta-Regel (vergleiche Definition 3.2.5.2). Dabei werden sowohl die Gewichte als auch der Schwellenwert verändert. Die Delta-Regel ist folgendermaßen definiert [NKK94]. Die Änderungen werden mit Δ bezeichnet, welches die Namensgebung für die Regel erklärt.

Definition 3.2.5.2 *Gegeben sei ein neuronales Netz LN mit linearer Aktivierungsfunktion und eine feste Lernaufgabe L. Die Gewichtsveränderung* $\Delta_p W(u, v)$ *für* $u \in U_I$ *und* $v \in U_O$ *sowie die Schwellenwertänderung* $\Delta_p \theta_v$ *werden nach der Propagation der Eingabe eines Musterpaares* $p \in L$ *wie folgt bestimmt:*

$$\Delta^{(p)} W(u, v) = \eta \cdot (t_v^{(p)} - a_v^{(p)}) \cdot a_u^{(p)}$$

$$\Delta^{(p)} \theta_v = \eta \cdot (t_v^{(p)} - a_v^{(p)}).$$

Dabei ist $t_v^{(p)}$ *die für die Ausgabeeinheit* $v^{(p)}$ *vorgesehene Aktivierung,* $a_v^{(p)}$ *deren tatsächliche Aktivierung und* $a_u^{(p)}$ *die mit der Eingabe übereinstimmende Aktivierung der Eingabeeinheit u für das Musterpaar p.* $\eta \in \mathbb{R}_{>0}$ *wird Lernrate genannt.*

3.2.6 Multilayer-Perceptron

Mit dem vorgestellten Perceptron können nur linear separierbare Probleme gelöst werden. Um auch solche Probleme lösen zu können, die nicht linear separierbar sind, wird das Perceptron-Modell verändert und unter anderem durch nicht-lineare Aktivierungsfunktionen erweitert. Zudem kann die Ausgabeschicht mehrelementig sein und somit zwischen mehr als nur zwei Klassen unterschieden werden. Außerdem müssen sowohl die Eingabe- als auch die Ausgabemuster nicht mehr binärwertig sein, sondern können reelle Werte annehmen. Beim Modell des Multilayer-Perceptron werden versteckte Schichten mit Neuronen eingeführt. Neuronen auf derselben Schicht sind nicht miteinander und auch nicht mit sich selbst verbunden sind, sondern nur mit den Neuronen auf der nächsthöheren Schicht. Es existieren demnach auch keine Verbindungen, die auf eine niedrigere Schicht zurückführen und auch keine *Short-Cuts*, bei denen eine Verbindung eine oder mehrere Schichten „überspringt“. Ein schematischer Aufbau eines solchen Netzes ist in Abbildung 3.6 zu sehen.

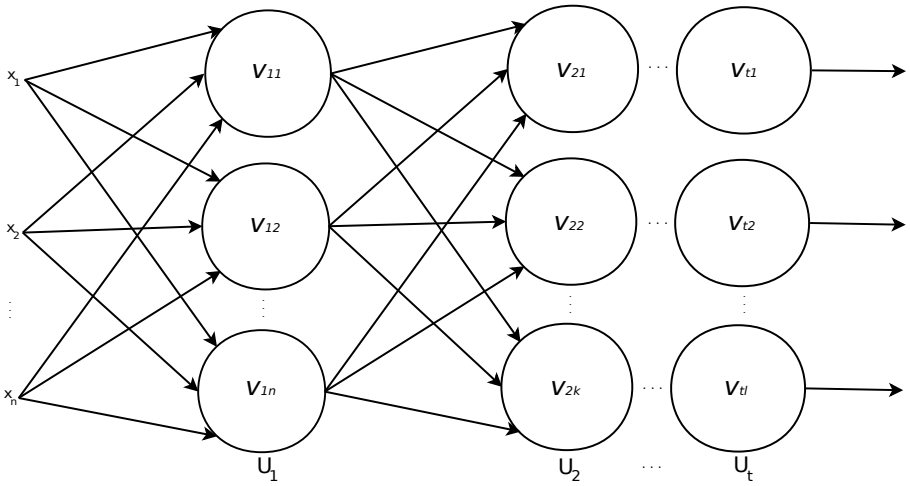

Abb. 3.6: *Aufbau eines Multilayer-Perceptron: n Neuronen auf der Eingabeschicht 1, k Neuronen auf der versteckten Schicht 2 und l Neuronen auf der Ausgabeschicht t. Die Verbindungsgewichte zwischen den Neuronen sind der Übersichtlichkeit halber nicht abgebildet.*

Da nun in mehr als nur zwei Klassen klassifiziert werden kann, muss die Ausgabe entsprechend kodiert sein. Bei beispielsweise drei Klassen sind so drei Neuronen auf der Ausgabeschicht vorhanden, wobei jedes Neuron für eine Klasse steht. Die Ausgabemuster der Trainingsmenge haben demnach die Form (1 0 0) für die erste, (0 1 0) für die zweite und (0 0 1) für die dritte Klasse. Die Entscheidung, in welche Klasse ein Muster fällt, hängt von der höchsten Aktivierung des jeweiligen Neurons ab. Beispielsweise führt die vom Multilayer-Perceptron errechnetet Ausgabe (0.2, 0.6, 0.9) somit zu einer Einordung in die dritte Klasse.

Bias-Neuron Damit für die Lernverfahren die Anpassung der Schwellenwerte θ_v erleichtert wird, wird ein so genanntes **Bias-Neuron** in das KNN integriert. Dazu wird ein Neuron in das Netz aufgenommen, welches konstant den Wert -1 ausgibt. Dieses Bias-Neuron ist mit jedem Neuron im KNN verbunden. Die Verbindungen werden mit θ_v, also den Schwellenwerten der verbundenen Neuronen gewichtet. So fließt der Schwellenwert durch die Anwendung der Propagierungsfunktion in die Netzeingabe ein. Die Schwellenwerte der Neuronen selbst werden

auf 0 gesetzt. Der Schwellenwert wird jetzt über die Anpassung der Gewichte durch die Lernverfahren verändert. Es entsteht ein äquivalentes Netz, bei dem lediglich die Schwellenwerte durch Verbindungsgewichte realisiert werden.
Ein Multilayer-Perceptron wird folgendermaßen in Anlehnung an Nauck et al. [NKK94] definiert.

Definition 3.2.6.1 *Ein Multilayer-Perceptron ist ein neuronales Netz MLP = (U, W, A, O,* NET, ex*), das folgende Charakteristika aufweist:*

- $U = U_1 \cup \ldots \cup U_t$ *ist eine Menge von Verarbeitungseinheiten, wobei* $t \geq 3$ *vorausgesetzt wird. Es gilt dabei,* $U_i \neq \emptyset \ \forall i \in \{1, \ldots, t\}$ *und* $U_i \cap U_j = \emptyset$ *für* $i, j \in \{1, \ldots, t\}$, $i \neq j$. U_1 *heißt Eingabeschicht und* U_t *Ausgabeschicht. Die* U_i *mit* $1 < i < t$ *werden innere (versteckte) Schichten genannt.*

- *Die Netzwerkstruktur ist durch die Abbildung* $W : U \times U \rightarrow \mathbb{R}$ *festgelegt, wobei nur Verbindungen zwischen direkt aufeinanderfolgendende Schichten existieren. Formal gilt also* $W(u, v) \Leftrightarrow u \in U_i, \ v \in U_{i+1} \ \forall i \in \{1, \ldots, t-1\}$.

- *A ordnet jeder Einheit* $u \in U$ *eine Aktivierunsfunktion* $A_u : \mathbb{R} \rightarrow [0, 1]$ *zur Berechnung der Aktivierung* a_u *zu, mit* $a_u = A_u(\mathrm{ex}(u)) = \mathrm{ex}(u) \ \forall u \in U_1$ *und mit* $a_u = A_u(\mathrm{net}_u) = f(\mathrm{net}_u) \ \forall u \in U_i, \ (i \in \{2, \ldots, n\})$. $f : \mathbb{R} \rightarrow [0, 1]$ *ist dabei eine für alle Einheiten fest gewählte nicht-lineare Funktion.*

- *O ordnet jeder Einheit* $u \in U$ *eine Ausgabefunktion* $O_u : \mathbb{R} \rightarrow [0, 1]$ *zur Berechnung der Ausgabe* o_u *zu, wobei* $o_u = O_u(a_u) = a_u \ \forall u \in U$ *gilt.*

- NET *ordnet jeder Einheit* $v \in U_i$, *mit* $2 \leq i \leq t$ *eine Propagierungsfunktion* $\mathrm{NET}_v : (\mathbb{R} \times \mathbb{R})^{|U_{i-1}|} \rightarrow \mathbb{R}$ *zur Berechnung der Netzeingabe* net_v *zu, mit*

$$\mathrm{net}_v = \sum_{u \in U_{i-1}} W(u, v) \cdot o_u.$$

  $\theta_v \in \mathbb{R}$, *der Bias der Einheit v, ist mithilfe eines Bias-Neurons in die Netzwerkstruktur integriert.*

- $\mathrm{ex} : U_1 \rightarrow [0, 1]$ *ordnet jeder Eingabeeinheit* $u \in U_1$ *ihre externe Eingabe* $\mathrm{ex}_u = \mathrm{ex}(u)$ *zu.*

Die Aktivierungsfunktionen für die Neuronen der versteckten Schichten und der Ausgabeschicht müssen nicht-linear sein, da sich sonst kein Unterschied zu einschichtigen Netzen einstellt. Mehrschichtige Netze mit linearen Aktivierungsfunktionen können auf ein einschichtiges Netz zurückgeführt werden [NKK94]. Nicht-lineare Aktivierungsfunktionen müssen zudem stetig sein, da für die Lernalgorithmen, die in Abschnitt 3.2.7 beschrieben werden, diese Funktion differenzierbar sein muss [NKK94]. Die in Abschnitt 3.2.3 genannte logistische Funktion ist ein Beispiel dafür, welche auch in den meisten Fällen Anwendung findet. Die Aktualisierung der Gewichte im Multilayer-Perceptron wird mithilfe der Lernverfahren vorgenommen, die im kommenden Abschnitt vorgestellt werden.

3.2.7 Lernverfahren

Wie im Abschnitt 3.2.4 erklärt, wird das Lernen durch die Veränderung der Gewichte zwischen den Neuronen realisiert. Im Folgenden werden die Verfahren Backpropagation und Resilient-Propagation vorgestellt, wobei letzteres auf dem Backpropagation-Verfahren aufbaut.

3.2.7.1 Backpropagation

Der Begriff Backpropagation ist darauf zurückzuführen, dass der Fehler, den ein Netz produziert, zurück (*back*) in Richtung der Eingabeschicht in das Netz *propagiert* wird. So wird der Fehler an die einzelnen Neuronen geleitet. Für diese kann nun der eigene Fehler bestimmt werden, welcher die Grundlage zur Anpassung der Gewichte darstellt. Backpropagation verallgemeinert die in Definition 3.2.5.2 vorgestellte Delta-Regel auf mehrschichtige Netze, weshalb das Verfahren auch *verallgemeinerte Delta-Regel* genannt wird. Der Backpropagation-Algorithmus ist ein Gradientenabstiegsverfahren. Dabei wird ein Minimum der Fehlerfunktion gesucht, die durch die Gewichtsmatrix W definiert ist. Der Gradient eines Punktes der Fehlerfunktion ist ein Vektor, der in Richtung des steilsten Anstiegs zeigt und dessen Steigungsgrad durch seinen Betrag festgelegt ist. Der negative Gradient zeigt demnach in Richtung des steilsten Abstiegs. Jedes Gewicht in W wird dann proportional zur partiellen Ableitung des Fehlers E angepasst:

$$\Delta W(u,v) \propto -\frac{\partial E}{\partial W(u,v)}.$$

In Abbildung 3.7 ist ein Gradientenabstieg zu sehen. Zunächst wird der negative Gradient für den Gewichtsvektor w_1 bestimmt. Dieser Gradient wird um die durch die Lernrate bestimmte Länge herabgestiegen. Für den resultierenden Gewichtsvektor wird wiederum der negative Gradient bestimmt und herabgegangen, bis das Minimum bei w_4 erreicht wird.
Die Anpassung der Gewichte durch den Backpropagation-Algorithmus wird in Definition 3.2.7.1 in Anlehnung an Nauck et al. [NKK94] gezeigt.

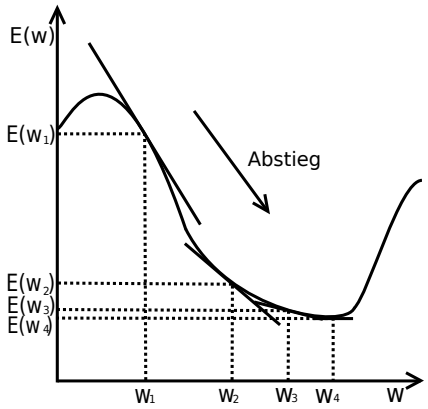

Abb. 3.7: *Gradientenabstiegsverfahren*

Definition 3.2.7.1 *Gegeben seien ein Multilayer-Perceptron MLP und* $U = U_1 \cup \ldots \cup U_n$ *mit der sigmoiden Aktivierungsfunktion* $A_u(\text{net}_u) = f(\text{net}_u)\ \forall u \in U_i,\ i \geq 2$, *sowie eine feste Lernaufgabe L. Der überwachte Lernalgorithmus, der die Veränderung der Gewichtsstruktur W von MLP nach der Propagation des Eingabemusters* $i^{(p)},\ p \in L$ *durch*

$$\Delta W(u,v) = \eta \cdot \delta_v^{(p)} \cdot o_u^{(p)}$$

mit $u \in U_{i-1},\ v \in U_i,\ 2 \leq i \leq n$ *und* $\eta > 0$ *bestimmt, wobei*

$$\delta_u^{(p)} = \begin{cases} f'(\text{net}_u^{(p)}) \cdot (t_u^{(p)} - o_u^{(p)}), & \text{falls } u \in U_n \\ f'(\text{net}_u^{(p)}) \cdot \sum\limits_{v \in U_{j+1}} \delta_v^{(p)} \cdot W(u,v), & \text{falls } u \in U_j,\ 2 \leq j \leq n-1 \end{cases}$$

gilt, heißt verallgemeinerte Delta-Regel oder Backpropagation-Algorithmus. Dabei ist $o_u^{(p)}$ *die Ausgabe der Einheit* u *nach der Propagation des Eingabemusters* $i^{(p)}$ *und* $t_v^{(p)}$, $v \in U_n$ *ist die durch das Ausgabemuster* $t^{(p)}$ *für eine Ausgabeeinheit* u_n *vorgegebenen Ausgabe (Aktivierung). Der Faktor* $\eta \in \mathbb{R}_{>0}$ *ist die Lernrate.*

Für $v \in U_i,\ 2 \leq i \leq n$ *wird* $\text{net}_v^{(p)}$ *berechnet durch*

$$\text{net}_v^{(p)} = \sum_{u \in U_{i-1}} W(u,v) \cdot o_u^{(p)},$$

wobei der Bias $\theta_v \in \mathbb{R}$ *der Einheit* v *mithilfe eines Bias-Neurons in die Netzwerkstruktur integriert ist.*

Bei der Berechnung der $\delta_v^{(p)}$ wird unterschieden, ob es sich bei einem Neuron um ein Ausgabe- oder ein verstecktes Neuron handelt. Durch die rekursiven Berechnungen der δ-Werte kann der Fehler rückwärts durch das Netz propagiert und so die Gewichtsanpassung für jedes Neuron vorgenommen werden. Für weitere Ausführungen sei auf das Buch von Nauck et al. [NKK94] verwiesen.

Probleme Beim Backpropagation-Algorithmus wird nicht zwangsläufig ein globales Minimum der Fehlerfunktion gefunden, sondern oft nur lokale Minima. Desweiteren können steile Schluchten in der Fehlerfunktion zur Oszillation führen, also das Minimum wird wiederholt übersprungen. So genannte Plateaus in der Fehlerfunktion bedeuten einen kleinen Betrag des Gradienten und damit eine nur geringe Gewichtsveränderung. Zudem gibt es bisher keine Methoden, welche eine optimale Lernrate berechnen. Zu große Lernraten können zum Verlassen eines globalen Minimum führen, aber auch die Chance bieten, aus lokalen Minima zu entkommen. Zu kleine Lernraten wiederum benötigen zu viele Schritte, um ein Minimum zu finden und können bei Plateaus fast zum Stillstand führen, was dann eine Unterscheidung zu einem (lokalen) Minimum nicht mehr zulässt. Veranschaulicht sind diese Probleme in den Abbildungen 3.8 bis 3.10.

Abhilfe bei Oszillation und Plateaus kann der so genannte *Momentum-Term* bringen [RHW86]. Dabei taucht ein neuer Term in der Gewichtsänderungsfunktion auf, bei dem die Gewichtsänderung des vor dem Muster p betrachteten Musters q gewichtet mit einbezogen wird:

$$\Delta^{(p)} W(u,v) = \eta \cdot \delta_v^{(p)} \cdot a_u^{(p)} + \beta \cdot \Delta^{(p)} W(u,v).$$

$\eta \in \mathbb{R}_{>0}$ ist die bekannte Lernrate. Der Parameter $\beta \in \mathbb{R}_{\geq 0}$ gibt an, wie stark der zuvor berechnete Gradient in die Berechnung der neuen Gewichtsänderung eingeht. Dadurch wird das Lernverfahren träger und zeigt eher eine Tendenz dazu, die Richtung der Änderung beizubehalten.
Eine weitere Methode, mit Plateaus umzugehen, ist die so genannte *Flat-Spot-Elimination.*

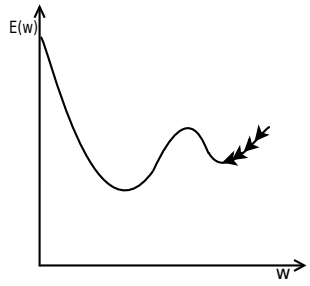

Abb. 3.8: *Lokales Minimum*

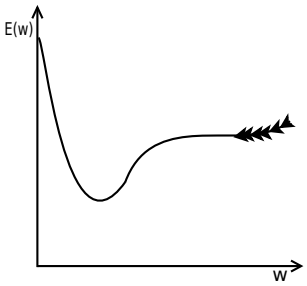

Abb. 3.9: *Plateau*

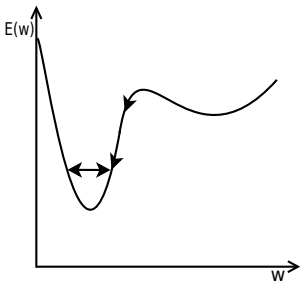

Abb. 3.10: *Oszillation*

Da bei der Ableitung der Aktivierungsfunktion Werte nahe 0 herauskommen können, was zu einer geringen Änderung der Gewichte führt, wird auf diese Ableitung noch der Flat-Spot-Elimination-Term addiert, der es ermöglicht, derartige Plateaus wieder zu verlassen (vergleiche [Fah88]).

Um eine Überanpassung des Netzes einzuschränken, kann ein maximaler Fehler d_{max} festgelegt werden, der noch toleriert wird. Ein Wert von 0.1 gibt an, dass eine Differenz von 0.1 zwischen tatsächlicher und produzierter Ausgabe als Fehler mit dem Wert 0 zurückpropagiert wird.

3.2.7.2 Resilient-Propagation

Resilient-Propagation (auch Resilient-Backpropagation) ist von Braun [Bra97] vorgestellt worden. Das Grundprinzip dieses Algorithmus ist es, dass die Größe der partiellen Ableitung der Fehlerfunktion keinen Einfluss mehr auf die Änderung der Gewichte hat, sondern nur noch das Vorzeichen der Ableitung. Die Höhe der Gewichtsänderung wird in dem so genannten *Update-Value* berechnet. Die Gewichtsänderung $\Delta W(u,v)$ berechnet sich, wie in Definition 3.2.7.2 angegeben.

Definition 3.2.7.2

$$\Delta W(u,v)^{(t)} = \begin{cases} -\Delta_{(u,v)}^{(t)}, & falls\ \frac{\partial E}{\partial W(u,v)}^{(t)} > 0 \\ +\Delta_{(u,v)}^{(t)}, & falls\ \frac{\partial E}{\partial W(u,v)}^{(t)} < 0 \\ 0, & sonst \end{cases}$$

mit t als Zeitpunkt. Die Höhe der Gewichtsveränderung wird zu jeder Epoche neu bestimmt durch

$$\Delta_{(u,v)}^{(t)} = \begin{cases} \eta^+ \Delta_{(u,v)}^{(t-1)}, & falls\ \frac{\partial E}{\partial W(u,v)}^{(t-1)} \frac{\partial E}{\partial W(u,v)}^{(t)} > 0 \\ \eta^- \Delta_{(u,v)}^{(t-1)}, & falls\ \frac{\partial E}{\partial W(u,v)}^{(t-1)} \frac{\partial E}{\partial W(u,v)}^{(t)} < 0 \\ \Delta_{(u,v)}^{(t-1)}, & sonst \end{cases}$$

mit $0 < \eta^- < 1 < \eta^+$ als Lernraten. Der Fehler für ein Musterpaar $p \in L$ lautet

$$e^{(p)} = \sum_{v \in U_O} (t_v^{(p)} - o_v^{(p)})^2 + 10^{-\alpha} \sum_{r,s \in U} W(r,s)^2,\ \alpha \in \mathbb{R}.$$

Wenn das Vorzeichen der Ableitung sich von Zeitpunkt $t-1$ zu t ändert, ist ein Minimum übersprungen worden. Dann wird der Wert für die Gewichtsanpassung um η^- reduziert, um sich dem Minimum der Fehlerfunktion langsamer anzunähern. Wenn das Vorzeichen unverändert bleibt, wird der Wert für die Gewichtsanpassung vergrößert, um schneller zu einem Minimum zu gelangen. Die Höhe der Anpassungswerte sind durch Δ_{max} nach oben und durch Δ_{min} nach unten beschränkt.

Gegenüber der Berechnung des Fehlers, die Braun beschreibt, fließt in den in dieser Arbeit verwendeten Resilient-Propagation-Algorithmus ein **Weight-Decay-Term** $10^{-\alpha} \sum_{r,s \in U} W(r,s)^2$ ein, der die Summe der quadrierten Gewichte im Netz mit in den Fehler einberechnet. Hohe Gewichte sollen vermieden werden, da sie die Generalisierungsfähigkeit eines KNN verringern. Weight-Decay ist von Werbos eingeführt worden [Wer88]. Resilient-Propagation arbeitet im so genannten Batch-Modus, in dem erst alle Änderungen der Gewichte zwischengespeichert werden und erst am Ende einer Epoche durch das Netz zurückpropagiert werden.

3.2.8 Diskussion: KNN

Künstliche neuronale Netze sind wie beschrieben in der Lage zu lernen, also einen Zusammenhang zwischen der Eingabe und Ausgabe herzustellen. Sie sind robust gegenüber verrauschten Daten. Einzelne Ausreißer in der Trainingsmenge haben keinen zu großen Einfluss auf die Qualität des Netzes, weshalb die KNN als fehlertolerant bezeichnet werden. Dagegen kann das Lernen auch zur Überanpassung an die Trainingsdaten führen, was die Generalisierungsfähigkeit stark verschlechtern kann. Zudem ist wegen des Black-Box-Prinzips das Erlernte nicht aus dem Netz ablesbar. Die Wahlfreitheit für die Parameter und die Topologie erschweren eine Bestimmung dieser, da es keine Vorgaben gibt, wie sie gewählt werden sollen und experimentell festgelegt werden müssen.

3.3 Support-Vector-Machines

In diesem Unterkapitel wird das Klassifikationsverfahren Support-Vector-Machines vorgestellt. Es werden die Grundlagen erklärt, die für das Verständnis dieses Klassifikationsverfahrens benötigt werden. Genauere Ausführungen sind in dem Buch von Scholkopf und Smola [SS01] nachzulesen.
Eine Support-Vector-Machine (SVM) in ihrer Grundform ist ein Klassifikationsverfahren, welches Muster in einem Hyperraum linear in zwei Klassen separieren kann und damit ein binärer Klassifizierer ist.

3.3.1 Linear separierbare Probleme

Es sind $l \in \mathbb{N}$ Paare gegeben aus Merkmalsvektor und zugehöriger Klasse $(\mathbf{x}_i, y_i),\ i = 1, \ldots, l$, $\mathbf{x}_i \in X,\ y_i \in \{-1, +1\}$. Die Muster sind linear trennbar. Wie im Abschnitt 3.1.1 bereits erwähnt, kann es viele verschiedene trennende Hyperebenen geben. Die optimale trennende Hyperebene ist genau die, bei welcher der Abstand der am nächsten gelegenen Datenpunkte der postiven bzw. negativen Klasse zur Hyperebenen am größten ist. Dieser Abstand wird mit $d_+ = d_- \in \mathbb{R}_{\geq 0}$ bezeichnet und die gesamte Breite der Hyperebene, die **Margin**, ist die Summe dieser beiden Abstände $d_+ + d_-$. Diese Margin soll nun so groß wie möglich werden, um eine gute Trennung der Klassen zu erreichen. Das Ermitteln dieser optimalen Hyperebene

beschreibt den Lernprozess der SVM.

Durch Multiplikation von $\mathbf{w}$ und θ mit der gleichen Konstanten $c \in \mathbb{R} \setminus 0$ wird die gleiche Hyperebene beschrieben [SS01]. Es gilt demnach $\{\mathbf{x}_i | \langle \mathbf{x}_i, \mathbf{w} \rangle + \theta = 0\} = \{\mathbf{x}_i | \langle \mathbf{x}_i, c \cdot \mathbf{w} \rangle + c \cdot \theta = 0\}$. Die Hyperebene lässt sich also nicht eindeutig beschreiben. Deshalb werden $\mathbf{w}$ und θ relativ zu den x_i skaliert, so dass die folgenden Nebenbedingungen gelten. Die so skalierte Hyperebene wird **kanonische Hyperebene** genannt.

$$\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta \geq 1, \quad \text{für } y_i = +1 \tag{3.1}$$

$$\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta \leq -1, \quad \text{für } y_i = -1. \tag{3.2}$$

In einer Nebenbedingung ausgedrückt lautet dies

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) - 1 \geq 0, \quad \forall i = 1, \ldots, l. \tag{3.3}$$

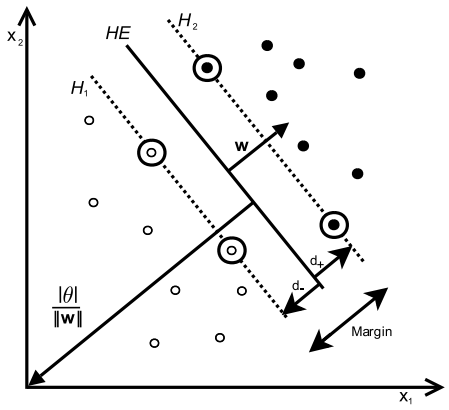

Abb. 3.11: *Optimale trennende Hyperebene (HE) mit maximalem Abstand zu den am nächsten liegenden Datenpunkten (Support-Vektoren, eingekreist)*

In Abbildung 3.11 ist eine solche optimale Hyperebene HE veranschaulicht. H_1 und H_2 sind parallele Verschiebungen der Hyperebene mit dem Abstand d_+ bzw. d_-. Für die Datenpunkte, die genau auf H_1 liegen, gilt $\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta = -1$. Der Abstand von H_1 zum Ursprung beträgt $\frac{|1+\theta|}{\|\mathbf{w}\|}$. Für die Datenpunkte, die genau auf H_2 liegen, gilt $\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta = 1$ und der Abstand von H_2 zum Ursprung beträgt $\frac{|-1+\theta|}{\|\mathbf{w}\|}$. Die Größe der Margin lässt sich nun durch die Differenz der Abstände der Hyperebenen H_1 und H_2 berechnen. Sie beträgt $\frac{|-1+\theta|}{\|\mathbf{w}\|} - \frac{|1+\theta|}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$, also $d_+ + d_- = \frac{1}{\|\mathbf{w}\|} + \frac{1}{\|\mathbf{w}\|}$. Das Ziel, die Margin zu maximieren, kann also dadurch erreicht werden, $\frac{2}{\|\mathbf{w}\|}$ zu maximieren bzw. $\|\mathbf{w}\|$ zu minimieren. Aufgrund der Monotonie lässt sich das Optimierungsproblem, welches es zu lösen gilt, auch folgendermaßen formulieren.

$$\min \ \tau(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$$

mit der Nebenbedingung 3.3.

Nun werden so genannte positive Lagrange-Variablen eingeführt, um die Nebenbedingung in die Optimierungsfunktion einfließen zu lassen (für weitere Ausführungen zu Lagrange-Variablen und Funktionen sei auf das Buch von Simmons [Sim72] verwiesen). Für jede Nebenbedingung 3.3 wird eine solche Lagrange-Variable $\alpha_i \in \mathbb{R}_{\geq 0}$ eingeführt, wobei $\alpha = (\alpha_1, \ldots, \alpha_l)$ ist. Diese Lagrange-Variablen müssen ebenfalls optimiert werden und dies führt zur Lagrange-Funktion

$$\begin{aligned} L(\mathbf{w}, \theta, \alpha) &= \tfrac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i (y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) - 1) \\ &= \tfrac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) + \sum_{i=1}^{l} \alpha_i. \end{aligned} \tag{3.4}$$

$L(\mathbf{w}, \theta, \alpha)$ muss nun bzgl. $\mathbf{w}$ und θ minimiert und bzgl. α_i maximiert werden. Dazu werden die partiellen Ableitungen nach θ und $\mathbf{w}$ gebildet und gleich 0 gesetzt werden. Es muss also

$$\frac{\partial}{\partial \theta} L(\mathbf{w}, \theta, \alpha) = 0$$

und

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, \theta, \alpha) = 0.$$

gelten. Dies führt zu

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

und

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i.$$

Durch Einsetzung in Gleichung 3.4 ergibt sich die duale Darstellung des Optimierungsproblems, welche bzgl. α maximiert werden muss [SS01]

$$\begin{aligned} L_D(\mathbf{w}, \theta, \alpha) &= \tfrac{1}{2} \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \sum_{j=1}^{l} \alpha_j y_j \mathbf{x}_j - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i (\sum_{j=1}^{l} \alpha_j y_j \mathbf{x}_j) + \theta) + \sum_{i=1}^{l} \alpha_i \\ &= \tfrac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &\quad -\theta \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i \\ &= \sum_{i=1}^{l} \alpha_i - \theta \sum_{i=1}^{l} \alpha_i y_i - \tfrac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \end{aligned} \tag{3.5}$$

mit den Nebenbedingungen

$$\alpha_i y_i \geq 0 \tag{3.6}$$

und

$$\sum_{i=1}^{l} \alpha_i y_i = 0. \tag{3.7}$$

Durch Einsetzen von 3.7 in 3.5 ergibt sich die veränderte Darstellung des dualen Optimierungsproblems

$$L_D(\mathbf{w}, \theta, \alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{3.8}$$

mit den Nebenbedingungen 3.6 und 3.7.
Wegen der Karush-Kuhn-Tucker-Bedingungen (KKT, notwendig für Optimalität einer Lösung), die in dem Buch von Scholkopf und Smola [SS01] näher erläutert werden, gilt

$$\alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) - 1) = 0, \quad \forall i = 1, \ldots, l. \tag{3.9}$$

Alle Trainingsdatenpunkte $\mathbf{x}_i$ mit $\alpha_i > 0$ liegen direkt auf einer der Hyperebenen H_1 oder H_2. Die $\mathbf{x}_i$ mit $\alpha_i = 0$ erfüllen die KKT-Bedingung 3.9 und liegen entfernt von H_1 bzw. H_2 und werden bei der weiteren Berechnung nicht weiter berücksichtigt, da nur die Datenpunkte auf den Hyperebenen H_1 und H_2 die Breite der Margin beeinflussen. Diese sind die *krititschen Elemente* der Trainingsdaten, die Support-Vektoren. Sie sind in Abbildung 3.11 umkreist dargestellt. Das Maximierungsproblem ist ein quadratisches Programm und α kann so optimal gelöst werden. Mit diesem optimalen α kann $\mathbf{w}$ berechnet werden mit

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i$$

mit $\mathbf{x}_i \in X_{SV}$ und $X_{SV} \subseteq X$ als Menge der Support-Vektoren. θ wird nun berechnet als Durchschnitt über alle θ der Support-Vektoren. Für alle $x_i \in X_{SV}, y_i \in \{-1, 1\}$ ergeben sich die einzelnen θ zu

$$\begin{aligned} & y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) - 1 &&= 0 \\ \Leftrightarrow \quad & \theta &&= \tfrac{1}{y_i} - \langle \mathbf{x}_i, \mathbf{w} \rangle \\ \Leftrightarrow \quad & \theta &&= y_i - \langle \mathbf{x}_i, \mathbf{w} \rangle . \end{aligned} \tag{3.10}$$

Das Ziel dieser Berechnungen ist die Bestimmung von $\mathbf{w}$ und θ, damit eine Entscheidungsfunktion aufgestellt werden kann, die jedes Muster in die zugehörige Klasse einteilt. Diese wird geschrieben als

$$f(\mathbf{x}) = sgn(\langle \mathbf{x}, \mathbf{w} \rangle + \theta), \quad \forall \mathbf{x} \in X. \tag{3.11}$$

sgn ist definiert für jedes $z \in \mathbb{R}$ als

$$sgn(z) = \begin{cases} 1 & \text{falls } z \geq 0 \\ -1 & \text{falls } z < 0 \end{cases}$$

3.3.2 Nicht separierbare Probleme

Wenn die Muster nicht linear separierbar sind, muss Gleichung 3.3 relaxiert werden. Dies führt zur **Soft-Margin-SVM** [SS01]. Dazu werden so genannte **Schlupfvariablen** $\zeta_i,\ i = 1, \ldots, l$ eingeführt. Damit lautet die Nebenbedingung nun

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) - 1 + \zeta_i \geq 0, \quad \forall i = 1, \ldots, l. \tag{3.12}$$

$$\zeta_i \geq 0, \quad \forall i = 1, \ldots, l. \tag{3.13}$$

Durch die Schlupfvariablen ist es möglich, Datenpunkte, die auf der *falschen* Seite der Hyperebene liegen, auf *richtige* Seite zu setzen. Dort werden diese Datenpunkte zu Support-Vektoren, siehe Abbildung 3.12.

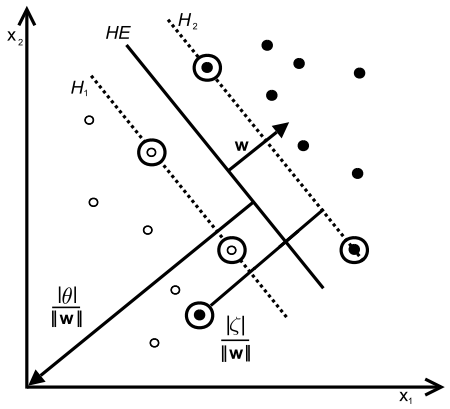

Abb. 3.12: *Optimal trennende Hyperebene im nicht-linear separierbaren Fall: Mithilfe der Schlupfvariablen ζ wird ein Datenpunkt, der auf der „falschen" Seite der optimal trennenden Hyperebene (HE) liegt, auf die zu HE parallel liegende Hyperebene H_2 projeziert und wird so zu einem Support-Vektor.*

Die Optimierungsfunktion lautet nun

$$\min\ \tau(\mathbf{w}, \zeta) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{l} \zeta_i,$$

$$C \in \mathbb{R}_{\geq 0},\ \zeta = (\zeta_1, \ldots, \zeta_l)$$

mit den Nebenbedingungen

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + \theta) \geq 1 - \zeta_i, \quad \forall i = 1, \ldots, l.$$

Die Summe der Schlupfvariablen gibt demnach einen Fehler an, welcher minimiert werden muss. C ist eine Konstante, die einen Trade-Off zwischen der Größe der Margin und der Minimierung des Fehlers bezeichnet. Sie stellt eine obere Grenze für die α_i dar. Dieser SVM-Typ wird C-SVM genannt.

3.3.3 Kernel

In den meisten Fällen sind die Daten nicht linear separierbar, wie beispielsweise in Abbildung 3.13 zu sehen. Eine Lösung für dieses Problem ist die Abbildung der Datenpunkte aus dem **Ursprungsmerkmalsraum** in einen gleich- oder höherdimensionierten **Feature-Space**. Wenn die Dimension groß genug ist, eventuell auch gegen ∞ geht, kann eine Hyperebene in diesen Feature-Space gelegt werden, welche eine Trennung der Daten vornimmt. Die lineare Trennung der Daten im Feature-Space ist somit äquivalent zur nicht-linearen Trennung im Ursprungsmerkmalsraum.
Eine solche Abbildung lautet

$$\phi : X^n \rightarrow H^m,$$

mit $\phi(\mathbf{x}) = \vec{x}$ und $n \leq m$. X^n ist der Ursprungsmerkmalsraum und H^m der Feature-Space.

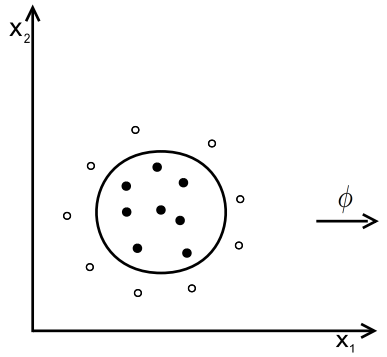

Abb. 3.13: *Datenpunkte im Ursprungsmerkmalsraum nicht linear separierbar*

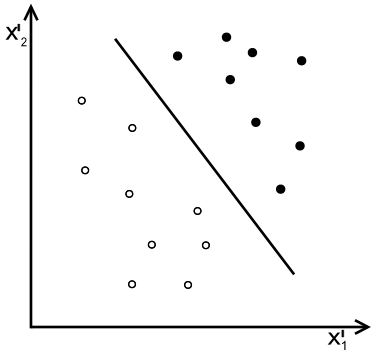

Abb. 3.14: *Datenpunkte im Feature-Space linear separierbar*

Die Entscheidungsfunktion aus 3.11 wird zu

$$f(\mathbf{x}) = sgn(\langle \phi(\mathbf{x}), \phi(\mathbf{w}) \rangle + \theta), \quad \forall \mathbf{x} \in X^n. \tag{3.14}$$

Die Berechnung von $\mathbf{w}$ und θ findet dann im Feature-Space statt. Das Abbilden in H und die Berechnung der Skalarprodukte im Feature-Space ist sehr zeit- und speicheraufwendig. Daher wird der so genannte **Kernel-Trick** angewendet [SS01]. Ein **Kernel** ist eine Funktion

$$K : \mathbb{R}^n \times \mathbb{R}^n \Rightarrow \mathbb{R}, \; K(\mathbf{x}, \mathbf{x}') = \left\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \right\rangle = \left\langle \vec{x}, \vec{x}' \right\rangle .$$

Dieser Kernel rechnet mit den Vektoren aus dem Ursprungsmerkmalsraum und berechnet nicht das Skalarprodukt der in den Feature-Space abgebildeten Vektoren. Da in der Lagrange-Funktion die Trainingsdaten als Skalarprodukte dargestellt sind, kann nun die Skalarproduktberechnung durch die Kernelfunktion ersetzt werden. Dadurch wird implizit in den Feature-Space abgebildet und das Skalarprodukt berechnet, ohne die eigentliche Abbildungsfunktion zu kennen. Mögliche Kernelfunktionen sind:

- **Linearer Kernel**: $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$
- **Polynomieller Kernel**: $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + b)^d$, mit $d \in \mathbb{N}, \; b \geq 0$

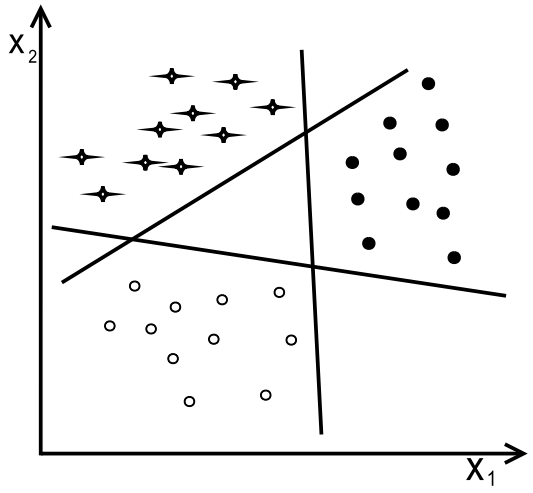

Abb. 3.15: *One-Versus-All-SVM: Trennung von mehr als 2 Klassen durch den Einsatz von mehreren Einzel-SVM.*

- **RBF-Kernel**: $K(\mathbf{x}, \mathbf{x}') = exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2})$, mit $\sigma \in \mathbb{R} \setminus 0$
- **Sigmoider Kernel**: $K(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \langle \mathbf{x}, \mathbf{x}' \rangle + \upsilon)$, mit $\kappa \in \mathbb{R}_{>0}, \upsilon \in \mathbb{R}_{<0}$

Für genauere Ausführungen zu Kernel sei auf das Buch von Scholkopf und Smola [SS01] verwiesen. Die Entscheidungsfunktion hat nun folgende Form

$$f(\mathbf{x}) = sgn(K(\mathbf{x}, \mathbf{w}) + \theta). \tag{3.15}$$

3.3.4 Mehrklassen-SVM

Die SVM sind binäre Klassifizierer, unterscheiden also nur zwischen zwei verschiedenen Klassen. In vielen Anwendungsfällen ist jedoch eine Trennung in mehr als zwei Klassen erforderlich. Im Folgenden wird die **One-Versus-All-Methode** vorgestellt.
Bei dieser Methode werden die Ergebnisse mehrerer binärer SVM kombiniert. Eine einzelne SVM trennt zwischen einer Klasse und dem Rest, so wie in Abbildung 3.15 zu sehen. Die Entscheidung, in welche der Klassen ein Muster fällt, wird anhand der Entscheidungsfunktion berechnet, bevor die sgn-Funktion angewendet wird. Die Einzel-SVM, die den höchsten Wert für eine Klasse besitzt, bestimmt somit die Klasse des Musters.

3.3.5 Diskussion: SVM

SVM sind sehr genaue Klassifizierer, da sie eine optimale Lösung für das gegebene Problem liefern. Sie sind robust bezüglich der Verrauschtheit der Trainingsdaten, da nur die Support-Vektoren betrachtet werden. Durch den Kernel-Trick können die SVM auch bei hochdimensionalen Probleme angewendet werden. Für eine gute Klassifikation ist der Parameter C entscheidend. Auch die Wahl des Kernels ist von großer Bedeutung, genauso wie dessen Parameter. Es gibt jedoch keine Verfahren zur Bestimmung des zu verwendenden Kernels und der Parameter. Diese müssen durch experimentelle Versuche bestimmt werden.

3.4 Zusammenfassung

Dieses Kapitel hat einen Überblick über die Mustererkennung gegeben. Zudem sind die Grundlagen der Klassifikationsverfahren künstliche neuronale Netze und Support-Vector-Machines ausführlich erklärt worden.

4 Verwandte Arbeiten

Wie schon in Kapitel 2 angesprochen, ist Poker ein Spiel mit unvollständiger Information, da nicht jeder Spieler die gleichen Informationen über den derzeitigen Spielzustand hat. Zudem ist beim Poker die Zufallskomponente enthalten, dass die Karten gemischt und zufällig gleichverteilt an die Spieler ausgegeben werden und auch die Gemeinschaftskarten im Vorhinein nicht bekannt sind, die während der Spielrunden aufgedeckt werden. Die Spieler müssen ihre Biet- und Spielstrategie demnach auf Annahmen über die gegnerischen Hole-Cards und die kommenden Gemeinschaftskarten aufbauen. Wegen dieser Eigenschaften ist Poker bereits häufig Gegenstand wissenschaftlicher Arbeiten gewesen.

Seit Beginn der Entwicklung von Computern wird versucht, Programme zu entwickeln, die selbstständig spielen. Diese sollen selbstständig eine Entscheidung treffen und eine bestimmte, den Regeln des Spiels entsprechende Aktion durchführen. Dazu baut das Programm einen so genannten Spielbaum [Sch08] auf, dessen innere Knoten jeweils einem Spielzustand entsprechen, in dem eine Entscheidungen getroffen wird. Die zu den nachfolgendenden Knoten führenden Kanten sind mit der jeweiligen Entscheidungsmöglichkeit belegt. In den Blättern des Spielbaums sind die erwarteten Auszahlungen eingetragen. Eine optimale Strategie verfolgt an jedem Entscheidungsknoten die Entscheidungsmöglichkeit, welche die höchste Auszahlung verspricht. Solche Spielbäume werden schnell sehr groß und erfordern einen sehr hohen Rechenaufwand.

In Spielen mit vollständiger Information, wie z. B. Schach, gibt es Suchheuristiken, die in auf diesen Spielbäumen arbeiten [Sch06]. Hierfür gibt es bereits viele gute Programme, die auch in der Lage sind, menschliche Gegner zu besiegen. Beispielsweise hat der Schach-Computer *Deep-Blue* von IBM, den damaligen Schachweltmeister Garry Kasparov 1997 besiegt [IBM97]. Derartige Suchmethoden können bei Spielen mit unvollständiger Information nicht angewendet werden [BDS02]. Koller und Pfeffer [KP97] stellen die spieletheoretische Herangehensweise an das Poker-Spiel vor. Es wird erklärt, dass wegen der fehlenden Informationen, die verschiedenen Spielzustände, also die inneren Knoten eines Spielbaums, nicht voneinander unterschieden werden können, da sie nach außen hin, also für alle anderen Spieler, die gleiche Information darstellen. Diese Knoten müssen nun durch *Information-Sets* ersetzt werden, in denen alle Zustände zusammengefasst werden, welche für den Spieler ununterscheidbar sind, wenn eine Entscheidung getroffen werden muss. Für diese Information-Sets stehen dem Spieler eine Menge Entscheidungsregeln zur Verfügung, die eine Wahrscheinlichkeitesverteilung über den zur Verfügung stehenden Möglichkeiten, den *randomized mixed strategies*, wiederspiegeln. Koller und Pfeffer legen besonders die Schwierigkeiten und hohe Komplexität des Pokerspiels dar. Billings et al. [BBD$^+$03] beschreiben, wie durch Reduktion der Anzahl der Knoten des Poker-Spielbaums von 10^{18} auf 10^7 für ein Zwei-Personen-Pokerspiel ohne zu großen Verlust an wichtigen Spieleigenschaften eine Approximation für eine spieletheoretisch optimale Spielstrategie für ein Pokerprogramm (auch **Poker-Bot**) entwickelt worden ist, mit welcher der Poker-Bot auch gegen menschliche Gegner gute Ergebnisse erzielt hat.

4.1 Poker-Bots

An der Universität von Alberta in Kanada gibt es die Forschungsgruppe „The University of Alberta Computer Poker Research Group (UACPRG)“, die sich besonders mit dem Thema Poker beschäftigt. Viele wissenschaftlichen Veröffentlichungen zum Thema Poker entstammen der Arbeit dieser Gruppe [UAC]. Billings et al. [BDS02] stellen die folgenden Merkmale heraus, die ein guter Pokerspieler benötigt und die daher auch für einen Poker-Bot implementiert werden sollen. Umgesetzt haben sie diese in ihrem Poker-Bot *Poki*.

- **Hand-Strength**: Sie stellt dar, wie stark eine Hand im Vergleich zu allen anderen Händen ist.
- **Hand-Potential**: Das Hand-Potential gibt die Wahrscheinlichkeit an, dass sich eine schlechte Hand noch verbessert, aber auch, dass sich eine starke Hand im Vergleich zu anderen Händen noch verschlechtert, wenn weiter Gemeinschaftskarten aufgedeckt werden.
- **Bluffing**: Bluffen ermöglicht es einem Spieler, auch mit einer schlechten Hand eine Spielrunde zu gewinnen, da die Gegner nicht wissen, wie stark die Hand wirklich ist.
- **Unpredicatability**: Unvorhersagbarkeit soll es dem Gegner schwierig machen, die eigene Strategie herauszufinden und seine eigene Strategie daran anzupassen.
- **Opponent modeling**: Die Gegnermodellierung wird im anschließenden Unterkapitel 4.2 vorgestellt.

4.2 Gegnermodellierung

Davidson [Dav02] beschreibt die Gegnermodellierung folgendermaßen. Die Gegnermodellierung ist in zwei Bereiche geteilt. Zum einen wird eine Gewichtsmatrix für die Hole-Cards des Gegners aufgestellt. In dieser ist für jede mögliche Zwei-Karten-Kombination, die ein Spieler halten kann, die bedingte Wahrscheinlichkeit abgetragen, dass ein Spieler diese Kartenkombination bei gegebenem Spielkontext hält. Diese Wahrscheinlichkeiten werden als Gewichte betrachtet und durch die Aktionen der Gegner beeinflusst und daher nach jeder Aktion eines Spielers angepasst. Die Gewichte besitzen Werte $g_{ij} \in [0,1] \subseteq \mathbb{R},\ i,j = 1,\ldots,13$. Die Gewichte werden normiert, so dass das höchste Gewicht in der Matrix den Wert 1 besitzt. Beispielsweise ist das Gewicht in Abbildung 4.1 für die Kartenkombination 7♡ 2♠ (die schlechteste mögliche Starthand) nach dem Flop geringer als für A♡ A♠ (die stärkste Starthand), da es wahrscheinlicher ist, dass ein Spieler die Hand 7♡ 2♠ bereits vor dem Flop foldet. In der Matrix wird zwischen 169 verschiedenen Händen unterschieden. Dies sind zum einen die Kombinationen, bei denen die Karten die gleiche Wertigkeit besitzen (beispielsweise 9♡ 9♣, 13 Kombinationen), die Karten unterschiedliche Wertigkeiten und Spielfarben besitzen (die Hand B♣ 8♢ beispielsweise wird mit B♠ 8♡ gleichgesetzt, 78 Kombinationen) und die Karten unterschiedliche Wertigkeiten, aber identische Spielfarbe besitzen (bei dieser besteht eine höhere Wahrscheinlichkeit auf einen Flush, 78 Kommbinationen). Wenn beide Hole-Cards die gleiche Spielfarbe besitzen, werden sie **suited**, ansonsten **offsuit** genannt. In der oberen Dreiecksmatrix in Abbildung 4.1 befinden sich die Gewichte für die suited, auf der unteren für die offsuited Hole-Cards. In der Diagonalen sind die Kombinationen abgetragen, die bereits

ein Paar bilden. Je dunkler ein Bereich der Matrix ist, desto höher ist das Gewicht für diese Kartenkombination. Das Beispiel zeigt die aktualisierten Gewichtsmatrizen nach einem Fold (links), einem Call (mittig) und einem Raise (rechts), nachdem die Preflop-Spielrunde gespielt ist.

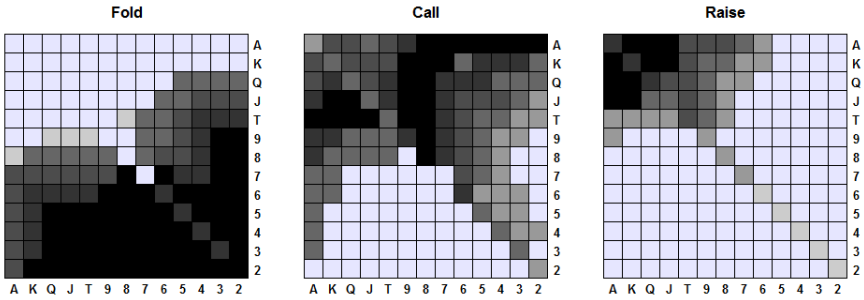

Abb. 4.1: *Gewichtsmatrix für Hole-Cards des Gegners: Je dunkler ein Bereich, desto höher ist das Gewicht für die Kartenkombination. Die obere Dreiecksmatrix stellt die suited, die untere die offsuit Kartenkombinationen dar. Die Diagonale zeigt die Kartenkombinationen mit identischer Wertigkeit. Die linke Abbildung zeigt die Gewichtsmatrix nach der Preflop-Spielrunde bei einem Fold, die mittige bei einem Call und die rechte bei einem Raise.*

Der zweite Teil der Gegnermodellierung ist die Aktionsvorhersage, also die Vorhersage welche Spielaktion Fold, Call/Check, Raise der Spieler mit welcher Wahrscheinlichkeit in der bestimmten Spielsituation als nächstes ausführen wird. Die Aktionen Call und Check werden als eine Aktion betrachtet. Diese Wahrscheinlichkeit wird in dem so genannten **Probability-Triple** [Dav02] ausgedrückt. Dieses Probability-Triple ist ein geordnetes Tupel mit Werten $PT = (f, c, r),\ f, c, r \in [0, 1] \subseteq \mathbb{R}$, mit $f + c + r = 1$. Dies drückt die Wahrscheinlichkeitsverteilung aus, mit der ein Spieler die Aktion Fold, Check/Call bzw. Raise ausführen wird. Dieses Probability-Triple wird bei Poki zur Aktualisierung der Gewichtsmatrix genutzt. Beispielhaft wird die Aktualisierung für die Hand B♠ 8♡ gezeigt. Das Gewicht vor der Aktualisierung beträgt 0.4. Das Probability-Triple ist $(0.1, 0.5, 0.4)$. Die Aktion, die ein Spieler nun tatsächlich ausgeführt hat, ist der Call. Somit berechnet sich der neue Eintrag in der Matrix zu $0.4 \times 0.5 = 0.2$. Bei einem Raise wäre das neue Gewicht $0.4 \times 0.4 = 0.16$. Für weitere Ausführungen sei auf die Arbeit von Davidson [Dav02] verwiesen.

Desweitern werden die Probability-Triple genutzt, um die Bietstrategie für Poki zu bestimmen, also welche Aktion Poki selbst durchführen soll. Wenn Poki eine Entscheidung treffen muss, simuliert er in Durchläufen, den **Trials**, die möglichen Entscheidungen bis an das Ende eines Spiels. Da der Spielzustand aufgrund der Unwissenheit über die Karten des Gegners nicht vollständig ist, werden für die Spieler wahrscheinliche Karten angenommen. Diese Auswahl der Karten beruht auf den Werten in der Gewichtsmatrix. Die Community-Cards werden zufällig gleichverteilt aus dem restlichen Deck gezogen. Wenn der modellierte Gegner nun an der Reihe ist, wird das Probability-Triple genutzt, welches in der Aktionsvorhersage erstellt worden ist. Dieses weist dem Spieler seine Aktion zu. So wird in vielen Durchläufen eine erwartete durchschnittliche Auszahlung errechnet. Die möglichen Entscheidungen für Poki, die simuliert werden, sind nur Check/Call und Raise, da beim Fold bereits klar ist, dass weder eine positive Auszahlung noch eine negative Auszahlung erreicht wird, da der Poki aus dem Spiel aussteigt. Die Aktion, welche die höchste durchschnittliche Auszahlung bringt, wird die von Poki zu

wählende Aktion. Dies ist die so genannte **Simulation-Based-Betting-Strategy** [BDS02], [Dav02].

Die Aktionsvorhersage ist das Kernstück der Gegnermodellierung. Darauf baut die eigene Strategiewahl auf. Die Vorhersage ist jedoch nicht einfach zu bestimmen. Sie hängt von vielen verschiedenen Faktoren ab. Ein Faktor ist die Unsicherheit über die Karten des Gegners und die kommenden Gemeinschaftskarten. Desweiteren weisen menschliche Spieler ihren Gegnern Aktionen basierend auf ihrer Intuition zu. Lernverfahren benötigen eine große Menge an Beobachtungen, um eine relativ gute Vorhersage machen zu können. Zudem ist es nicht ersichtlich, was alles für einen Spieler bei seiner Entscheidungsfindung wichtig ist. Der eine Spieler betrachtet beispielsweise die relative Position zu seinen Mitspielern als sehr wichtig, wobei ein anderer diesem Merkmal eine nicht so hohe Bedeutung zuordnet.

4.3 Methoden der Aktionsvorhersage

Die Aktionsvorhersage liefert eine Verteilung für die möglichen Aktionen Fold, Check/Call und Raise, durch das genannte Probability-Triple. Um diese zu erstellen, können verschiedenen Verfahren genutzt werden. Experten-Systeme sind eine Ansammlung von Regeln in der Form „*wenn ..., dann ...*“, die von Experten erstellt werden, wozu jedoch viel Wissen über die Poker-Domäne vorhanden sein muss. Dieses Methode wird **Generic-Opponent-Modeling** [Dav02] genannt. Damit wird den Gegnern jedoch unterstellt, dass diese sich an die gleichen Regeln halten. Das Generic-Opponent-Modeling ist ein statisches System, da es einmal erstellt wird und nicht dazulernen kann. Zudem fließen die tatsächlichen Aktionen der Spieler nicht mit in die Entscheidungsfindung ein.

Bei der Methode des **Specific-Opponent-Modeling** [Dav02] basiert die Aktionsauswahl des Gegners auf den vergangenen Aktionen des Gegners. Es ist ein statistisches Verfahren, welches die Bietfrequenz eines Spielers zu einer Spielsituation darstellt. Dazu wird zwischen zwölf Spielsituationen unterschieden. Jede Situation ist eine Kombination aus der Spielrunde (Preflop, Flop, Turn, River) und der Anzahl der Erhöhungen (keine, eine, zwei oder mehr), die bereits getätigt worden sind, wenn der Spieler an der Reihe ist. Es sind bedingte Wahrscheinlichkeiten in der Form $P(Raise|Turn \wedge OneBet)$, wobei in diesem Fall die bedingte Wahrscheinlichkeit für die Aktion Raise angegeben wird, wenn die aktuelle Spielrunde der Turn ist und bereits eine Erhöhung vor dem Gegner getätigt worden ist. Diese Werte werden im Verlauf des Spiels immer aktualisiert, um eine möglichst genaue Wahrscheinlichkeitsverteilung für die Aktion des Gegner zu bekommen.

Davidson, auch ein ehemaliger Mitarbeiter der UACPRG, hat ein künstliches neuronales Netz entwickelt, welches die Aktion eines Gegners vorhersagt [Dav99], [Dav02]. In der ersten Arbeit [Dav99] benutzt Davidson ein Multilayer-Perceptron mit 19 Eingabeneuronen, welche die aktuelle Spielsituation widerspiegeln, vier Neuronen auf einer versteckten Schicht und drei Ausgabeneuronen, die repräsentativ für die Aktionen Fold, Call/Check und Raise stehen. In der Arbeit von 2002 [Dav02] besteht seine Eingabe in veränderter Form aus nur 17 Neuronen. Er hat damit gute Ergebnisse erzielt.

4.4 Abgrenzung

Im Gegensatz zu den Arbeiten von Davidson wird in der vorliegenden Arbeit ein KNN mit anderer Topologie benutzt. Zudem sind die verwendeten Muster unterschiedlich, weshalb sich auch die Menge der Eingabeneuronen unterscheidet. Ein weiterer Unterschied besteht darin, dass zusätzlich zu einem Netz mit 3 Ausgabeneuronen, repräsentativ für Fold, Check/Call und Raise, ein weiteres Netz untersucht wird, in dem es 4 Ausgabeneuronen gibt. Diese unterscheiden Fold, Check, Call und Raise als vier unterschiedliche Aktionen. Somit ist eventuell eine bessere Abgrenzung der Aktionen Check und Call zum Fold bzw. Raise zu erreichen. Für die spätere Auswertung werden die Ergebnisse von Check und Call wieder zusammen betrachtet. In dieser Arbeit werden zudem weitere KNN benutzt, um die Klassifikation zu verfeinern. Neben KNN für alle Spielrunden zusammen werden noch solche betrachtet, die nur für die einzelnen Spielrunden trainiert werden. Desweiteren werden SVM genutzt, um die Klassifikation der Daten vorzunehmen.

Ein Vergleich der vorliegenden Arbeit mit denen von Davidson ist nicht möglich, da er weder die genaue Quelle seiner Datenbasis angibt, noch die genauen Spieler, die er testet. Außerdem enthält seine Datenbasis Informationen, die in der dieser Arbeit zugrunde liegenden Datenbasis nicht zur Verfügung stehen, wie die Höhe des Pots zu jedem Zeitpunkt und der genaue Einsatz eines Spielers bei jeder Aktion (vergleiche Unterkapitel 5.2). In seine Eingaben fließen desweiteren Vorhersagen von Expertensystemen ein, welche die UACPRG ebenfalls entwickelt hat. Diese Informationen sind nicht öffentlich zugänglich. Seine Ergebnisse liegen zudem nur auszugsweise vor. Eine Kontaktaufnahme zu Davidson ist erfolglos geblieben.

4.5 Zusammenfassung

In diesem Kapitel sind einige Arbeiten angesprochen worden, die sich mit dem Thema Poker befassen, und zeigen, warum dieses Spiel ein Thema für wissenschaftliche Arbeiten ist. Mögliche Methoden zur Vorhersage der Aktion des Spielers sind vorgestellt worden. Zudem sind Unterschiede zu bisherigen Arbeiten und neue Ansätze aufgezeigt worden.

5 Evaluierung

Dieses Kapitel beschreibt die Durchführung der Tests und die Auswertung der Ergebnisse. Die verwendete Software und die Datenbasis, auf der die Klassifikationsverfahren arbeiten, werden vorgestellt.

5.1 Verwendete Software

Für die Nutzung der künstlichen neuronalen Netze wird die Software **SNNS** (Stuttgart-Neural-Network-Simulator [SNN]) in der Version 4.3 verwendet. Diese Software ist an der Universität Stuttgart entwickelt worden, wird mittlerweile jedoch von der Universität Tübingen gewartet. Die daraus verwendeten Lernverfahren sind *BackpropMomentum* und *RProp*. Die Parametrisierung dieser Lernverfahren in SNNS wird in Unterkapitel 5.4 näher erläutert.

Die SVM werden mit der Software **RapidMiner** in der Version 4.2 Community-Edition erstellt und angewendet. RapidMiner wird an der Technischen Universität Dortmund entwickelt (ehemals YALE [rap], [MWK+06]). Das verwendete Modul für die SVM ist LIBSVM [CL01] von Chih-Chung Chang und Chih-Jen Lin.

Die Daten aus der Datenbasis werden in einer MySQL-Datenbank (Version 5.0.51) [mys] gespeichert. Für die Vorverarbeitung der Daten wird Java genutzt. Die Verwendete Java-Version ist die Version 6 mit der Aktualisierung 5 (Build 1.6.0_05-b13) [jav].

5.2 Daten

Die Klassifikationsverfahren benötigen eine Eingabe, die sie verarbeiten und auf der sie lernen können. Um diese Eingabe zu erhalten, müssen die Daten zunächst gesammelt, bereinigt und daraus Muster erzeugt werden. Diese müssen für die Klassifikationsverfahren verarbeitbar sein und in einer dementsprechenden Form vorliegen. Gespeichert werden die Daten in einer MySQL-Datenbank und die Muster werden mithilfe eines selbstgeschriebenen Java-Programms erstellt.

5.2.1 Poker-Datenbank

Bevor im Internet Online-Kasinos eröffnet haben, hat es bereits einen Internet-Relay-Chat (IRC) gegeben, in dem gepokert worden ist. Die hier verwendeten Poker-Daten stammen aus der „Michael Maurer's IRC Poker Database“ [pok]. Ein Programm, der IRC-Observer-Bot, hat die meisten spielrelevanten Daten protokolliert. Dazu gehören beispielsweise, welche Spieler an einem Spiel teilnehmen, in welcher Reihenfolge sie am Tisch sitzen, wie hoch die Anzahl der Chips ist, die ein Spieler am Tisch zur Verfügung hat, usw. Zu jedem einzelnen Spiel stehen zwei Dateien mit Informationen zur Verfügung. Die Informationen aus einer beispielhaften Hand-Informations-Datei sind in Tabelle 5.1 zu sehen.

Tab. 5.1: *Hand-Information*

Zeitpunkt	#Spieler	#Spieler/Pot-Größe				Gemeinschaftskarten
		P	F	T	R	
766303976	8	6/600	6/1200	6/1800	3/2400	3♠ J♣ Q♢ 5♣ A♡

In ihr sind die Informationen zu finden, welche Karten auf dem Board liegen und wie viele Spieler zu Beginn der Spielrunde teilnehmen. *#Spieler/Pot-Größe* gibt an, wie viele Spieler in der Spielrunde *P (Preflop)*, *F (Flop)*, *T (Turn)* und *R (River)* noch aktiv sind und welche Höhe der Pot zu Beginn der jeweiligen Spielrunde hat. Der *Zeitpunkt* ist ein Zeitstempel, der einzigartig ist und zur Unterscheidung der verschiedenen Spiele genutzt wird. Tabelle 5.2 stellt beispielhaft die Spieler-Informationen dar.

Tab. 5.2: *Spieler-Information*

Spieler	Zeitpunkt	Pos.	P	F	T	R	Chips	Inv. Ch.	Gewinn	Karten
Marzon	766303976	1	Bc	rc	kc	kf	12653	300	0	
Spiney	766303976	2	Bc	cc	kc	f	10237	300	0	
Doublebag	766303976	3	cc	r	r	rc	7842	500	0	J♡ Q♡
Neoncap	766303976	4	f	—	—	—	7857	0	0	
Maurer	766303976	5	f	—	—	—	12711	0	0	
Andrea	766303976	6	cc	c	c	f	7190	300	0	
Zorak	766303976	7	r	c	c	cc	4460	500	0	A♠ K♣
DBEKS	766303976	8	c	c	c	r	4304	500	2400	A♢ Q♠

Dort sind die Namen der teilnehmenden Spieler und der Zeitstempel zu sehen. Die Position am Tisch zeigt die Spalte *Pos.* an. Die Anzahl der Chips, die ein Spieler zu Beginn eines Spiels besitzt, sind an der Spalte *Chips* abzulesen. *Inv. Ch.* zeigt die Anzahl der Chips, die ein Spieler im gesamten Spiel investiert hat und *Gewinn* die jeweilige Ausschüttung des Pots an die Spieler. Wenn es zum Show-Down gekommen ist, werden die *Karten* der bis dahin aktiven Spieler angezeigt. In den Spalten *P (Preflop)*, *F (Flop)*, *T (Turn)* und *R (River)* ist die Aktionsabfolge der einzelnen Spieler zu sehen. *B* bedeutet, der Spieler muss einen Blind bringen, *f* bedeutet Fold, *k* bedeutet Check, *c* bedeutet Call und *r* bedeutet Raise. Das „—“ zeigt an, dass ein Spieler keine Aktion durchführt, da er bereits aus dem Spiel ausgeschieden ist. An diesen Aktionsequenzen kann der Ablauf eines Spiels nachvollzogen werden.

In dem hier betrachteten Fall müssen die Spieler Marzon und Spiney den Small-, respektive Big-Blind setzen. Doublebag callt, Neoncap und Maurer folden. Andrea callt auch, aber Zorak erhöht. DBEKS callt diese Erhöhung. Nun ist Marzon wieder an der Reihe und callt die Erhöhung von Zorak, ebenso wie Spiney, Doublebag und Andrea. Jetzt haben alle aktiven Spieler den gleichen Einsatz in den Pot einbezahlt und die Preflop-Spielrunde ist vorbei. Daraufhin wird der Flop aufgedeckt und Marzon ist als erster an der Reihe zu agieren. Er erhöht, Spiney callt und Doublebag erhöht erneut. Neoncap und Maurer sind bereits ausgestiegen und können daher keine weitere Aktion durchführen. Die restlichen aktiven Spieler callen alle die Erhöhung von Doublebag und es kommt der Turn, usw. Doublebag, Zorak und DBEKS verbleiben bis zum Show-Down. In Verbindung mit der Hand-Informations-Datei ist zu sehen,

dass DBEKS die beste Hand bildet mit dem Ass-Paar A♢ A♡, dem Damenpaar Q♠ Q♢ und dem Buben J♣. Diese Hand ist besser als die zwei Paare von Doublebag Q♡ Q♢ und J♡ J♣ mit dem Ass-Kicker A♡. Zorak besitzt nur das Ass-Paar A♠ A♡. Somit gewinnt DBEKS die 2400 Chips.

Insgesamt sind ungefähr zehn Millionen Pokerspiele im Zeitraum von 1995 bis 2001 protokolliert worden. Die Daten sind für viele verschiedene Pokervarianten vorhanden. Für Texas Hold'em gibt es verschiedene Pokerräume, die sich zum einen in der Höhe der Blinds und dem Vorhandensein eines Limits unterscheiden, zum anderen, ob nur menschliche Spieler oder auch Poker-Bots zugelassen sind. Die für diese Arbeit relevanten Daten stammen aus dem Pokerraum *No-Limit-Texas-Hold'em (h1-nobots)*, ohne Poker-Bots und ohne Limit, mit Small-Blinds in Höhe von 10 und Big-Blinds in Höhe von 20 Chips.

5.2.2 Datenprobleme

Nicht alle diese Pokerdaten sind verwertbar. In einigen Fällen haben Spieler mitten in einer Spielrunde das Spiel verlassen oder sind aufgrund technischer Probleme für eine einzelne Runde nicht anwesend gewesen. Zudem sind teilweise Aktionen protokolliert, die so jedoch regeltechnisch nicht durchführbar sind. Solche Daten sind nicht nutzbar und müssen zuvor ausgefiltert werden. Ein weiteres Problem sind fehlende Informationen. Da die gehaltenen Hole-Cards nur von den Spielern protokolliert sind, die bis in den Show-Down gekommen sind, fließt die Information über die gehaltenen Karten nicht in die Generierung der Muster ein. Ein besonders schwerwiegendes Problem ist, dass der Einsatz eines Spielers in einem Spiel nur über das gesamte Spiel verfügbar ist. Es ist aber nicht berechenbar, wie hoch der Einsatz bei jeder einzelnen Aktion gewesen ist, also welchen Betrag die einzelne Erhöhung und der Call hat. Für die Berechnung einiger Eingabeattribute sind diese jedoch notwendig, wie beispielsweise die Größe des Pots. Daher werden nur Muster für die jeweils erste Aktion eines Spielers zu Beginn einer Spielrunde generiert.

5.2.3 Generierung der Muster

Auf dieser Datenbasis werden nun die Muster generiert. Die Merkmale der Muster beschreiben eine Spielsituation und sind als reelle Zahlen in das Intervall $[0, 1] \subseteq \mathbb{R}$ skaliert. Dies ist notwendig, da sonst sehr hohe Werte eines Merkmals einen zu großen und sehr kleine Werte eines Merkmals einen zu geringen Einfluß auf den Gesamtfehler eines Klassifikationsverfahrens haben und somit die Gewichtsanpassung zu unterschiedlich ausfällt [DHS00]. Nun folgt die Beschreibung der einzelnen Merkmale, die jeweils Einfluss auf die Entscheidungsfindung eines Spielers haben. Dabei ist eine Auswahl von Merkmalen getroffen worden, die auf den Ausführungen in den Pokerbüchern [Ach07], [Skl04], [Skl99] und [Gre06] basiert. Die Merkmale, welche die aktuelle Spielsituation beschreiben, sind aufgeteilt in solche, die das Board beschreiben und solche, die Informationen über den Gegner sowie allgemeine Infomationen enthalten. Zunächst wird das Board beschrieben. Wichtige Karten für die Aktionsfindung sind besonders die Karten mit hoher Wertigkeit.

- **Stage**: Die jeweilige Spielrunde wird angegeben mit $0 \mathrel{\widehat{=}} \mathit{Flop}$, $0.5 \mathrel{\widehat{=}} \mathit{Turn}$ und $1 \mathrel{\widehat{=}} \mathit{River}$.
- **Queen**: Liegt eine Dame auf dem Board? $0 \mathrel{\widehat{=}} \mathit{false}$, also nein und $1 \mathrel{\widehat{=}} \mathit{true}$, also ja.

- **King**: Liegt ein König auf dem Board? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **Ace**: Liegt ein Ass auf dem Board? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **PairOnBoard**: Liegt bereits ein Paar auf dem Board? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **Colour**: Maximale Anzahl der Karten einer gleichen Spielfarbe auf dem Board.
- **Straight**: Ist eine Straight möglich mit den Karten auf dem Board? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.

Die allgemeinen Informationen sind im Folgenden beschrieben.

- **PlayersDealtCards**: An den Tischen des IRC-Poker-Servers sind maximal zehn Spieler erlaubt. Dieser Wert berechnet sich zu (Anzahl Spieler zu Beginn einer Spielrunde) $\cdot$ 0.1.
- **PCActive**: Gibt die Anzahl der noch aktiven Spieler am Tisch an und wird berechnet mit (Anzahl aktiver Spieler) $\cdot$ 0.1.
- **ChipleaderInGame**: Dieses Merkmal gibt an, ob der so genannte **Chipleader**, also der Spieler, der am Tisch die meisten Chips besitzt, einer der aktiven Spieler in der aktuellen Spielrunde ist. $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **Potsize**: Die Größe des Pots wird als Verhältnis der im Pot befindlichen Chips zu allen am Tisch verfügbaren Chips angegeben.
- **AggressivePlayerBehind**: Befinden sich hinter dem betrachteten Gegner noch aggressive Spieler, also solche, die in der Vergangenheit oft erhöht haben? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **NumberBets**: Die Anzahl der Erhöhungen, die vor dem Gegner getätigt worden sind, keine (0), eine (0.5), zwei oder mehr (1).

Die Merkmale, die den Gegner beschreiben, sind die folgenden.

- **Position**: Die relative Position des Gegners zu den anderen aktiven Spielern. 0.5 bedeutet beispielsweise, dass die Hälfte der anderen aktiven Spieler vor und die andere Hälfte hinter dem Gegner sitzen.
- **Bankroll**: Bankroll gibt die Anzahl der Chips des Gegners im Verhältnis zu der Anzahl der Chips des Chipleaders an.
- **Blind**: Muss der Gegner zu Beginn des Spiels den Big-Blind oder den Small-Blind setzen? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **BetLastRound**: Hat der Gegner in der Spielrunde zuvor erhöht? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **PlayerChipleader**: Ist der Gegner der aktuelle Chipleader am Tisch? $0 \mathrel{\widehat{=}} false$, $1 \mathrel{\widehat{=}} true$.
- **BankrollActiveChipleader**: Wie hoch ist der Anteil der Chips des Gegners an der Anzahl der Chips, die der aktive Spieler mit den meisten Chips am Tisch hat?
- **BankrollActiveLowstack**: Wie hoch ist der Anteil der Chips des aktiven Spielers mit der geringsten Anzahl der Chips an denen des Gegners?

- **Playfrequency**: Mit welcher Frequenz hat der Gegner sich den Flop in der Vergangenheit angeschaut?
- **Betfrequency**: Mit welcher Frequenz hat der Gegner in der Vergangenheit Erhöhungen angesetzt?

5.2.4 Ausgabe

In der Ausgabe ist die jeweilige Aktion des Gegners kodiert. Bei den verwendeten neuronalen Netzen steht (1 0 0) für Fold, (0 1 0) für Check/Call und (0 0 1) für Raise. Bei der Ausgabe, die zwischen den vier Aktionen unterscheidet, steht (1 0 0 0) für Fold, (0 1 0 0) für Check, (0 0 1 0) für Call und (0 0 0 1) für Raise. Die SVM kann mit nominalen Werten für die Ausgabe arbeiten, wobei der Name der Aktion repräsentativ für deren Klasse steht.

5.2.5 Aufteilung der Muster

Für jeden Spieler werden die Muster in zwei disjunkte Mengen aufgeteilt. 70% der Muster sind jeweils der Trainingsmenge zugeordnet und 30% der Testmenge. Damit die Klassifikation nicht zu sehr von der Auswahl der Muster abhängt, werden insgesamt elf verschiedene Trainings- und Testmengen erzeugt. Die erste Aufteilung ist chronologisch geordnet, also die ersten 70% der Muster sind in der Trainingsmenge, die letzten 30% in der Testmenge. Weitere zehn Test- und Trainingsmengen basieren auf einer zufälligen Verteilung im genannten Verhältnis. Die zufällige Verteilung beruht auf verschiedenen Seeds, so dass eine gute Verteilung der Muster auf die Mengen vorliegt. Die Mengen werden mit *sort.Pat.* für die chronologisch sortierten Mustermengen und *rand.Pat.0 bis rand.Pat.9* für die randomisierten Mustermengen bezeichnet. Aus diesen Mengen, die insgesamt alle Muster für einen Spieler enthalten, also für jede Spielrunde Flop, Turn und River, werden Untertrainings- und -testmengen erzeugt, die jeweils nur die Muster für die jeweilige Spielrunde enthalten. Die Preflop-Spielrunde wird komplett aus den Mustern herausgefiltert und nicht betrachtet. Die Spielweise in dieser Spielrunde unterscheidet sich zu stark von der in den übrigen Spielrunden, da viel weniger Informationen vorhanden sind. Davidson [Dav99] hat dadurch eine stark verbesserte Klassifikation erreicht. Da keine Annahmen über eine erwartete durchschnittliche Fehlerrate gemacht werden können, werden die Parameter für die Lernverfahren an einer Menge von Parameter-Testspielern ermittelt. Die ermittelten Parameter werden für die anschließenden Tests mit einer von den Parameter-Testspielern disjunkten Spielermenge getestet. Einen Überblick über die Spieler gibt der folgende Abschnitt.

5.2.6 Spieler

Für die Bestimmung der Parameter steht eine Parameter-Testmenge von sechs Spielern zur Verfügung, die zufällig gleichverteilt aus der Menge aller Spieler ausgewählt sind. Im Einzelnen sind dies die Spieler, die in Tabelle 5.3 dargestellt sind. *Sp.* gibt die Spielernummer, *Playfreq.* gibt die Frequenz eines Spielers an, in der er sich den Flop angeschaut hat. *Betfreq.* stellt dar, in welcher Frequenz der Spieler in den Spielrunden durchschnittlich erhöht hat. *Sp.-Typ* gibt den Spielertypen an. *lp* steht für loose-passive, *la* für loose-aggressive, *tp* für tight-passive und *ta* für tight-aggressive. Die Spielertypbestimmung basiert auf ihrer Playfrequency und ihrer Betfrequency, vergleiche Unterkapitel 2.3. *Nickname* gibt den Namen an, den der Spieler im IRC zum Pokern genutzt hat. *Anz. Spiele ges.* gibt an, an wie vielen Spielen der Spieler

insgesamt teilgenommen hat. *Anz. Muster ges.* gibt die Anzahl aller Muster an, die aus den Spielen generiert worden sind, in denen der Spieler sich den Flop, Turn oder River angeschaut hat. Für Spieler, die sich selten den Flop anschauen, also dessen Playfrequency niedrig ist, sind demnach weniger Muster als insgesamt gespielte Partien verfügbar. Im Gegensatz dazu sind bei Spielern, die sich häufiger den Flop anschauen, mehr Muster verfügbar.

Tab. 5.3: *Spieler der Parameter-Testmenge*

Sp.	Betfreq.	Playfreq.	Sp.-Typ	Anz. Spiele ges.	Anz. Muster ges.	Nickname
P0	0.46	0.55	la	200	246	beomar
P1	0.17	0.31	tp	853	535	fuzzyfish
P2	0.11	0.52	lp	721	776	OpusX
P3	0.43	0.45	la	1276	1330	TKChuck
P4	0.49	0.35	ta	1927	1458	TresEquis
P5	0.25	0.42	lp	1318	1146	wolfdog99

Die Spieler, die abschließend getestet und ausgewertet werden, sind in Tabelle 5.4 dargestellt.

Tab. 5.4: *Spieler der Testmenge*

Sp.	Betfreq.	Playfreq.	Sp.-Typ	Anz. Spiele ges.	Anz. Muster ges.	Nickname
0	0.46	0.62	la	1900	2592	Aries
1	0.22	0.62	lp	861	1279	Cardwiz2
2	0.24	0.63	lp	464	724	Cheyenne
3	0.14	0.40	tp	2018	1921	cyclops
4	0.32	0.51	lp	1318	1639	shavedleg
5	0.61	0.11	ta	5122	1414	Squidy
6	0.65	0.80	la	3037	5983	theking

5.3 Durchführung und Dokumentation der Tests

Die Durchführung und Dokumentation der Tests gliedert sich folgendermaßen. Zunächst werden für die KNN und die SVM die vorexperimentellen Planungen beschrieben. Diese befassen sich bei den KNN mit der Bestimmung der Topologie und den Parametern für die verwendeten Lernverfahren Backpropagation und Resilient-Propagation anhand der Spieler aus der Parameter-Testmenge. Nach der Festlegung der Topologie und der Parameter werden die abschließenden Tests durchgeführt. Dabei werden die Muster der Spieler der Test-Menge klassifiziert. Auch für die SVM werden in den Vorexperimenten zunächst die Kernel und ihre Parameter bestimmt und anschließend wird die Klassifikation der Muster durchgeführt. Im Anschluss werden die Ergebnisse der Klassifikation tabellarisch und graphisch aufbereitet. Daran schließen sich die Auswertungen und Vergleiche der Ergebnisse an. Abbildung 5.1 gibt einen Überblick über die durchgeführten Arbeitsschritte.

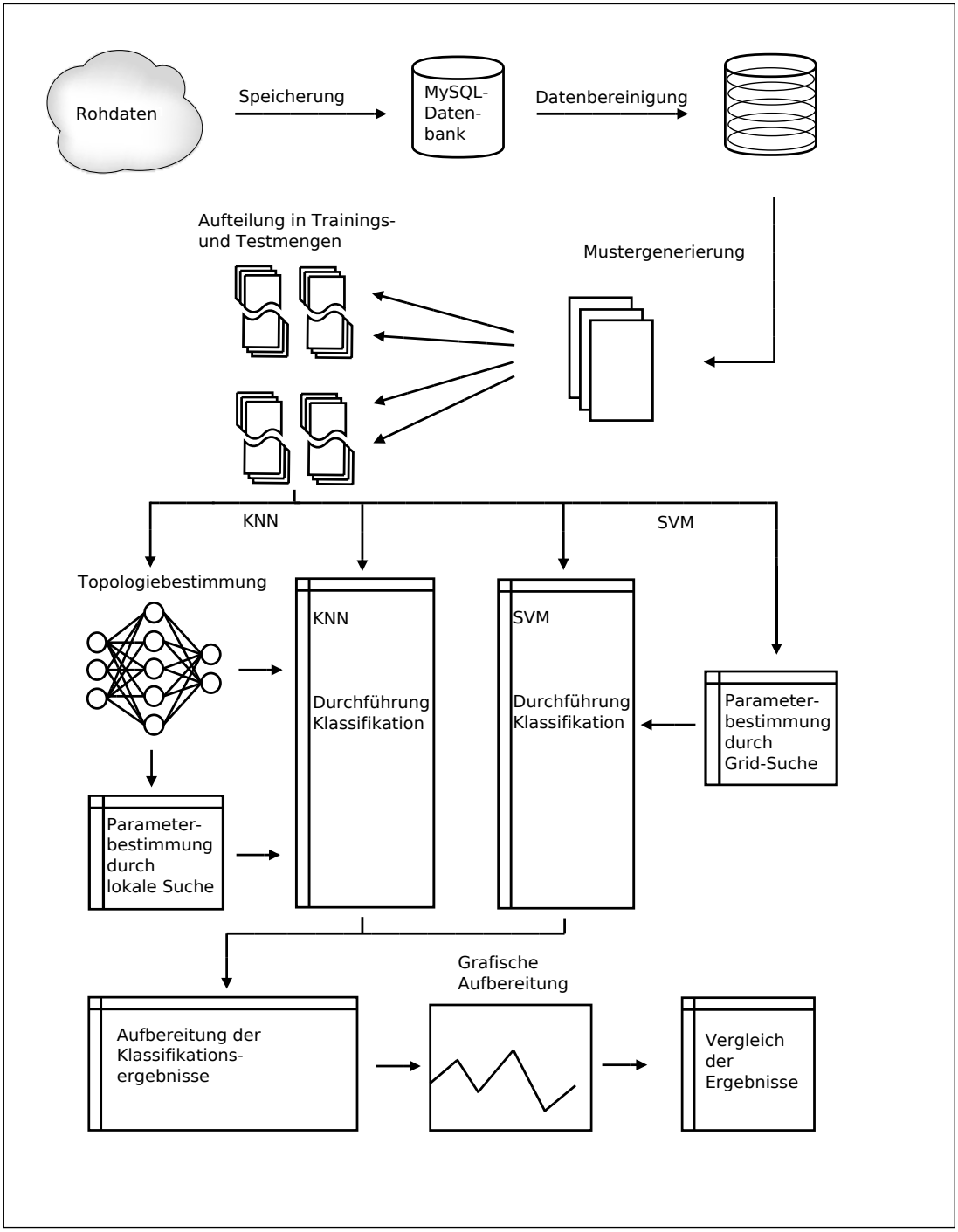

Abb. 5.1: *Übersicht über den Ablauf der durchgeführten Arbeitsschritte*

5.4 Durchführung: Künstliche neuronale Netze

In diesem Unterkapitel wird der Aufbau und die Durchführung der Tests der künstlichen neuronalen Netze beschrieben.

5.4.1 Vorexperimentelle Planung

Die vorexperimentelle Planung befasst sich bei den künstlichen neuronalen Netzen mit der Bestimmung der Topologie der Netze und der Bestimmung der Parameter für die Lernverfahren. Diese werden in den folgenden Unterabschnitten beschrieben.

5.4.1.1 Topologie

Die Bestimmung der Topologie erfolgt experimentell. Dazu werden KNN mit verschiedenen Topologien erzeugt und an den Parameter-Testspielern getestet. Die Anzahl der Eingabe- und Ausgabeneuronen sind aufgrund der Problembeschreibung auf 22 Eingabe- und 3 bzw. 4 Ausgabeneuronen festgelegt. Es werden Netze aufgebaut mit einer versteckten Schicht und 5, 15, 25, 35 und 45 versteckten Neuronen und solche mit zwei versteckten Schichten mit 5, 15, 25, 35 und 45 versteckten Neuronen.

Die Parameter werden für beide Lernverfahren auf die Standardwerte des SNNS festgelegt, die auf Erfahrungen der Entwickler des Programms aus anderen Versuchen beruhen und dienen somit als erste Referenz. Beim Backpropagation-Verfahren sind dies $\eta = 0.2$, $\beta = 0.5$, $\gamma = 0.1$, $d_{max} = 0.1$ und beim Resilient-Propagation-Verfahren $\eta^- = 0.5$, $\eta^+ = 1.2$, $\Delta(u,v)^{(0)} = 0.1$, $\Delta(u,v)_{max} = 50$ und $\alpha = 4$.

In stichprobenartigen Vortests zur Ermittlung einer sinnvollen Anzahl an Epochen (Tests im Intervall [50, 250] mit Schrittweite 50 und den Standardparametern) hat sich der Standardwert des SNNS von 100 als zu hoch erwiesen, weshalb der halbe Standardwert von 50 Epochen gewählt wird. Als Aktivierungsfunktion wird die logistische Aktivierungsfunktion und als Ausgabefunktion die Identität gewählt (vergleiche Abschnitt 3.2.3). Die Startgewichte der neuronalen Netze werden im Intervall [-0.5, 0.5] zufällig gleichverteilt festgelegt, wie Davidson diese auch in seinen Arbeiten [Dav99], [Dav02] gewählt hat.

Resilient-Propagation arbeit im Batch-Modus (vergleiche Unterabschnitt 3.2.7.2). Im Gegensatz agiert das Backpropagation-Verfahren in dem vom SNNS festgelegten Online-Modus, wobei nach jeder einzelnen Präsentation eines Trainingbeispiels eine Anpassung der Gewichte vorgenommen wird. In jeder Epoche werden dem Netz die einzelnen Trainingsbespiele in zufällig veränderter Reihenfolge präsentiert.

Getestet werden die Trainings- und Testmuster der Parameter-Testspieler, die alle Spielrunden enthalten. Jeder Test wird fünfmal durchgeführt und aus diesen Tests die mittlere Klassifikationsrate ermittelt. Durch die Mittelung wird der Abweichung der Klassifikationsrate durch die zufällige Initialisierung des Netzes entgegengewirkt. Als Fehlermaß wird der Mean-Squared-Error (vergleiche Abschnitt 3.2.4) verwendet und als Entscheidungskriterium für die Güte eines Netzes die Klassifikationsrate gewählt (vergleiche Abschnitt 3.1.2).

Abbildung 5.2 zeigt die mittleren Klassifikationsraten der verschiedenen Topologien. An der Abszisse sind die Topologien abgetragen, dabei bedeutet *5*, dass eine versteckte Schicht mit 5 Neuronen, und *5 5*, dass zwei versteckte Schichten mit jeweils 5 Neuronen verwendet werden. BP3 steht für Backpropagation mit 3 Ausgabeneuronen, BP4 für 4 Ausgabeneuronen. RE steht für Resilient-Propagation. Es ist zu sehen, dass die Anzahl der versteckten Schichten und die Anzahl der Neuronen auf den versteckten Schichten einen relativ geringen Einfluss auf die Klassifikationsrate haben, da die Differenz der Klassifikationsraten der einzelnen Topologien sich nicht sehr stark unterscheiden. Bei BP3 liegt die Abweichung zwischen der besten Klassifikationsrate bei der Topologie mit 25 Neuronen auf einer Schicht (0.648) und der schlechtesten Klassifikationsrate bei zwei Schichten mit jeweils 5 Neuronen (0.637) bei 0.011. Auch bei den anderen Verfahren ist die Differenz sehr gering (BP4: 0.014, RE3: 0.012, RE4: 0.013). Die KNN mit Resilient-Propagation erzielen bessere Ergebnisse als die KNN mit Backpropagation. Jedoch liegen dabei die Differenzen durchschnittlich nur bei 0.017. Die jeweils besten Klassifikationsaten der Verfahren sind bei einer versteckten Schicht und 25 Neuronen auf dieser Schicht zu erkennen. Diese Topologie wird für die Versuche mit den neuronalen Netzen im weiteren genutzt und ist auch Grundlage für die Bestimmung der Parameter für die Lernverfahren.

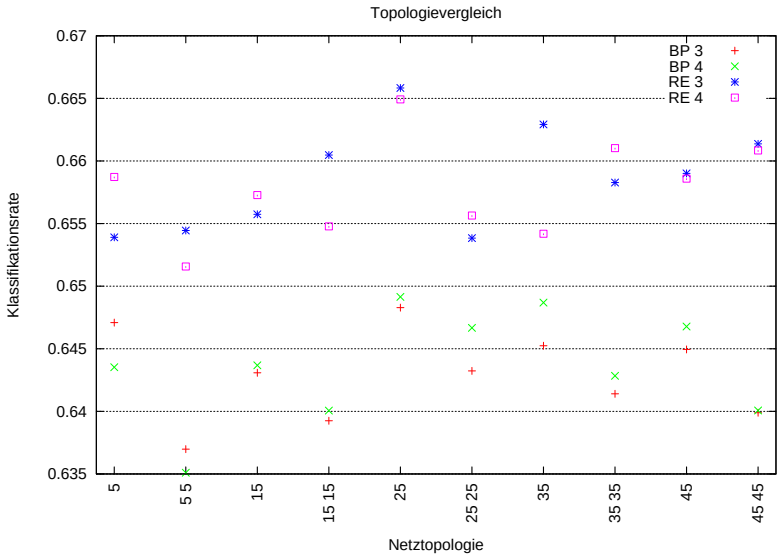

Abb. 5.2: *Vergleich der Klassifikationsraten bei unterschiedlicher Typologie der KNN: „5“ bedeutet 5 Neuronen auf einer versteckten Schicht, „5 5“ bedeutet jeweils 5 Neuronen auf zwei versteckten Schichten.*

5.4.1.2 Bestimmung der Parameter

Für das Backpropagation-Verfahren mit Momentum-Term gibt es beim SNNS (Modul *BackpropMomentum*) die folgenden Parameter, die eingestellt werden können (vergleiche Unterabschnitt 3.2.7.1).

- η: Wert der Lernrate.
- β: Wert des Momentum-Terms.
- γ: Wert des Flatspot-Elimination-Terms.
- d_{max}: Wert des maximal propagierten Fehlers.

Beim Resilient-Propagation-Verfahren (Modul *RProp* im SNNS) wird nur der Wert von α angepasst. Die Lernraten $\eta^- = 0.5$, $\eta^+ = 1.2$ und $\Delta_{min} = 0.000001$ sind durch das SNNS festgelegt und entsprechen den von Braun [Bra97] empfohlenen Werten. Da die Werte von $\Delta^{(0)}_{(u,v)}$ und Δ_{max} laut Braun nicht kritisch sind, werden sie auf die Standardwerte von $\Delta^{(0)}_{(u,v)} = 0.1$ und $\Delta_{max} = 50$ eingestellt. Die Parameter sind in Unterabschnitt 3.2.7.2 erläutert worden.

Für die Bestimmung der Parameter wird ein Lokale-Suche-Algorithmus eingesetzt. Dieser sucht ausgehend von einer Initiallösung in deren Nachbarschaft nach weiteren Lösungen. Wenn eine Parameterbelegung gefunden wird, die eine Verbesserung zur vorherigen Lösung darstellt, wird ausgehend von dieser Lösung in deren Nachbarschaft weitergesucht. Zur Bewertung der Lösung wird die Klassifikationsrate $f(\mathbf{p}) = CR,\ CR \in [0,1] \subseteq \mathbb{R}$ betrachtet. Sie stellt die so genannte Fitness dar. Dabei ist $\mathbf{p} = (\eta, \beta, \gamma, d_{max}) \in \mathbb{R}^4,\ \eta \in \mathbb{R}_{>0},\ \beta, \gamma, d_{max} \in \mathbb{R}_{\geq 0}$ für das Backpropagation-Lernverfahren und $\mathbf{p} = (\alpha),\ \alpha \in \mathbb{R}_{\geq 0}$ für das Resilient-Propagation-Verfahren ein Parametervektor. Die Fitness gilt es zu maximieren: $\max_{\mathbf{p}} f(\mathbf{p})$.

Der Pseudo-Code des Lokale-Suche-Algorithmus ist in Algorithmus 1 zu sehen. Dabei werden die initialen Werte für die einzelnen Parameter auf die erwähnten Standardwerte festgesetzt (Zeile 1.3). Die Klassifikationsrate, also die Fitness dieser Belegung, wird errechnet (Zeile 1.4) und dient als erste Referenz für die weiteren Parameterbestimmungen. Die Berechnung der Fitness erfolgt, wie in Algorithmus 2 zu sehen. Dabei wird zunächst ein KNN aufgebaut mit der vorgegebenen Topologie, der Aktivierungs- und Ausgabefunktionen und den Parametern $\mathbf{p}$=$p[\,]$ (Zeile 2.2). Das KNN wird mit den Trainingsmustern angelernt, bis die maximale Anzahl an Epochen erreicht wird (Zeile 2.3). Die Muster der Testmenge werden an dem angelernten KNN getestet und die Anzahl der korrekt klassifizierten Muster ermittelt (Zeile 2.4). Das Verhältnis der korrekt klassifizierten Muster zur Größe der Testmenge ergibt die Klassifikationsrate und damit die Fitness (Zeile 2.5). Solange eine maximale Anzahl an Iterationen noch nicht erreicht ist (Zeile 1.5), werden im nächsten Schritt die neuen Parameter so bestimmt, dass sie ähnlich zu den bisher besten Parametern sind, also in ihrer Nachbarschaft liegen (Zeile 1.7). Dazu werden beim Backpropagation-Verfahren mit Momentum-Term zwei der Parameter verändert, um einen nicht zu ähnlichen, aber auch nicht zu unähnlichen Nachbarn zu generieren. Beim Resilient-Propagation-Algorithmus wird nur der Wert von α angepasst. Die Nachbarschaftsfunktion für den Parametervektor des Backpropagation-Verfahrens lautet folgendermaßen.

$neighbour_{BP} : \mathbb{R}^4 \rightarrow \mathbb{R}^4$,
$neighbour_{BP}(\mathbf{p}) = neighbour_{BP}(\eta, \beta, \gamma, d_{max}) = (\eta + h \cdot factor \cdot s_\eta, \beta + h \cdot factor \cdot s_\eta, \gamma + h \cdot factor \cdot s_\gamma, d_{max} + h \cdot factor \cdot s_{d_{max}})$,
mit der Anpassungsrichtung $h \in \{-1, 1\}$, dem Schrittweitenfaktor $factor \in \{1, 2, 3\}$ und den Schrittweiten $s_\eta = 0.2 \cdot \eta_0$ (η_0 ist der Initialwert von η), $s_\beta = 0.2 \cdot \beta_0$, $s_\gamma = 0.2 \cdot \gamma_0$, $s_{d_{max}} = 0.2 \cdot d_{max_0}$.

Für das Resilient-Propagation-Verfahren lautet die Nachbarschaftsfunktion wie folgt.
$neighbour_{RE} : \mathbb{R} \rightarrow \mathbb{R}$,
$neighbour_{RE}(\mathbf{p}) = neighbour_{RE}(\alpha) = (\alpha + h \cdot factor \cdot s_\alpha)$, mit $s_\alpha = 0.2 \cdot \alpha_0$.

Der Pseudocode der Nachbargenerierung für den Parametervektor des Backpropagation-Verfahrens ist in Algorithmus 3 zu sehen. Ein Nachbar wird generiert, indem zunächst zufällig gleichverteilt ein Parameter ausgewählt wird (Zeile 3.4). Die Richtung der Anpassung, wird zufällig gleichverteilt bestimmt (Zeile 3.5). Der Betrag der Änderung hängt zum einen davon ab, wie groß die Schrittweite für den jeweiligen Parameter festgelegt ist. Diese beträgt 20% des Initialwertes. Zum anderen beeinflusst der derzeitige Iterationsschritt den Betrag der Veränderung eines Parameters. Der Algorithmus terminiert nach einer maximalen Anzahl an Iterationsschritten, die auf 100 festgelegt ist. Diese 100 Iterationsschritte bieten dem Algorithmus die Möglichkeit, einen ausreichend großen Bereich der Nachbarschaft zu erkunden. Beim ersten Drittel dieser Iterationsschritte wird die Schrittweite der Parameter mit dem Faktor 3 multipliziert, im zweiten Drittel mit dem Faktor 2 und im letzten mit dem Faktor 1. Die Berechnung des Schrittweitenfaktors findet in Zeile 1.6 statt. Das führt dazu dass zu Beginn noch größere Schrittweiten möglich sind und so die Möglichkeit gegeben ist, dass der Algorithmus lokale Minima und Plateaus überspringen kann. In einer späteren Phase der Laufzeit des Algorithmus wird dementsprechend in einer engeren Nachbarschaft gesucht. Der neue Parameterwert errechnet sich dann, wie in Zeile 3.6 zu sehen. Für die neu ermittelten Parameter wird wiederum die Klassfikationsrate ermittelt (Zeile 1.8) und mit der bisher besten Klassifikationsrate verglichen (Zeile 1.9). Wenn die Fitness sich nicht verschlechtert hat, wird die neu ermittelte Parameterbelegung als neue Referenz genutzt (Zeilen 1.10 und 1.11).

Algorithmus 1 Bestimmung der optimalen Parameter für die Lernfunktion

```
1:  function findOptimalParameters
2:      i = 0
3:      p_i[ ] = generateInitialSolution                ▷ Initiale Parameterbelegung
4:      f_i = calcFitness(p_i[ ])                       ▷ Berechnung der Fitness
5:      while i ≤ maxIteration do                       ▷ Abbruchkrit. max. Anzahl Iterationen
6:          factor = calcFactor(i, maxIteration)        ▷ Berechnung des Schrittweitenfaktors
7:          p_{i+1}[ ] = generateNeighbour(p_i[ ], factor, stepSize[ ])▷ Berechnung d. Nachb.
8:          f_{i+1} = calcFitness(p_{i+1}[ ])           ▷ Berechnung der Fitness des Nachbarn
9:          if f_{i+1} ≤ f_i then                       ▷ Bestimmung der z. Z. besten Parameter
10:             f_{i+1} = f_i
11:             p_{i+1}[ ] = p_i[ ]
12:         end if
13:         i++
14:     end while
15:     return p_i[ ]
16: end function
```

Der Algorithmus wird fünfmal ausgeführt auf der Menge der sechs Parameter-Testspieler. Es werden zunächst nur diejenigen Muster gewählt, die auf der chronologischen Aufteilung der Trainings- und Testmuster basieren (vergleiche Abschnitt 5.2.5). Für jeden Durchlauf wird eine Parameterkombination ermittelt. Von jeder der Parameterkombinationen der einzelnen

Algorithmus 2 Berechnung der Fitness einer Parameterbelegung

```
1: function CalcFitness(p[ ])
2:     KNN = createKNN(p[ ],...)
3:     trainedKNN = trainKNN(KNN, TrainingsPattern[ ], maxEpoch)
4:     correctClassified = testKNN(trainedKNN, TestPatttern[ ])
5:     f = correctClassified/size(TestPatttern[ ])
6:     return f
7: end function
```

Algorithmus 3 Generierung eines Nachbarn

```
1: function generateNeighbour(p[ ], factor,stepSize[ ])
2:     j = 0
3:     while j ≤ 1 do
4:         p[n] = randSelectParameter(p[ ])
5:         h = rand(-1,1)
6:         p[n] = p[n] + h · factor · stepSize[n]
7:         j++
8:     end while
9:     return p[ ]
10: end function
```

Spieler wird der Median jedes einzelnen Parameters ermittelt. Diese Werte ergeben die Parameterbelegung für die abschließenden Tests. Die Ergebnisse sind in den Abbildungen 5.3 und 5.4 mithilfe von Boxplots dargestellt. Dabei steht *eta 3* für den Parameter η, der bei 3 Ausgabeneuronen ermittelt worden ist. Es sind nur die Parameter abgebildet, die bei der Bestimmung des Nachbarn auch verändert worden sind.

Ein Boxplot besteht aus dem Median, zwei so genannten Quartilen und den beiden Extremwerten. Die Extremwerte begrenzen den Boxplot und geben jeweils den höchsten und den niedrigsten Wert an, der ermittelt worden ist. Der Median stellt den Wert dar, der die Menge der gemessenen Werte in zwei Hälften teilt, also liegen 50% der gemessenen Werte über dem Median und 50% darunter. Das obere und untere Ende der Box stellen das erste und dritte Quartil dar. Die Quartile trennen jeweils die oberen bzw. unteren 25% der Werte ab. Sie umfassen daher insgesamt 50% der ermittelten Werte.

Es ist zu sehen, dass die Parameter teilweise sehr stark abweichen (*eta 3*, *beta 3*, *eta 4*, *beta 4*, *alpha 3*, *alpha 4*). Es können also durch sehr unterschiedliche Parameterkombinationen beste Klassifikationsraten erreicht werden. Daher wird auf die Tests mit den randomisierten Trainings- und Testmustermengen bei der Parameterbestimmung verzichtet und für die abschließenden Test werden die Werte für die Parameter nun auf den Median der einzelnen Parameter dieser Tests festgelegt. Sie ergeben sich wie in den Tabellen 5.5 und 5.6 dargestellt. Sie unterscheiden sich nur unwesentlich von den Standardparametern, die vom SNNS vorgegeben sind.

Um die optimale Anzahl an Epochen zu bestimmen, wird der Fehlerverlauf der Testläufe untersucht. Für jeden Spieler ist die lokale Suche fünfmal durchgeführt worden. Aus diesen

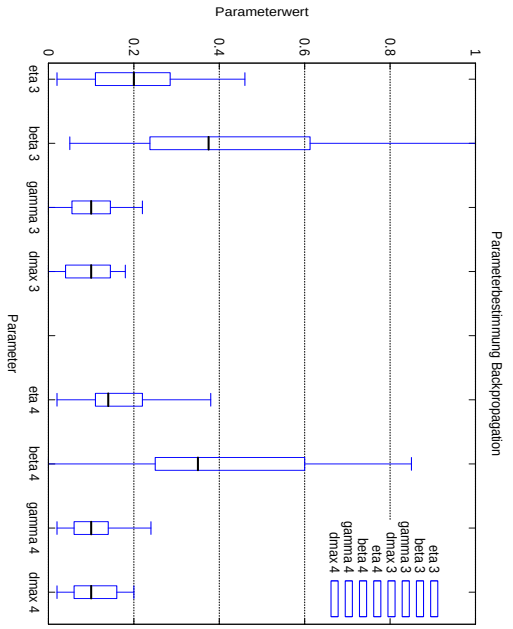

Abb. 5.3: *Parameterbestimmung des Backpropagation-Verfahrens: Für die veränderbaren Parameter des Backpropagation-Verfahrens sind die Boxplots der durch den Lokale-Suche-Algorithmus ermittelten Parameter dargestellt, jeweils für 3 und 4 Ausgabeneuronen.*

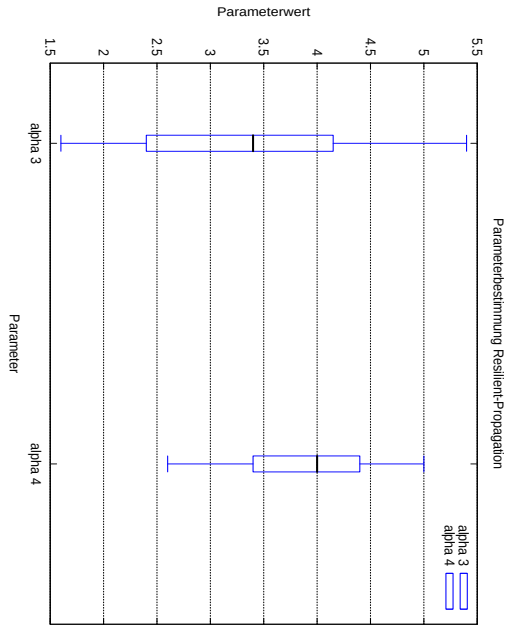

Abb. 5.4: *Parameterbestimmung des Resilient-Propagation-Verfahrens: Für den veränderbaren Parameter des Resilient-Propagation-Verfahrens sind die Boxplots der durch den Lokale-Suche-Algorithmus ermittelten Parameter dargestellt, jeweils für 3 und 4 Ausgabeneuronen.*

Tab. 5.5: *Ermittelte Parameter des Backpropagation-Verfahrens für 3 und 4 Ausgabeneuronen im Vergleich zu den Standardparametern des SNNS*

A.	η	β	γ	d_{max}
3	0.2	0.37	0.1	0.1
4	0.14	0.35	0.1	0.1
Std. Param.	0.2	0.5	0.1	0.1

Tab. 5.6: *Parameter des Resilient-Propagation-Verfahrens für 3 und 4 Ausgabeneuronen im Vergleich zu den Standardparametern des SNNS*

A.	α	η^-	η^+	$\Delta^{(0)}_{(u,v)}$	Δ_{max}
3	3.4	0.5	1.2	0.1	50
4	4	0.5	1.2	0.1	50
Std. Param.	4	0.5	1.2	0.1	50

Testläufen wird der Fehlerverlauf der jeweils besten Parameterkombination gewählt. Über alle diese Fehlerverläufe der einzelnen Spieler wird nun der Durchschnitt des Fehlers in den einzelnen Epochen ermittelt. Der Verlauf des Klassifikationsfehlers der Trainings- und Testmengen ist in den Abbildungen 5.5 bis 5.8 zu sehen. An der Abszisse sind die Epochen abgetragen und an der Ordinate der entsprechende Mean-Squared-Error. Es ist zu erkennen, dass die Fehlerrate der Trainingsmenge kontinuierlich abnimmt, je höher die Anzahl der Epochen ist. Dieser Verlauf ist zu erwarten, da sich das neuronale Netz immer mehr den Trainingsbeispielen anpasst und daher die Trainingsmenge immer präziser klassifiziert wird. Der Fehlerverlauf der Testbeispiele ist auch typisch für neuronale Netze. Zunächst nimmt er mit jeder Epoche ab, bis er ein Minimum erreicht, wonach er dann mit jeder Epoche wieder zunimmt. Ab diesem Zeitpunkt findet die Überanpassung des neuronalen Netzes an die Trainingsdaten statt, und es verliert an Generalisierungsfähigkeit, wie in Unterkapitel 3.1.3 beschrieben. Deshalb wird der Lernvorgang im Minimum abgebrochen. Das Backpropagation-Verfahren erreicht das Minimum des Mean-Squared-Errors der Testmenge im Bereich um 20 Epochen, wogegen das Resilient-Propagation-Verfahren bereits nach ca. 15 Epochen das Minimum erreicht. Daher wird für die abschließenden Tests die Anzahl der Epochen auf 20 für das Backpropagation- und 15 für das Resilient-Propagation-Verfahren festgelegt.

5.4.2 Versuchsaufbau

Für jeden der Testspieler werden jeweils mehrere KNN aufgebaut. Ein KNN lernt an den Mustern, die alle Spielrunden enthalten. Für die einzelnen Spielrunden Flop, Turn und River werden jeweils spielrunden-spezifische KNN aufgebaut, die nur an den Mustern lernen, welche die jeweilige Spielrunde enthalten. Diese vier Netze werden sowohl für 3 als auch 4 Ausgabeneuronen und für die beiden Lernverfahren erstellt. Insgesamt ergeben sich 16 KNN pro Spieler. Die einzelnen KNN sind aufgebaut aus 22 Eingabeneuronen, einer versteckten Schicht mit 25 Neuronen und 3 bzw. 4 Ausgabeneuronen. Die Topologien der Netze beruhen auf der Problembeschreibung und den vorexperimentellen Untersuchungen aus Unterabschnitt 5.4.1. Darauf beruhen auch die Parametereinstellungen für die Lernverfahren. Diese sind in den Tabellen 5.5 und 5.6 zu sehen. Als Aktivierungsfunktion wird für alle Neuronen die logistische Funktion und als Ausgabefunktion die Identität gewählt. Die Anfangsbelegungen der Gewichte werden zufällig gleichverteilt aus dem Intervall [-0.5, 0.5] gewählt. Die Anzahl der Epochen wird auf die in Unterabschnitt 5.4.1.2 bestimmten 20 für Backpropagation und 15 für Resilient-Propagation festgelegt, nachdem das Training stoppt und mit der Testmenge getestet wird.

Jeder Versuch wird fünfmal durchgeführt, um eventuellen Schwankungen durch die zufällig gewählten Anfangsbelegungen der Gewichte entgegenzuwirken. Alle Tests werden auf den in Unterkapitel 5.2.5 beschriebenen elf Trainings- und Testmengen durchgeführt. Als Software kommt der SNNS zum Einsatz.

5.5 Durchführung: Support-Vector-Machines

Dieses Unterkapitel beschreibt den Aufbau und die Durchführung der Tests, die mit den Support-Vector-Machines umgesetzt werden.

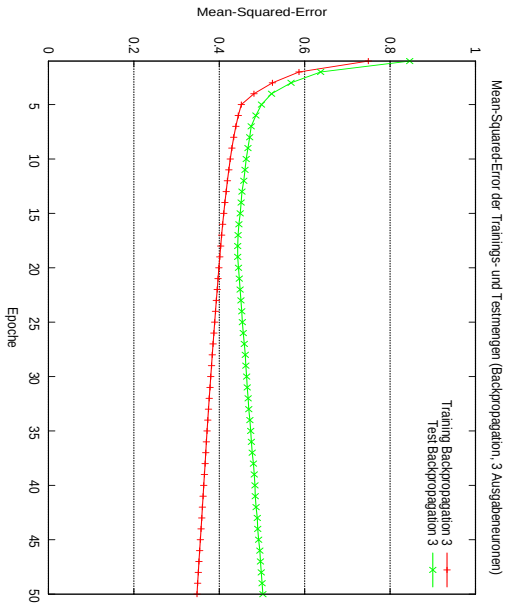

Abb. 5.5: *Fehlerverlauf der Trainings- und Testmuster gemittelt über alle Spieler der Parameter-Testmenge (Backpropagation und 3 Ausgabeneuronen)*

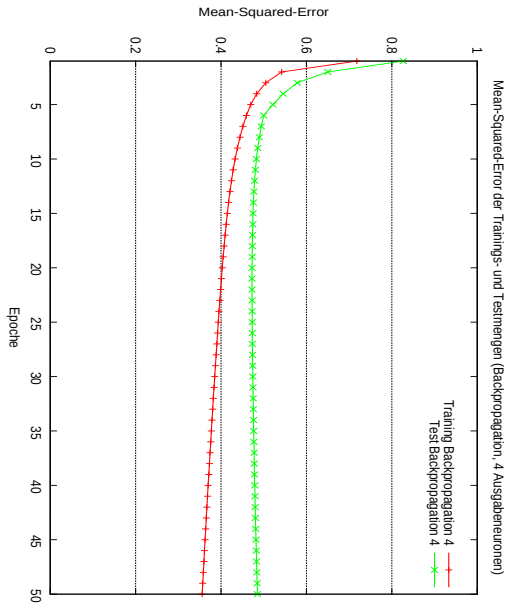

Abb. 5.6: *Fehlerverlauf der Trainings- und Testmuster gemittelt über alle Spieler der Parameter-Testmenge (Backpropagation und 4 Ausgabeneuronen)*

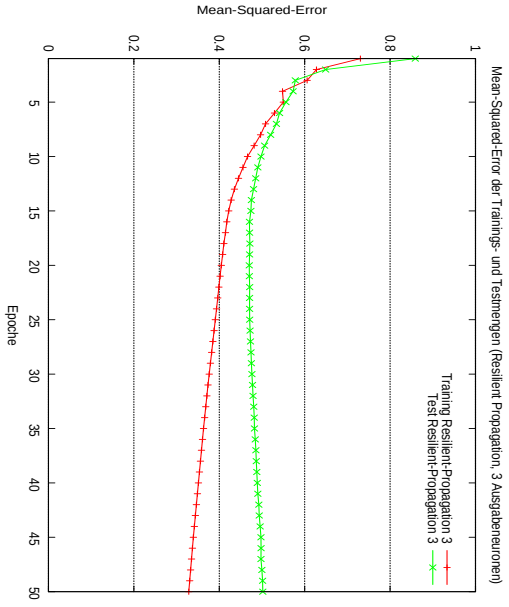

Abb. 5.7: *Fehlerverlauf der Trainings- und Testmuster gemittelt über alle Spieler der Parameter-Testmenge (Resilient-Propagation und 3 Ausgabeneuronen)*

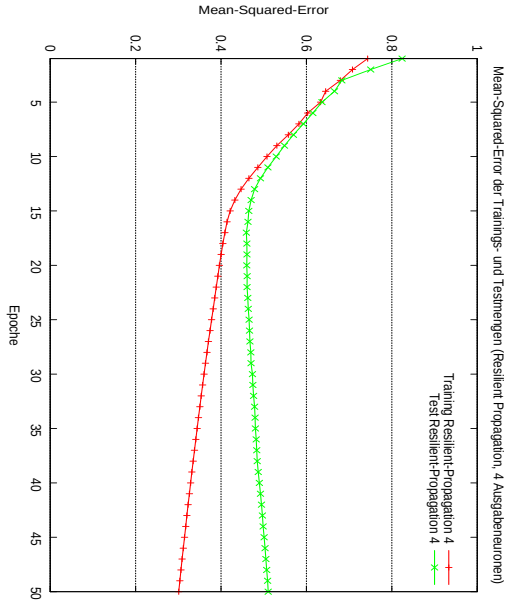

Abb. 5.8: *Fehlerverlauf der Trainings- und Testmuster gemittelt über alle Spieler der Parameter-Testmenge (Resilient-Propagation und 4 Ausgabeneuronen)*

5.5.1 Vorexperimentelle Planung

Die vorexperimentelle Planung befasst sich bei den Support-Vector-Machines mit der Bestimmung des Kernels und der Parameter, die für die Kernel benötigt werden.

5.5.1.1 Parameterbestimmung

Bei der SVM müssen im Gegensatz zu den künstlichen neuronalen Netzen nicht so viele Parameter eingestellt werden. In stichprobenartigen Vortests haben sich als Kernelfunktion der lineare Kernel und der polynomielle Kernel als vielversprechend herausgestellt, weshalb diese beiden Kernel getestet werden. Für die Ermittlung der Parameter werden von den Parameter-Testspieler die Beispielmuster genutzt, die alle Spielrunden beinhalten. Da zwischen drei respektive vier verschiedenen Ausgaben unterschieden wird, wird die Mehrklassen-SVM One-Versus-All verwendet (vergleiche Abschnitt 3.3.4).

5.5.1.2 Linearer Kernel

Für den linearen Kernel muss ein geeigneter Parameter C bestimmt werden. Dieser Wert wird mithilfe einer Grid-Suche ermittelt. Dabei wird der Parameter C in einem Intervall $[0, 100]$ mit einer Schrittweite von 5 untersucht. Stichprobenartige Vortests zur Bestimmung des C (Tests im Intervall von $[0, 250]$ mit Schrittweite 50) haben Werte oberhalb von 100 als zu hoch gezeigt. Mit dreifacher Kreuzvalidierung werden die Muster der Parameter-Testspieler getestet, die alle Spielrunden enthalten. Dabei werden die Muster in drei Partitionen geteilt, wobei jeweils zwei Partitionen die Trainingsmenge darstellen und eine Partition die Testmenge. Die Ergebnisse werden über die Durchläufe der einzelnen Spieler gemittelt. Die Ergebnisse der Grid-Suche sind in den Abbildungen 5.9 und 5.10 zu sehen, jeweils für 3 bzw. 4 Ausgabeneuronen. Die Abszisse gibt die Werte für C in einer Schrittweite von 5 an und an der Ordinate sind die entsprechenden Klassifikationsraten abgetragen. Im Durchschnitt über die Klassifikationsraten über alle Spieler ist zu erkennen, dass für $C = 5$ die durchschnittlich beste Klassifikationsrate erreicht wird. Ein größeres C verbessert die Klassifikationsrate nicht.

5.5.1.3 Polynomieller Kernel

Beim polynomiellen Kernel hat sich in stichprobenartigen Vortests ($b \in [2, 10]$, Schrittweite 1, konstantem $C = 5$ und $d = 2$) gezeigt, dass die Änderung des Parameters b eine sehr geringe Auswirkung auf die Klassifikationsrate hat. Daher wird der Parameter b auf den standardmäßig verwendeten Wert $b = 1$ festgelegt. Der Grad des polynomiellen Kernels wird mithilfe einer Grid-Suche bestimmt. Der Wert $C = 5$ wird zunächst auf den ermittelten Wert des linearen Kernels gesetzt. Getestet werden alle Grade d, $d \in [2, 10] \subseteq \mathbb{N}$. Als vielversprechendster Grad stellt sich $d = 2$ heraus, wie in den Abbildungen 5.11 und 5.12 zu sehen.

Es wird eine erneute Grid-Suche ausgeführt, die für den Grad $d = 2$ einen optimalen Wert für C ermittelt. Das Ergebnis ist in den Abbildungen 5.13 und 5.14 zu erkennen ist. Dabei lässt sich zunächst ein Anstieg von C festellen, der auch bei $C = 100$ noch anzusteigen scheint. Daher wird eine weitere Grid-Suche mit einer Schrittweite von 50 durchgeführt, um zu überprüfen, ob sich die Klassifikationsrate bei wachsendem C noch weiter verbessert. Wie in den Abbildungen 5.15 und 5.16 zu sehen, ist durchschnittlich ab einem $C = 100$ keine

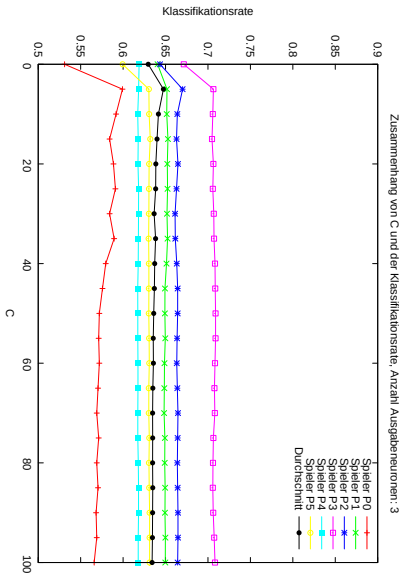

Abb. 5.9: *Grid-Suche für Parameter C mit 3 Ausgabeneuronen und linearem Kernel*

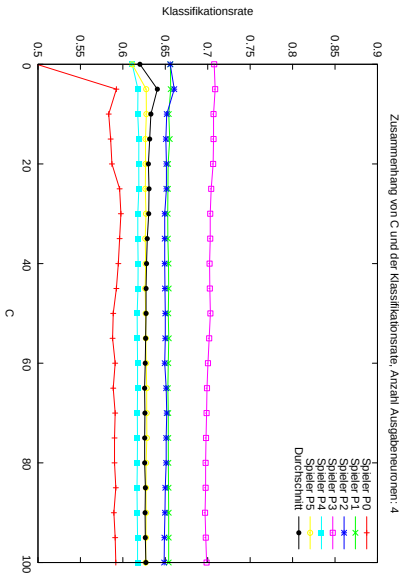

Abb. 5.10: *Grid-Suche für Parameter C mit 4 Ausgabeneuronen und linearem Kernel*

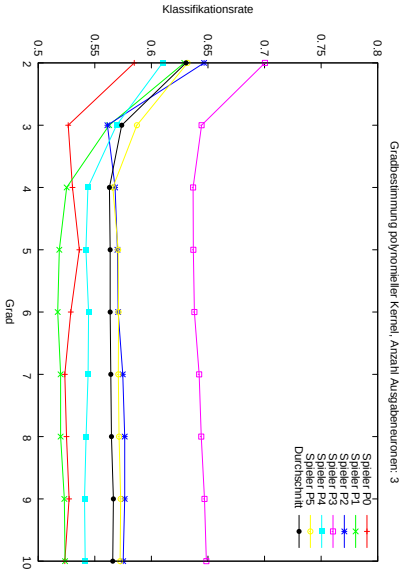

Abb. 5.11: *Gradbestimmung für polynomiellen Kernel mit 3 Ausgabeneuronen*

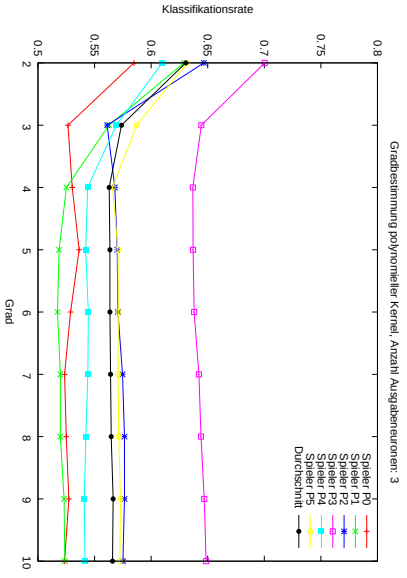

Abb. 5.12: *Gradbestimmung für polynomiellen Kernel mit 4 Ausgabeneuronen*

nennenswerte Erhöhung der Klassifikationsrate zu verzeichnen. Damit wird der Parameter $C = 100$ für den polynomiellen Kernel festgelegt.

5.5.2 Versuchsaufbau

Auch hier werden für jeden der Testspieler 16 SVM aufgebaut, jeweils für die Muster mit allen Spielrunden, für die Flop-, Turn-, River-Spielrunden, den linearen und polynomiellen Kernel und die Anzahl der verschiedenen Ausgaben 3 und 4. Für den linearen Kernel wird wie in den vorexperimentellen Planungen beschrieben der Wert $C = 5$ gewählt, für den polynomiellen Kernel $C = 100$, $b = 1$ und der Grad $d = 2$. Die SVM lernen an den in Abschnitt 5.2.5 vorgestellten Trainings-Mustern für die jeweiligen Spielrunde und werden an den entsprechenden Testmustern getestet. Da zwischen mehr als zwei Klassen unterschieden wird, wird die Mehrklassen-SVM One-Versus-All verwendet. Die Tests werden mithilfe des Progamms RapidMiner mit dem Modul LIBSVM durchgeführt.

5.6 Ergebnisse

Für die Auswertung der Tests mit den KNN wird der Median der fünf Testläufe betrachtet, um der Abweichung der Klassifikationsrate durch die zufällige Initialisierung der Gewichte im KNN entgegenzuwirken. Bei der SVM ist keine Mehrfachausführung nötig, da keine randomisierten Komponenten enthalten sind.

Bei den Ergebnissen der Versuche, in denen zwischen den vier Aktionsmöglichkeiten unterschieden wird, werden die Aktionen Check und Call zur Aktion Check „*verschmolzen*“, also zusammengefasst, indem ein vorhergesagter Check bei einem tatsächlichen Call einer richtigen Klassifikation entspricht und ein vorhergesagter Call bei tatsächlichem Check ebenfalls als korrekt klassifiziert angesehen wird. Somit können diese mit den Ergebnissen der Versuche mit drei Aktionen verglichen werden. Aus den Ergebnissen, welche die Lernverfahren für die Muster mit allen Spielrunden erzeugen, werden die einzelnen Spielrunden extrahiert, um sie mit den Versuchen mit den einzelnen Spielrunden vergleichbar zu machen.

Die Darstellung in so genannten Confusion-Matrizen gibt eine Übersicht darüber, wie die Testmuster klassifiziert worden sind. Die Tabellen 5.7 und 5.8 stellen beispielhaft solche Confusion-Matrizen dar, bei der die Confusion-Matrix 5.7 in eine mit nur drei möglichen Aktionen überführt wird, also Check und Call zusammengefasst werden. Tabelle 5.8 stellt die zusammengefasste Confusion-Matrix dar. Dabei sind in den einzelnen Spalten die vorhergesagte Aktion und in den Zeilen die tatsächliche Aktion zu sehen. Die Diagonale zeigt die korrekt klassifizierten Muster an. Am Spalten- bzw. Zeilenende sind die Summen der Aktion abgetragen, die vorhergesagt bzw. tatsächlich durchgeführt worden sind. In der letzten Zelle ist die Summe der korrekt klassifizierten Muster zu sehen. Ihr Verhältnis zur gesamten Anzahl der Muster ergibt die Klassifikationsrate, die als Bewertungskriterium für die Güte des Klassifikators verwendet wird. Dargestellt sind die Confusion-Matrizen beispielhaft für die Testmustermenge *rand.Pat.6* von *Spieler 4* für die Spielrunde *Turn*, klassifiziert durch eine SVM mit *linearem Kernel* (auch lineare SVM genannt). Die Interpretation dieser Confusion-Matrizen erfolgt in Abschnitt 5.7.3.

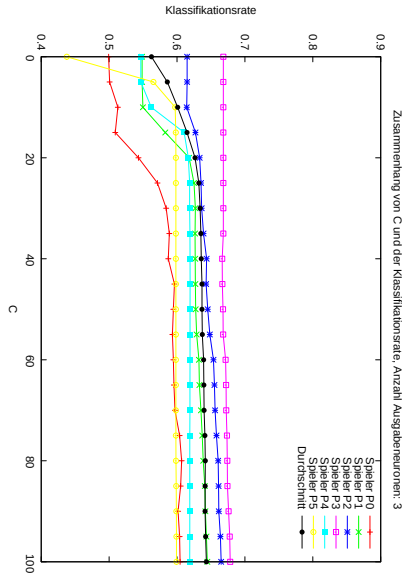

Abb. 5.13: *Grid-Suche für Parameter C mit 3 Ausgabeneuronen, polynomieller Kernel, Schrittweite 5*

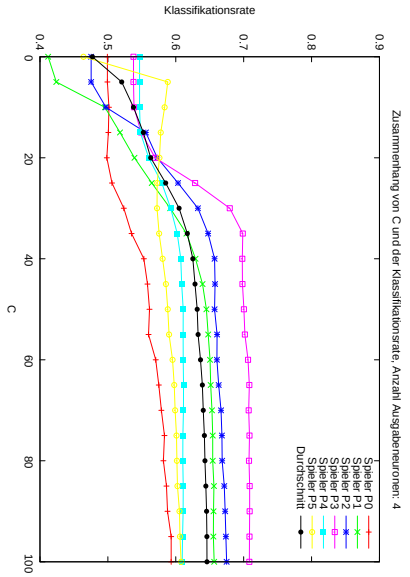

Abb. 5.14: *Grid-Suche für Parameter C mit 4 Ausgabeneuronen, polynomieller Kernel, Schrittweite 5*

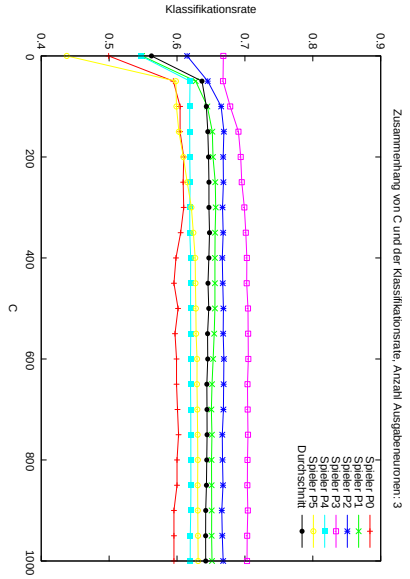

Abb. 5.15: *Grid-Suche für Parameter C mit 3 Ausgabeneuronen, polynomieller Kernel, Schrittweite 50*

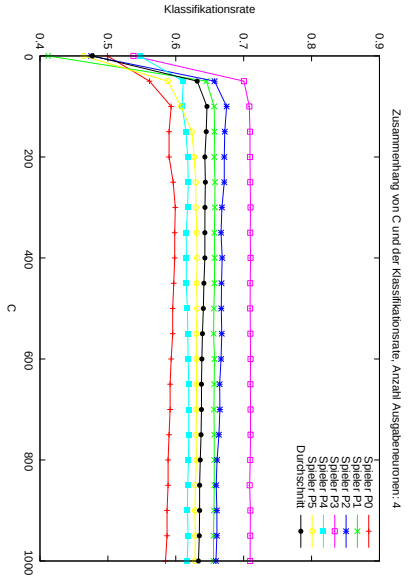

Abb. 5.16: *Grid-Suche für Parameter C mit 4 Ausgabeneuronen, polynomieller Kernel, Schrittweite 50*

Tab. 5.7: *Beispielhafte Confusion-Matrix, Sp.4, Turn, rand.Pat.6, linearer Kernel, 4 Ausgabeneuronen, Klassifikationsrate: 67.2% (232/345)*

	vorherg. Fold	vorherg. Check	vorherg. Call	vorherg. Raise	Summe
Fold	30	0	4	2	36
Check	0	109	0	10	119
Call	20	1	7	15	43
Raise	7	51	3	86	147
Summe	57	161	14	113	232

Tab. 5.8: *Beispielhafte „verschmolzene" Confusion-Matrix, Sp.4, Turn, rand.Pat.6, linearer Kernel, zusammengefasst zu 3 Ausgabeneuronen, Klassifikationsrate: 67.5% (233/345)*

	vorherg. Fold	vorherg. Call	vorherg. Raise	Summe
Fold	30	4	2	36
Call	20	117	25	162
Raise	7	54	86	147
Summe	57	175	113	233

Simpler Klassifikator Die Klassifikatoren, die an allen Spielrunden lernen, werden *All-Klassifizierer* genannt, die für die einzelnen Spielrunden *Flop-*, *Turn-* und *River-Klassifizierer*. Für den Vergleich mit den Klassifikationsraten der KNN und SVM wird ein *simpler Klassifikator* genutzt. Dieser sagt für ein Muster immer die Aktion vorher, die ein Spieler in einer Spielrunde am häufigsten wählt. Die resultierende Klassifikationsrate wird *Referenz* oder *Referenz-Klassifikationsrate* genannt und in den Grafiken mit *Meistgewaehlt* dargestellt.

5.6.1 Eignung der Klassifikationsverfahren

Es soll untersucht werden, wie gut künstliche neuronale Netze und Support-Vector-Machines Spielsituationen beim Pokerspiel klassifizieren, so dass eine Vorhersage der gewählten Aktion eines Spielers möglich ist. Um dies beurteilen zu können, wird die erzielte Klassifikationsrate mit der Referenz-Klassifikationsrate verglichen.

Tabelle 5.9 stellt dar, wie sich die Muster auf die jeweiligen Testmengen verteilen. *Flop Ant. M.* (Anteil Muster) zeigt an, wieviel Prozent der gesamten Testmuster Flop-Muster sind, also nur die Muster enthält, die Flop-Spielsituationen darstellen. Es ist zu erkennen, dass bei allen Spielern etwa 42% aller Testmuster Flop-Spielsituationen, 32% Turn- und 26% River-Spielsituationen sind. Die Werte sind gemittelt über die elf verschiedenen Testmustermengen. Es ist zu erkennen, dass alle Spieler sich in ungefähr identischem Verhältnis an den Spielrunden teilnehmen, wenn sie nicht vor dem Flop ausgestiegen sind. Der Anteil der von einem Spieler in der jeweiligen Spielrunde am häufigsten gewählte Aktion ist in der Spalte *Mgw. Akt.* (meistgewählte Aktion) abzulesen und gibt die Referenz-Klassifikationsrate des simplen Klassifikators an. In der Klammer ist die Aktion abzulesen, die in der Spielrunde von dem Spieler am häufigsten ausführt wird. Für den Call steht das (C) und für das Raise das (R).

Fold ist bei keinem Spieler die meistgewählte Aktion in einer Spielrunde. *Spieler 1*, *3* und *4* checken in jeder Spielrunde am häufigsten. Dasselbe gilt für *Spieler 2*, wobei dieser jedoch in der River-Spielrunde häufiger raiset. Die *Spieler 0*, *5* und *6* raisen in allen Spielrunden mit Ausnahme von *Spieler 6*, der in der River-Spielrunde stattdessen häufiger checkt. Diese Spieler sind als aggressiver einzustufen als die übrigen Spieler, die eher nur callen. *Spieler 2* weist in der Flop-Spielrunde mit dem Wert von 0.80 den höchsten Wert für eine meistgewählte Aktion auf, wobei *Spieler 0* mit 0.46 in der Flop-Spielrunde den niedrigsten Wert aufweist. Auch in der Spielrunde Turn weist *Spieler 0* den niedrigsten Wert auf. In der River-Spielrunde ist es *Spieler 6* mit einem Wert von 0.49.

Tab. 5.9: *Verteilung der Muster und meistgewählte Aktion: Für die jeweiligen Spielrunden ist der Anteil der Muster an allen Mustern zu sehen. Mgw. Akt. gibt an, wie hoch der Anteil der meistgewählten Aktion ist. Diese Aktion ist der (C)all oder das (R)aise. Der höchster Wert der meistgeählten Aktion ist fett, der niedrigste Wert ist kursiv dargestellt.*

Sp.	All	Flop		Turn		River	
	Mgw. Akt.	Mgw. Akt.	Ant. M.	Mgw. Akt.	Ant. M.	Mgw. Akt.	Ant. M.
0	0.49 (R)	*0.46* (R)	0.46	0.53 (R)	0.32	0.53 (R)	0.22
1	0.65 (C)	0.71 (C)	0.41	0.65 (C)	0.34	0.54 (C)	0.25
2	0.64 (C)	**0.80** (C)	0.41	0.69 (C)	0.32	0.53 (R)	0.27
3	0.69 (C)	0.70 (C)	0.41	0.70 (C)	0.34	0.66 (C)	0.25
4	0.56 (C)	0.54 (C)	0.42	0.54 (C)	0.32	0.60 (C)	0.26
5	0.70 (R)	0.66 (R)	0.40	0.72 (R)	0.33	0.75 (R)	0.27
6	0.54 (R)	0.60 (R)	0.41	0.55 (R)	0.32	0.49 (C)	0.27

Spieler, bei denen der Anteil der meistgewählten Aktion niedrig ist, sind schwierig vorherzusagen. Sie wählen nicht häufig die gleiche Aktion, sondern variieren ihr Spiel. Eine hohe Klassifikationsrate zeigt an, dass das Klassifikationsverfahren eine Regelmäßigkeit im Spielverhalten des Spielers erkennt und gut dazu in der Lage ist, die Aktion des Spielers vorherzusagen. Die Differenz zwischen der Klassifikationsrate eines Klassifikators und der Referenz-Klassifikationsrate zeigt an, wie sehr sich der Einsatz von Klassifikationsverfahren bei einem Spieler lohnt.

Die Ergebnistabellen und Plots für alle Spieler sind im Anhang A zu finden. Im Folgenden werden exemplarisch die Ergebnisse für *Spieler 0* dargestellt. Tabelle 5.10 zeigt die Ergebnisse für die KNN, in der ersten Spalte für das Backpropagation-Verfahren mit 3 Ausgabeneuronen und in der zweiten Spalte mit 4 Ausgabeneuronen, wobei die Aktionen Check und Call zusammengefasst sind. Die dritte und vierte Spalte zeigen die Ergebnisse für das Resilient-Propagation-Verfahren, auch jeweils mit 3 und 4 Ausgabeneuronen. Die letzte Spalte gibt die Referenz-Klassifikationsrate des simplen Klassifikators an. Die Zeilen stellen die Muster dar, auf denen getestet worden ist. *Flop*, *Turn* und *River* beinhalten jeweils nur die Muster der jeweiligen Spielrunde. Bei *All* wird mit den Mustern getestet, in denen alle Spielrunden vertreten sind. Aus diesen All-Ergebnissen werden die Ergebnisse für die einzelnen Spielrunden extrahiert. Dazu werden beispielsweise für den *Flop* nur die Ergebnisse des All-Klassifizierers für die Flop-Muster betrachtet. Dies zeigt der Zusatz *v.A.* an. Die abgebildeten Werte sind jeweils der Durchschnitt aus den Klassifikationsraten der elf Testmustermengen (vergleiche Abschnitt 5.2.5).

Es ist zu erkennen, dass mit der Klassifikationsrate von 0.5895 (in der Tabelle 5.10 kursiv dargestellt) für den *Flop* mit dem Backpropagation-Verfahren und 3 Ausgabeneuronen (auch kurz *BP3* genannt) der mit Abstand schlechteste Wert erreicht wird. Für die gleiche Spielrunde wird mit dem Resilient-Propagation-Verfahren und 3 Ausgabeneuronen (auch kurz *RE3* genannt) mit 0.6606 eine um 0.0711 bessere Klassifikationsrate erreicht (Abweichungen werden absolut angegeben, wenn nicht anders bemerkt). Trotzdem liegt BP3 noch immer deutlich über der Referenz des simplen Klassifikators. Eine weitere große Differenz ist bei der Spielrunde *Turn v.A.* mit BP3 und RE3 von 0.034 zu sehen. Den höchsten Wert erreicht die KNN mit 3 Ausgabeneuronen und dem Resilient-Propagation-Verfahren in der Turn-Spielrunde (in der Tabelle fett dargestellt). Innerhalb der übrigen Spielrunden unterscheiden sich die Klassifikationsraten der verschiedenen KNN nicht wesentlich voneinander. Es ist jedoch zu erkennen, dass für *Spieler 0* das Resilient-Propagation-Verfahren abgesehen von *Turn* und *River v.A.* stets besser klassifiziert als das Backpropagation-Verfahren. Die größte Verbesserung gegenüber der Referenz-Klassifikationsrate wird in der Turn-Spielrunde mit RE3 erreicht. Die Verbesserung beträgt 0.223.

Tab. 5.10: *Durchschnittliche Klassifikationsraten, Sp. 0, KNN: Bester Wert fett, schlechtester Wert kursiv dargestellt.*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.6739	0.6817	0.6966	0.6940	0.4935
Flop	*0.5895*	0.6102	0.6606	0.6591	0.4639
Flop v.A.	0.6316	0.6386	0.6563	0.6551	0.4639
Turn	0.7533	0.7498	**0.7562**	0.7487	0.5332
Turn v.A.	0.7148	0.7154	0.7488	0.7393	0.5332
River	0.6915	0.6829	0.7081	0.7079	0.5255
River v.A.	0.6907	0.7046	0.6962	0.7002	0.5255

Tabelle 5.11 stellt die Ergebnisse für *Spieler 0* mit den SVM dar, jeweils für den linearen und polynomiellen Kernel mit 3 respektive 4 Ausgabeneuronen. Hier ist zu sehen, dass die SVM mit linearem Kernel und 4 Ausgabeneuronen (auch kurz Lin4) die beste Klassifikationsrate beim *Turn* von 0.7566 erreicht. Am schlechtesten schneidet die SVM mit linearem Kernel und 3 Ausgabeneuronen bei *Flop v.A.* ab. Hier beträgt die Klassifikationsrate 0.6595. Die Abweichungen in den einzelnen Spielrunden zwischen dem linearen und polynomiellen Kernel sind gering. Die größte Abweichung liegt bei 0.0247 (*River*). Am geringsten weichen die Klassifikationsraten beim *Turn* ab, wo die Abweichung nur höchstens 0.0040 beträgt. Es ergibt sich eine durchschnittliche Abweichung der Klassifikationsraten der einzelnen Spielrunden durch die SVM von nur 0.0143. Gegenüber der Referenz-Klassifikationsrate erreicht Lin4 in der Turn-Spielrunde die größte Verbesserung von 0.2234. Beim *Turn* erreichen sowohl die KNN als auch die SVM die größten Verbesserungen gegenüber der Referenz. In dieser Spielrunde ist *Spieler 0* demnach gut vorhersagbar.

In den Abbildungen 5.17 und 5.18 sind die Klassifikationsraten der elf Testmustermengen (*sort.Pat bis rand.Pat.9*) und die durchschnittliche Klassifikationsrate dieser Mustermengen von *Spieler 0* graphisch dargestellt, auszugsweise für die Spielrunden Flop (3 Ausgabeneuronen) und Turn (4 Ausgabeneuronen). Im Vergleich zu den Klassifikationsraten der Lernverfah-

Tab. 5.11: *Durchschnittliche Klassifikationsraten, Sp. 0, SVM: bester Wert fett, schlechtester Wert kursiv dargestellt.*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.6935	0.6975	0.7003	0.7046	0.4935
Flop	0.6620	0.6700	0.6674	0.6727	0.4639
Flop v.A.	*0.6595*	0.6686	0.6718	0.6747	0.4639
Turn	0.7553	**0.7566**	0.7532	0.7526	0.5332
Turn v.A.	0.7492	0.7352	0.7415	0.7481	0.5332
River	0.7091	0.7076	0.6844	0.6847	0.5255
River v.A.	0.6835	0.7031	0.6999	0.7039	0.5255

ren ist die Referenz-Klassifikationsrate des simplen Klassifikators angegeben. Die Verbindungslinien der Punkte in den Grafiken dienen allgemein zur Übersichtlichkeit, damit die Punkte eines einzelnen Verfahrens besser zu verfolgen sind, da sie teilweise von anderen Punkte überdeckt sind. Es ist direkt ersichtlich, dass die Klassifikationsraten aller Klassifikationsverfahren sowohl in der Flop-Spielrunde als auch in der Turn-Spielrunde deutlich über der Referenz des simplen Klassifikators liegen. Sowohl die Ergebnisse der KNN als auch die der SVM weichen nicht stark voneinander ab. Eine Ausnahme stellt das Backpropagation-Verfahren in der Flop-Spielrunde dar, wobei die Klassifikationsraten teilweise deutlich von denen der anderen Verfahren abweichen. Nur bei der Mustermenge *rand.Pat.9* ist bei allen Verfahren eine nahezu identische Klassifikationsrate zu sehen. Die größte Abweichung zu den Klassifikationsraten der übrigen Verfahren ist bei der Mustermenge *rand.Pat.0* mit 0.15 zu sehen. Eventuell sind die 20 Epochen für das Training der KNN mit dem Backpropagation-Verfahren nicht ausreichend für *Spieler 0* gewesen und eine Verbesserung der Klassifikationsrate stellt sich erst nach weiteren Epochen ein.

In der Turn-Spielrunde in Abbildung 5.18 sind keine solche Ausreißer zu beobachten, jedoch liegen die Klassifikationsraten des KNN mit dem Backpropagation-Verfahren durch den All-Klassifizierer abgesehen von der Mustermenge *sort.Pat.* stets unterhalb denen der übrigen Lernverfahren. Die größte Abweichung von 0.084 ergibt sich bei der Mustermenge *rand.Pat.8.* Das Backpropagation-Verfahren hat bei *Spieler 0* durchschnittlich am schlechtesten abgeschnitten, wogegen die SVM mit beiden Kernel am besten abgeschnitten hat.

Abbildung 5.19 zeigt Boxplots der Ergebnisse des All-Klassifizierers und 3 Ausgabeneuronen. Dargestellt sind die Boxplots für die lineare SVM (SVM Lin.), die polynomielle SVM (SVM Pol.), das KNN mit Backpropagation (KNN BP) und mit Resilient-Propagation (KNN RE). Die in der Abbildung dargestellten Boxplots zeigen die Verteilung der Klassifikationsraten der elf Testmustermengen an. Für jeden Spieler sind die Boxplots für die Ergebnisse der Klassifikationsverfahren nebeneinander dargestellt, um die Verfahren miteinander vergleichen zu können. Zudem ist in der Abbildung die durchschnittliche Referenz-Klassifikationsrate der Spielrunde eingetragen. Es ist zu erkennen, dass der All-Klassifizierer aller Lernverfahren für alle Spieler Klassifikationsraten erreicht hat, die oberhalb der Referenz-Klassifikationsrate liegen. Die lineare SVM liefert jedoch bei *Spieler 3* und *Spieler 6* bei einigen Mustermengen eine geringere Klassifikationsrate. Im Durchschnitt jedoch liegen sie über der Referenz. Bei *Spieler 5* ist eine Streuung zu sehen, bei der die beiden Extrema sich um 0.193 unterscheiden. In Ab-

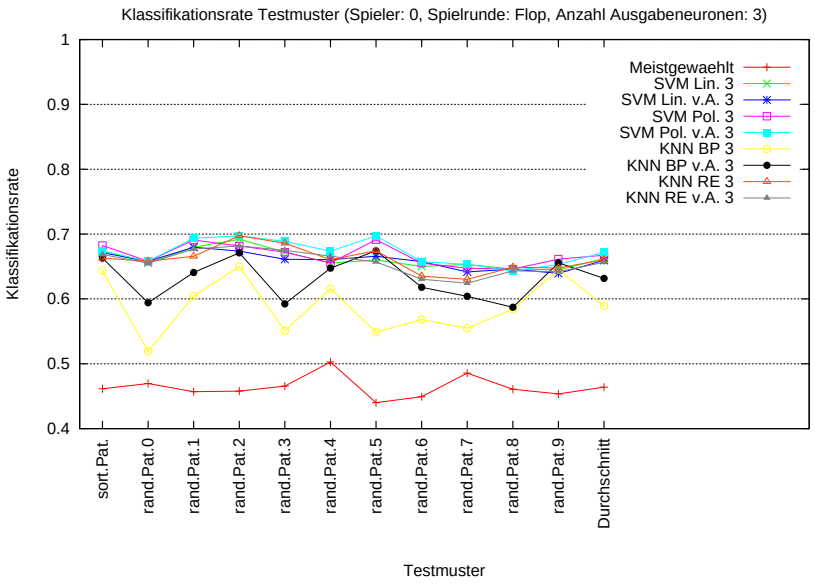

Abb. 5.17: *Klassifikationsraten der Testmustermengen, Sp. 0, Flop, 3 Ausgabeneuronen*

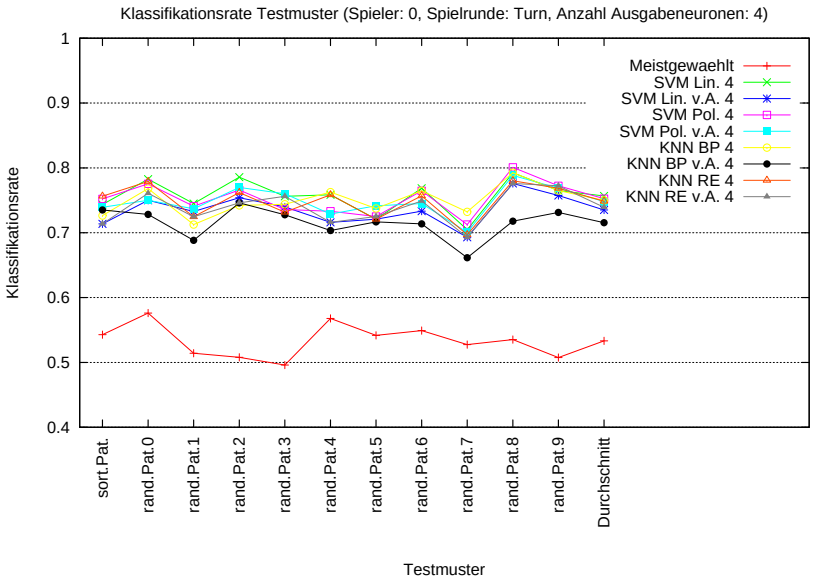

Abb. 5.18: *Klassifikationsraten der Testmustermengen, Sp. 0, Turn, 4 Ausgabeneuronen*

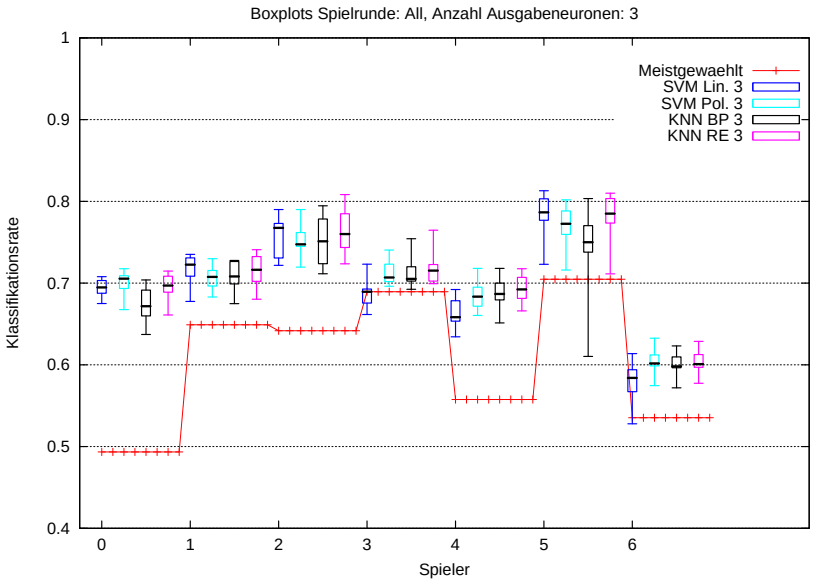

Abb. 5.19: *Boxplots der Spielrunde All, 3 Ausgabeneuronen*

bildung 5.20, welche die Klassifikationsraten für die einzelnen Testmustermenge von *Spieler 5* darstellen, ist jedoch zu erkennen, dass in der Mustermenge mit den chronologisch sortierten Mustern (*sort.Pat.*) auch die Referenz-Klassifikationsrate niedriger liegt und noch immer eine bessere Klassifikation durch das KNN mit dem Backpropagation-Verfahren erreicht wird. Die Abweichung bei dieser Mustermenge zu den anderen Verfahren liegt bei ca. 0.11. Bei allen Mustermengen erreichen die übrigen Verfahren annähernd gleiche Klassifikationsraten.

Bei *Spieler 0* ist in Abbildung 5.19 der größte Unterschied zwischen der Referenz-Klassifikationsrate und der von den KNN und SVM erzielten Klassifikationsraten zu sehen. Im Durchschnitt erreichen die Verfahren eine verbesserte Klassifikationsrate von ungefähr 0.2. Im Gegensatz dazu liegt bei *Spieler 3* diese Differenz nur bei durchschnittlich 0.019. Dies ist auch in den Tabellen 5.12 und 5.13 zu sehen, welche die durchschnittlichen Klassifikationsraten für *Spieler 3* darstellen. Es ist zu sehen, dass die SVM mit linearem Kernel und 3 Ausgabeneuronen in allen Spielrunden, abgesehen vom *Flop* und *Flop v.A.*, die jeweils schlechteste Klassifikationsrate liefert. Bei *All*, *Turn v.A.*, *River* und *River v.A.* liegen sie sogar unterhalb der Referenz-Klassifikationsrate. Dagegen klassifizieren die KNN sowohl mit dem Backpropagation- als auch mit dem Resilient-Propagation-Verfahren nahezu durchgängig besser als die SVM. Das KNN mit Resilient-Propagation-Verfahren und 4 Ausgabeneuronen erreicht die maximale Verbesserung zur Referenz-Klassifikationsrate von 0.0422 bei *River*. Die Bietstrategie von *Spieler 3* ist demnach schwierig zu durchschauen. Dennoch schneiden die Klassifikationsverfahren durchschnittlich nicht schlechter als der simple Klassifikator ab.

Die Boxplots zu den Ergebnissen der Spielrunde Flop sind in Abbildung 5.21 zu sehen. Es werden hier nur die Klassifikationsraten der Flop-Muster betrachtet, die durch den Einsatz

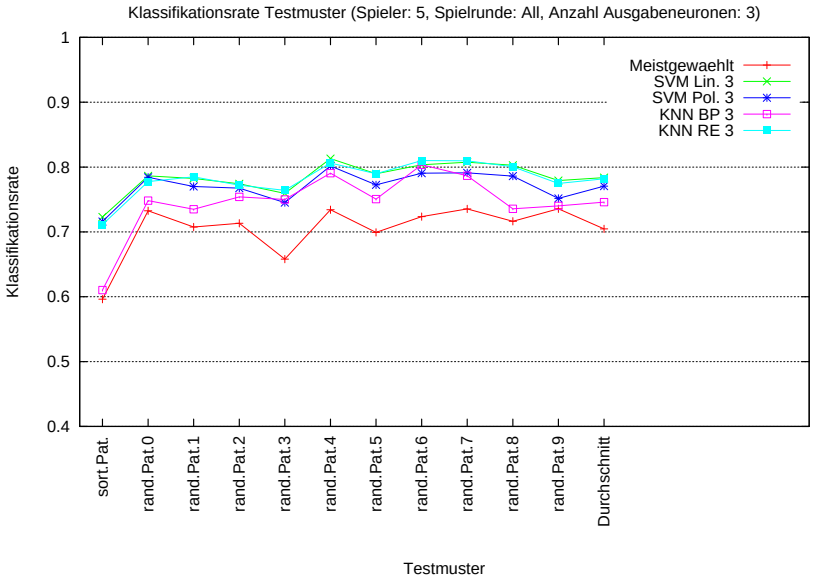

Abb. 5.20: *Klassifikationsraten der Testmustermengen, Sp. 5, All, 3 Ausgabeneuronen*

Tab. 5.12: *Durchschnittliche Klassifikationsraten, Sp. 3, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.6873	0.6994	0.7128	0.7136	0.6894
Flop	0.7117	0.7144	0.7083	0.7073	0.7029
Flop v.A.	0.7097	0.7174	0.7094	0.7075	0.7029
Turn	0.7009	0.7082	0.7093	0.7021	0.6966
Turn v.A.	0.6855	0.7009	**0.7315**	0.7301	0.6966
River	0.6577	0.6785	0.6755	0.6822	0.6586
River v.A.	*0.6533*	0.6684	0.6939	0.7017	0.6586

Tab. 5.13: *Durchschnittliche Klassifikationsraten, Sp. 3, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.7120	0.7125	0.7168	0.7118	0.6894
Flop	0.7119	0.7063	0.7085	0.7125	0.7029
Flop v.A.	0.7086	0.7179	0.7143	0.7141	0.7029
Turn	0.7190	0.7267	0.7251	0.7218	0.6966
Turn v.A.	0.7234	0.7224	**0.7317**	0.7174	0.6966
River	*0.6675*	0.6808	0.6828	0.7008	0.6586
River v.A.	0.6948	0.6944	0.6947	0.6865	0.6586

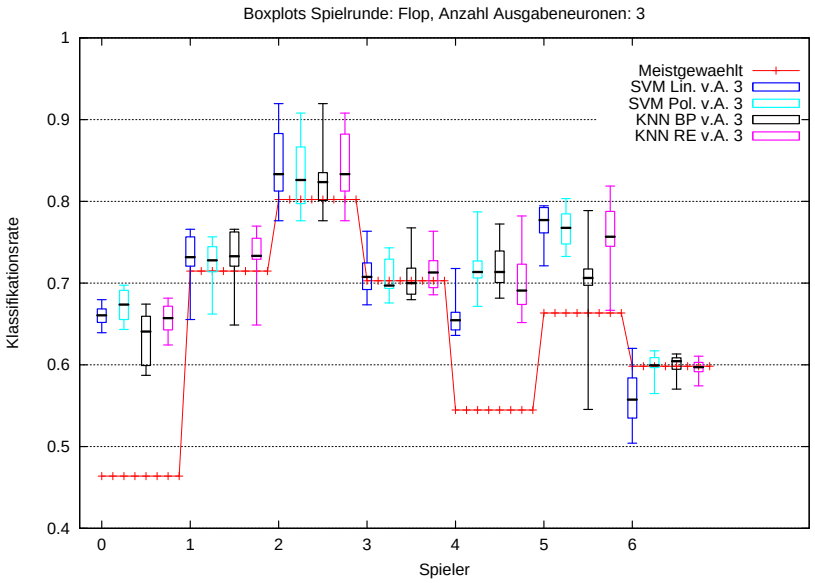

Abb. 5.21: *Boxplots der Spielrunde Flop, 3 Ausgabeneuronen*

des All-Klassifizierers ermittelt worden sind. Dies zeigt der Zusatz *v.A.* an. Es ist zu erkennen, dass bei *Spieler 0* auch in dieser Spielrunde die Klassifikationsraten deutlich über der Referenz liegen. Bei *Spieler 2* ist zu sehen, dass trotz der sehr hohen Referenz-Klassifikationsrate von 0.80 die durchschnittlichen Ergebnisse der Lernverfahren noch immer besser sind. Auch bei den *Spielern 1*, *3* und *6* ist zu sehen, dass die Klassifikationsraten der Verfahren über denen der Referenz liegen.

Abbildung 5.22 stellt die Klassifikationsraten der Testmustermengen für *Spieler 6* dar. Es ist besonders gut zu erkennen, dass die lineare SVM nicht gut dazu in der Lage ist, die Flop-Muster des *Spielers 6* zu klassifizieren. Es gibt keine Anhaltspunkte, die dieses ungewöhnliche Verhalten erklären. Die polynomielle SVM und die KNN erreichen jedoch auch nur Klassifikationsraten, die ungefähr der Referenz-Klassifikationsrate entspricht. Dieser Effekt kann dadurch zustande gekommen sein, dass die Klassifikationsverfahren nur eine trennende Hyperebene gefunden haben, die nahezu jedes Muster in die Klasse Raise klassifiziert, da dadurch der geringste Fehler entsteht.

Wie in Abbildung 5.23 zu erkennen, sind die Turn-Muster der Spieler im Vergleich zu den Flop-Mustern besser zu klassifizieren. Abgesehen von einigen kleinen Ausreißern (*Spieler 1* mit der SVM und polynomiellen Kernel, *Spieler 6* mit der SVM und linearem Kernel) liegen die Klassifikationsergebnisse immer über den Referenzen. Nur bei *Spieler 3* weichen die Ergebnisse der linearen SVM von den übrigen Klassifikationsverfahren stärker ab. Diese Abweichung ist bereits bei dem All-Klassifizierer bemerkt worden. Auch die starke Abweichung der Extrema beim Backpropagation-Verfahren bei *Spieler 5* von 0.21 ist bereits in den Ergebnissen vom *Flop* und *All* aufgefallen. *Spieler 0* und *4* werden wiederum im Vergleich zur

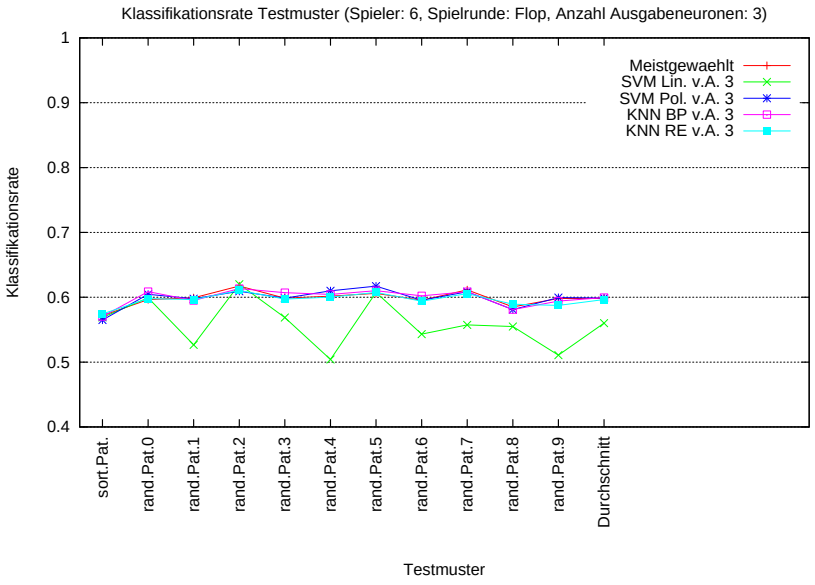

Abb. 5.22: *Klassifikationsraten der Testmustermengen, Sp. 6, Flop, 3 Ausgabeneuronen*

Referenz-Klassifikationsrate von allen Verfahren sehr gut klassifiziert. Bei *Spieler 0* beträgt die Differenz durchschnittlich 0.2, bei *Spieler 4* 0.14.

In der River-Spielrunde (Abbildung 5.24) ist zunächst zu erkennen, dass die Extremwerte der Klassifikationsraten bei allen Spielern eine größere Differenz aufweisen als in den anderen Spielrunden. Dies ist eventuell dadurch bedingt, dass für die River-Spielrunde die wenigsten Muster vorhanden sind und dadurch der Einfluss dieser Muster auf die Klassifikationsleistung beim Training des All-Klassifierers geringer gewesen ist. Es ist zudem zu sehen, dass *Spieler 1* in der River-Spielrunde besser als in den anderen Spielrunden vorherzusagen ist. Die durchschnittliche Verbesserung gegenüber der Referenz-Klassifikationsrate liegt bei 0.132. Für *Spieler 4*, bei dem in den Spielrunden Flop und Turn die Klassifikationsraten durchschnittlich um 0.16 bzw. 0.14 über der Referenz gelegen haben, beträgt in der River-Spielrunde die Verbesserung nur noch 0.047.

Eine sehr hohe Abweichung der Extrema ist bei *Spieler 2* zu sehen, wobei die Abweichung beim Backpropagation-Verfahren 0.248 beträgt. Diese Streuung wird in Abbildung 5.25 sichtbar, in der die Klassfikationsraten der Testmustermengen für *Spieler 2* in der River-Spielrunde abgebildet sind. Für die hohe Abweichung sind die Mustermengen *rand.Pat.4* (minimaler Wert für Backpropagation) und *rand.Pat.6* (maximaler Wert für Backpropagation) verantwortlich. Besonders auffällig ist hier die Mustermenge *rand.Pat.6*, bei der die SVM mit linearem Kernel und das KNN mit dem Backpropagation-Verfahren eine Klassifikationsrate erzielen, die um 0.265 über der Referenz-Klassifikationsrate des simplen Klassifikators liegen. Bei der Mustermenge *rand.Pat.4* dagegen liegen die Klassifikationsraten aller Verfahren abgesehen von der SVM mit polynomiellem Kernel um durchschnittlich 0.085 unter der Referenz. Die sehr un-

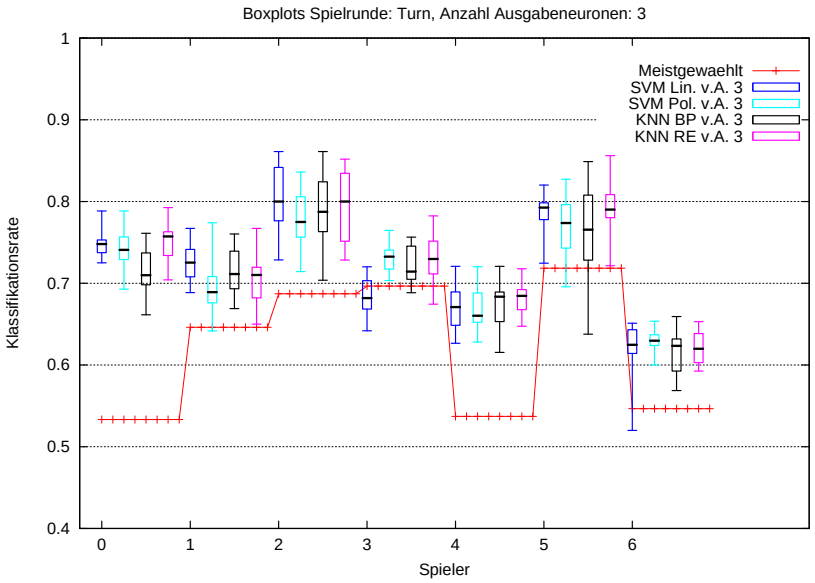

Abb. 5.23: *Boxplots der Spielrunde Turn, 3 Ausgabeneuronen*

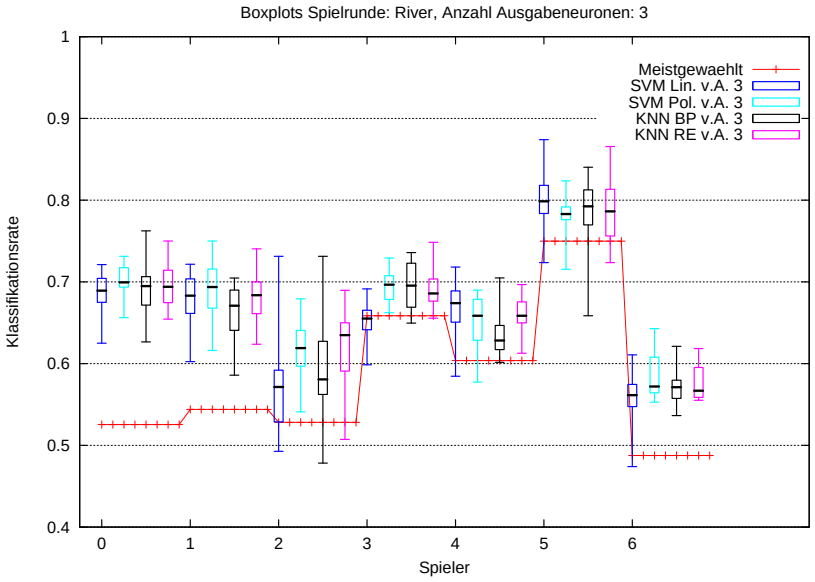

Abb. 5.24: *Boxplots der Spielrunde River, 3 Ausgabeneuronen*

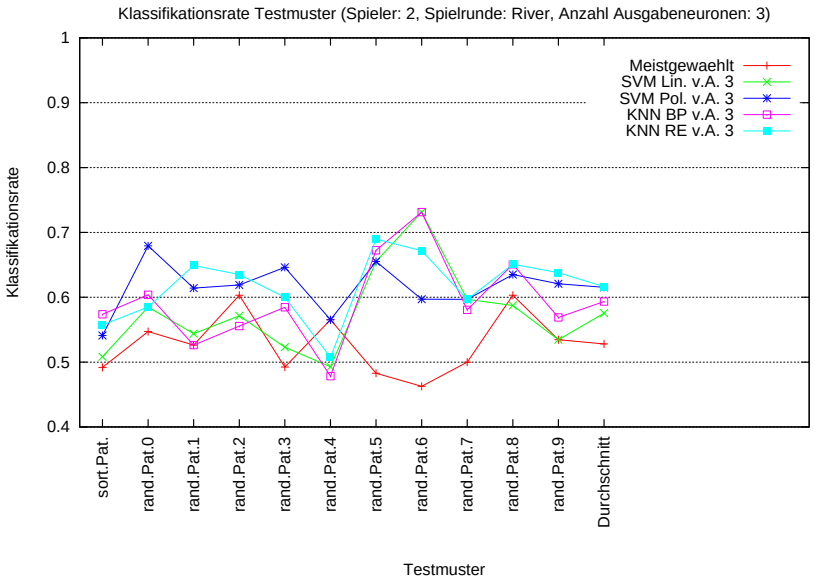

Abb. 5.25: *Klassifikationsraten der Testmustermengen, Sp. 2, River, 3 Ausgabeneuronen*

terschiedlichen Klassifikationsraten der Testmustermengen kann darauf zurückzuführen sein, dass für *Spieler 2* nur sehr wenige Muster für die River-Spielrunde zur Verfügung stehen.

Zusammenfassend ist aus den beschriebenen Boxplots für die Klassifikationsverfahren der einzelnen Spielrunden bei *Spieler 0*, der seine Aktionen relativ ausgeglichen wählt, zu erkennen, dass eine starke Verbesserung von durchschnittlich 0.2 gegenüber den Referenz-Klassifikationsraten der einzelnen Spielrunden erzielt wird. Auch bei *Spieler 4* wird abgesehen von der River-Spielrunde stets eine Klassifikationsrate erzielt, die um durchschnittlich 0.15 über der Referenz liegt. Die Klassifikationsraten von *Spieler 3* fallen im Vergleich zur Referenz nur geringfügig besser aus. Die geringe Verbesserung zur Referenz ist auch bei *Spieler 6* in der Spielrunde River zu sehen. In den übrigen Spielrunden liegen bei *Spieler 6* die durchschnittlichen Klassifikationsraten um ca. 0.08 bis 0.09 über der Referenz. *Spieler 1* weist in der River-Spielrunde eine große Verbesserung zur Referenz auf. Bei *Spieler 2* werden in allen Spielrunden bessere Klassifikationsraten im Vergleich zum simplen Klassifikator erreicht. *Spieler 5* liegt abgesehen vom KNN mit Backpropagation bei der Flop-Spielrunde ca. 0.1 über der Referenz.

Tabelle 5.14 stellt für jeden Spieler die maximalen Klassifikationsergebnisse dar, die mit einem Klassifikationsverfahren für die einzelnen Spielrunden erreicht worden sind. Durchschnittlich sind die Aktionen der Spieler zu 0.7158 korrekt klassifiziert worden. Dabei werden bei *Spieler 5* mit 0.7931 die beste und bei *Spieler 6* mit 0.6171 die schlechteste durchschnittliche Klassifikationsrate erreicht. In der River-Spielrunde ist für *Spieler 5* mit 0.8077 der mit Abstand beste Wert und in der All-Spielrunde mit 0.7870 ebenfalls der beste Wert erreicht worden. In der Flop- und Turn-Spielrunde haben die Klassifikationsverfahren für *Spieler 2* die besten Ergebnisse erzielt. *Spieler 6* ist in allen Spielrunden am schlechtesten klassifiziert worden

Tab. 5.14: *Maximale durchschnittliche Klassifikationsraten aller Spieler in den verschiedenen Spielrunden: Maximaler Wert einer Spielrunde ist fett, der minimale Wert ist kursiv dargestellt.*

Spielrunde	Sp. 0	Sp. 1	Sp. 2	Sp. 3	Sp. 4	Sp. 5	Sp. 6	Durchschn.
All	0.7046	0.7166	0.7645	0.7168	0.6998	**0.7870**	*0.6092*	0.7141
Flop	0.6747	0.7351	**0.8428**	0.7179	0.7334	0.7804	*0.6010*	0.7265
Turn	0.7566	0.7250	**0.8046**	0.7317	0.6842	0.7974	*0.6457*	0.7350
River	0.7091	0.6978	0.6179	0.7017	0.6656	**0.8077**	*0.6125*	0.6875
Durchschn.	0.7113	0.7186	0.7574	0.7170	0.6957	**0.7931**	*0.6171*	0.7158

und liegt durchschnittlich um 0.19 unter den besten Ergebnissen der anderen Spieler. In der Flop-Spielrunde beträgt die Differenz sogar 0.2418. Die Aktionen in der Turn-Spielrunde sind bei allen Spielern durchschnittlich mit 0.7350 am besten vorhersagbar, wogegen in der River-Spielrunde mit 0.6875 die schlechteste Klassifikationsrate erreicht wird. Bei *Spieler 2* ist zu erkennen, dass eine große Differenz zwischen den Klassifikationsraten beim *Flop* und *Turn* zu der beim *River* auftritt. Anscheinend ändert *Spieler 2* sein Spielverhalten in dieser Spielrunde häufig. Eine so starke Schwankung ist bei keinem anderen Spieler zu erkennen. Bei *Spieler 5* beträgt diese Differenz lediglich 0.0273.

Aussagen darüber, dass sich die Lernverfahren für einen bestimmten Spielertypen besonders eignen, lassen sich nicht allgemein treffen. *Spieler 0* und *6* sind als *loose-aggressive* eingestuft (vergleiche Unterkapitel 2.3 und Abschnitt 5.2.6). Wie in den bereits beschriebenen Boxplots zu sehen, sind für *Spieler 0* durchweg große Verbesserungen zur Referenz erkennbar (die Boxplots der Spielrunden sind im Anhang noch einmal gegenübergestellt, siehe Abbildungen A.9 bis A.16). Mit einer durchschnittlichen Klassifikationsrate über alle Spielrunden von 0.7113 (siehe Tabelle 5.14) wird eine Klassifikationsrate erreicht, die im Schnitt bei allen Spielern ermittelt worden ist. Dagegen werden bei *Spieler 6* besonders in der Flop-Spielrunde nur Ergebnisse vergleichbar mit der Referenz erreicht. Auch in den übrigen Spielrunden fallen die Verbesserungen zur Referenz nicht so hoch aus wie bei *Spieler 0*. Zudem ist *Spieler 6*, wie schon erwähnt, der Spieler, bei dem die schlechteste durchschnittliche Klassifikationsrate erreicht worden ist.

Als *loose-passive* werden die *Spieler 1*, *2* und *4* bezeichnet. Auch bei diesen Spielern fallen die Verbesserungen zur Referenz verschieden gut aus. Für *Spieler 4* sind abgesehen von der River-Spielrunde immer sehr gute Ergebnisse im Vergleich zur Referenz erreicht worden. Bei *Spieler 1* dagegen sind besonders in der Flop-Spielrunde Klassifikationsraten ermittelt worden, die sich nur unwesentlich von der Referenz unterscheiden, in den übrigen Spielrunden weisen sie dagegen Verbesserungen auf. Das ist auch bei *Spieler 2* zu erkennen. Bei ihm sind in der Flop- und Turn-Spielrunde die besten Klassifikationsergebnisse ermittelt worden (siehe Tabelle 5.14). Auch wenn beim *Flop* die ermittelte Klassifikationsrate keine große Verbesserung zur Referenz aufweist, sind die Aktionen von *Spieler 2* dennoch gut vorhersagbar.

Der einzige Vertreter für den Spielertypen *tight-passive* ist *Spieler 3*. Für diesen Spieler sind in allen Spielrunden Klassifikationsraten ermittelt worden, die nur unwesentlich besser als die Referenz-Klassifikationsrate sind, wie in den Boxplots zu erkennen. Die durchschnittliche

Klassfikationsrate liegt mit 0.717, wobei dieser Wert dem Durchschnitt bei den hier betrachteten Spielern entspricht.

Auch für den Spielertypen *tight-aggressive* ist nur ein Spieler vertreten, *Spieler 5*. Als *tight-aggressive* werden im Allgemeinen sehr gute Pokerspieler bezeichnet [Ach07]. Bei allen Spielrunden ist für *Spieler 5* zu erkennen, dass die ermittelten Klassifikationsverfahren bessere Ergebnisse liefern als der simple Klassifikator. Zudem ist *Spieler 5* der am besten vorhersagbare Spieler. Es kann jedoch aufgrund der fehlenden Vergleichsspieler für diesen Spielertypen nicht verallgemeinert werden, dass die Spieler dieses Typs immer gut vorhersagbar sind.

Zusammenfassend lässt sich sagen, dass sowohl die künstlichen neuronalen Netze als auch die Support-Vector-Machines in der Lage sind, die Klassifikation der Spielsituationen besser zu vollziehen als der simple Klassifikator, der nur die meistgewählte Aktion vorhersagt. Zudem wird mit einer durchschnittlichen Klassifikationsrate von 0.7158 eine gute Klassifikation vollzogen. Keines der Lernverfahren hat sich als „bestes“ herausgestellt und klassifiziert durchweg besser als ein anderes. Jedoch haben die KNN mit dem Backpropagation-Verfahren häufiger schlechte Ergebnisse geliefert. Auch die Klassifikationsraten der SVM mit linearem Kernel sind teilweise sehr viel schlechter ausgefallen als die der übrigen Verfahren. Auch lässt sich nicht feststellen, dass die Klassifikationsverfahren bei bestimmten Spielrunden immer gute Ergebnisse liefern und bei anderen immer schlechte. Dies unterscheidet sich von Spieler zu Spieler.

5.6.2 Trennung von Call und Check

Es wird untersucht, ob die Trennung zwischen den Aktionen Check und Call und somit eine Erweiterung von drei auf vier Klassen eine Verbesserung der Klassifikation bringt. Dazu sind in Tabelle 5.15 die maximale Verbesserung und maximale Verschlechterung der Klassifikationsraten aller Spielrunden zu sehen, die durch Trennung zwischen Call und Check erreicht werden und erst nach der Klassifikation wieder, wie zu Beginn dieses Kapitels beschrieben, zusammengefasst werden, gegenüber den Klassifikationsraten der Klassifizierer, die Call und Check als eine Aktion behandeln. Die Werte sind aus den Tabellen mit den durchschnittlichen Klassifikationsraten der Spieler ermittelt worden (vergleiche Tabellen A.1 bis A.14 im Anhang). Für *Spieler 1* tritt die maximale Verbesserung durch die Trennung von Check und Call von 0.0122 bei dem KNN mit Resilient-Propagation für den *Flop* auf. Mit 3 Ausgabeneuronen wird eine Klassifikationsrate von 0.7158 erreicht, mit 4 Ausgabeneuronen von 0.7279. Dabei wird die maximale Verschlechterung von 0.0159 mit dem Backpropagation-Verfahren bei *Flop v.A.* erzielt (0.7335-0.7176).

Es ist für alle Spieler zu erkennen, dass die Trennung teilweise eine Verbesserung und teilweise sogar eine Verschlechterung der Klassifikationsrate bringt. Eine große Abweichung ist bei *Spieler 5* zu erkennen. Diese Verschlechterung von 0.0416 ist bei den KNN mit Backpropagation und der Spielrunden River zu finden, wobei der All-Klassifizierer angewendet wird (vergleiche Tabelle A.11). Diese Abweichung kann als Ausreißer bezeichnet werden, da die anderen Spielrunden höchstens Abweichungen der Klassifikationsrate von 0.016 aufweisen. Zur genaueren Betrachtung dieses Ausreißers sind die Klassifikationsraten der Testmustermengen des *Spielers 5* in Abbildung 5.27 dargestellt. Es ist zu erkennen, dass besonders die Mustermengen *rand.Pat.0* und *rand.Pat.2* diese Abweichung bewirken. Durch die geringe Klassifikationsrate

Tab. 5.15: *Maximale Verbesserung und Verschlechterung der Klassifikationsraten (über alle Spielrunden) durch Trennung zwischen Check und Call gegenüber den Ergebnissen der Klassifikationsverfahren, die Call und Check als nur eine Aktion behandeln.*

Sp.	Max. Verb.	Max. Verschl.
0	0.0207	0.0140
1	*0.0122*	0.0159
2	0.0123	0.0292
3	0.0208	0.0143
4	0.0288	0.0163
5	0.0160	**0.0416**
6	0.0278	0.0133

bei diesen Mustermengen ist die durchschnittliche Klassifikationsrate für das KNN mit dem Backpropagation-Verfahren und 4 Ausgabeneuronen ebenfalls gering und führt so zu dem genannten Ausreißer. Auch hier ist die Anzahl von 20 Epochen für das Training eventuell bei den Mustermengen zu gering. Um den Unterschied zu den Klassifikationsraten der Klassifizierer ohne die Trennung von Check und Call zu erkennen, sind diese in Abbildung 5.26 dargestellt. Hier sind keine hohen Abweichungen bei den Mustermengen zu erkennen und daher ist die durchschnittliche Klassifikationsrate auch besser als die bei der Trennung von Check und Call. Abgesehen von den angesprochenen Ausreißern ist bei diesem Spieler zu sehen, dass die Klassifikationsraten der einzelnen Verfahren bei der Trennung von Check und Call nur unwesentlich von denen abweichen, bei denen nur zwischen 3 Aktionen unterschieden wird.

Um die Unterschiede der Klassifikationsraten zwischen der Trennung von Check und Call und der Betrachtung als eine Aktion zu veranschaulichen, sind in den Abbildungen 5.28 und 5.29 die Klassifikationsraten graphisch dargestellt (für die anderen Spielrunden sei auf den Anhang A verwiesen). In der Abbildung 5.28 ist zu erkennen, dass die Trennung von Call und Check keinen Vorteil bringt. Bei den *Spielern 1* und *5* ist sogar zu sehen, dass durch die Trennung von Check und Call beim Backpropagation-Verfahren eine schlechtere Klassifikationsrate erreicht wird. Bei *Spieler 5* ist zudem zu sehen, dass das Resilient-Propagation-Verfahren eine um 0.04 bessere durchschnittliche Klassifikationsrate aufweist als das Backpropagation-Verfahren, jedoch durch die Trennung von Check und Call nur eine minimal bessere Klassifikationsrate erreicht wird. Bei den SVM (siehe Abbildung 5.29) ist auch zu erkennen, dass die Trennung zwischen Call und Check keinen großen Einfluss auf die Klassifikationsrate hat. Bei den *Spielern 3* und *6* wird der SVM mit linearem Kernel eine sehr geringe Verbesserung durch die Trennung erreicht. Alle Klassifikationsergebnisse liegen über der Referenz-Klassifikationsrate des simplen Klassifikators.

Tabelle 5.7 (siehe Seite 62) stellt eine typische Confusion-Matrix dar, wie sie für 4 Ausgabeneuronen ermittelt wird. Durchgängig bei allen Tests aller Spieler ist das Muster zu erkennen, dass ein tatsächlicher Fold nahezu nie als Check vorhersagt wird. Ebenfalls wird fast nie ein Check als Fold vorhergesagt. Auch ein Call wird sehr selten als Check und ein tatsächlicher Check sehr selten als Call vorhergesagt. Die Klassifikationsverfahren sind demnach sowohl gut in der Lage, Fold von Check zu unterscheiden als auch zwischen den Aktionen Check und Call zu trennen. Ein falsch vorhergesagtes Raise tritt sowohl bei tatsächlichem Check als auch bei

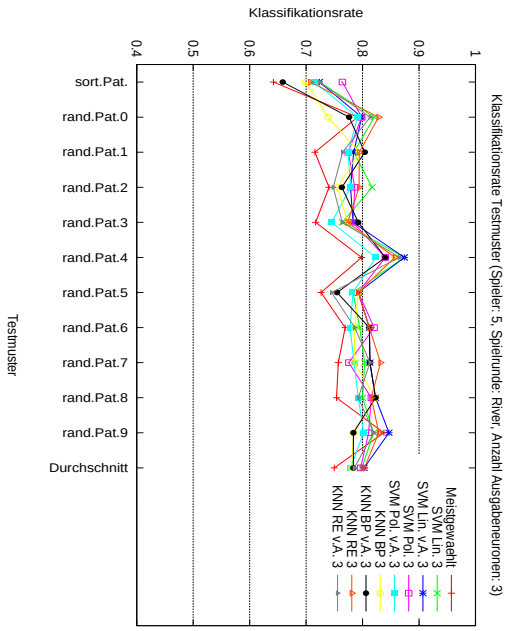

Abb. 5.26: *Klassifikationsraten der Testmustermengen, Sp. 5, River, 3 Ausgabeneuronen*

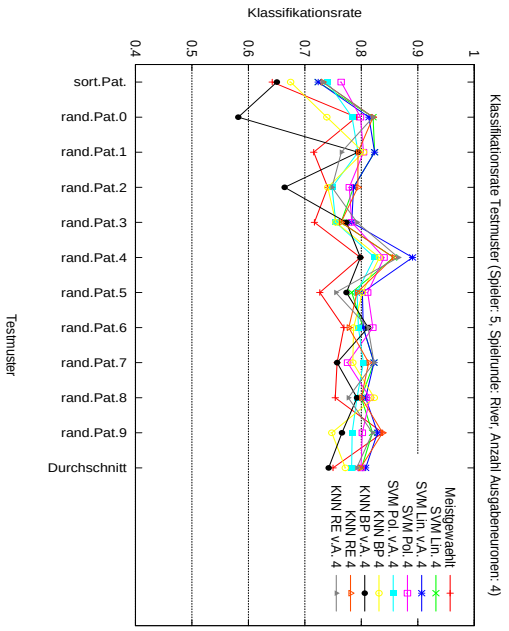

Abb. 5.27: *Klassifikationsraten der Testmustermengen, Sp. 5, River, 4 Ausgabeneuronen*

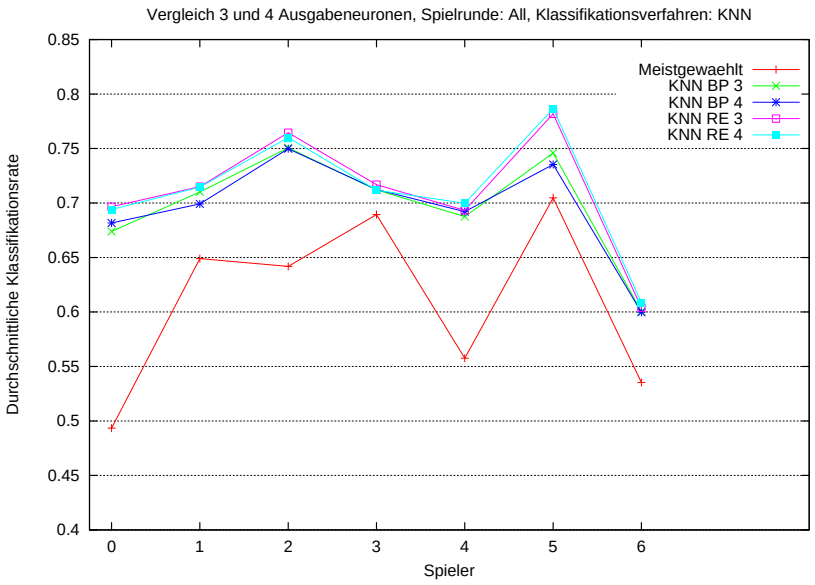

Abb. 5.28: *Vergleich 3 und 4 Ausgabeneuronen, All, KNN*

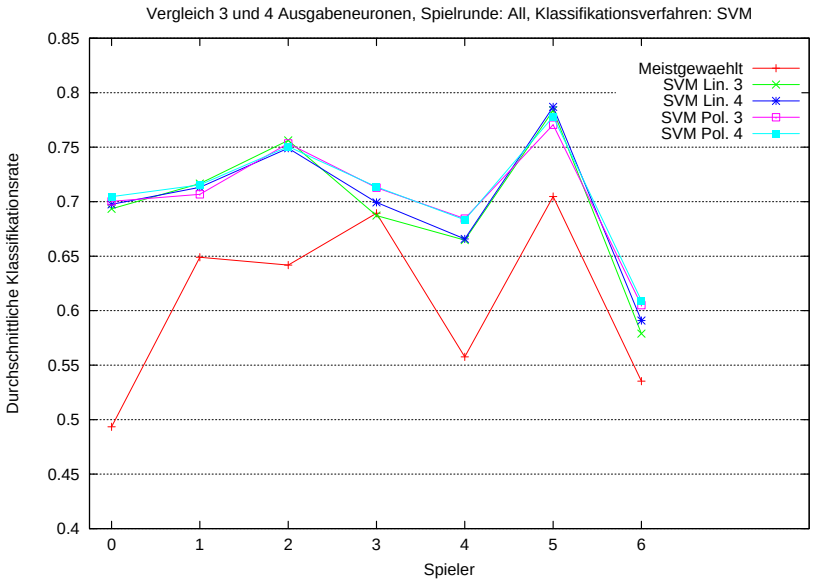

Abb. 5.29: *Vergleich 3 und 4 Ausgabeneuronen, All, SVM*

tatsächlichem Call auf. Eine bessere Trennung ist somit nicht erreicht worden. Es zeigt sich aber, dass durch die Trennung der Aktionen Check und Call zumindest eine Verbesserung der Abgrenzung zu einem Fold erreicht wird. Die Aktion Fold ist bei allen Spielern jedoch die am seltensten durchgeführte Aktion und beeinflusst die Klassifikationsrate nur zu einem geringen Anteil. Das gute Trennverhalten zwischen Check und Call hat jedoch keinen großen Einfluss auf die Klassifikationsrate, da durch die „Verschmelzung" der Aktionen Check und Call wieder annähernd das Ergebnis erreicht wird, welches auch durch die Trennung in nur drei Klassen erreicht wird. Die Lernverfahren haben jedoch eine Regel des Pokerspiels implizit herausstellen können, nämlich dass in einer Spielsituation, in der bereits geraiset worden ist, der Check keine Alternative darstellt, sondern gecallt werden muss. Zusammenfassend lässt sich sagen, dass die Trennung von Check und Call für die Abgrenzung zum Fold nützlich ist, was sich aber wegen des geringen Anteils dieser Aktion nicht stark auf die Klassifikationsrate auswirkt.

5.6.3 Trennung zwischen Spielrunden

Desweiteren wird untersucht, ob es zu einer Verbesserung der Klassifikationsrate führt, wenn für jede einzelne Spielrunde getrennt klassifiziert wird. Tabelle 5.16 gibt einen Überblick darüber, wie sich die Klassifikationsrate des All-Klassifizierers gegenüber den Klassifizierern für die einzelnen Spielrunden unterscheiden. Es sind die jeweiligen maximalen Verbesserungen und maximalen Verschlechterungen der verschiedenen Klassifikationsverfahren bei den einzelnen Spielrunden zu sehen. Die Berechnung der Verbesserungen und Verschlechterungen wird an *Spieler 2* gezeigt. Die Verbesserung von 0.0112 beim *Flop* ergibt sich aus der Differenz der Klassifikationsrate der SVM mit polynomiellem Kernel und 3 Ausgabeneuronen (Flop-Klassifizierer *Flop*: 0.8428) und (All-Klassifizierer *Flop v.A.*: 0.8316). Dagegen tritt die größte Verschlechterung von 0.0148 in der Flop-Spielrunde bei dem KNN mit Resilient-Propagation auf (0.8403-0.8255). Alle anderen Abweichungen bewegen sich innerhalb dieser maximalen Abweichungen. In der River-Spielrunde sind die größten Abweichungen von 0.0766 (0.0402+0.0374) zu sehen. Für keine der Spielrunden ist durchweg nur eine Verbesserung erfolgt und in keiner nur Verschlechterungen. In Tabelle 5.16 ist desweiteren zu erkennen, dass bei *Spieler 4* und *Spieler 5* in der Spielrunde River gar keine Verbesserung durch den Einsatz der spielrunden-spezifischen Klassifizierern erreicht wird. Dagegen sind bei *Spieler 0* und *Spieler 6* in der Spielrunde Turn nie Verschlechterungen aufgetreten. In allen anderen Spielrunden sind teilweise Verbesserungen, aber auch Verschlechterungen der Klassfikationsrate zu erkennen. Bei *Spieler 1* ist zu sehen, dass der Einsatz der spielrunden-spezifischen Klassifizierer Verschlechterung oder teilweise nur minimale Verbesserung der Klassifikationsrate gegenüber dem All-Klassifizierer ergibt.

Zur Veranschaulichung dieses Sachverhalts sind dazu in Abbildung 5.30 die Klassifikationsraten der spielrunden-spezifischen Klassifizierer und des All-Klassifizierers für *Spieler 1* gegenübergestellt. *Flop* stellt die Klassifikationsrate für die Flop-Muster durch den Flop-Klassifizierer und *Flop v.A.* durch den All-Klassifizierer dar. Hier bestätigt sich die Aussage, dass der Einsatz der spielrunden-spezifischen Klassifizierer keine Verbesserung bringt. Der All-Klassifizierer erreicht für die einzelnen Spielrunden durchweg bessere oder zumindest nicht schlechtere Klassifikationsraten als die spielrunden-spezifischen Klassifizierer. Ausnahmen sind die SVM mit linearem Kernel in der River-Spielrunde. Hier werden sowohl bei 3 als auch bei 4 Ausgabeneuronen minimal bessere Klassifikationsergebnisse durch die Unterscheidung der

Spielrunden erreicht. Im Vergleich zur Referenz-Klassifikationsrate schneiden die Klassifikationsergebnisse der SVM und KNN überall besser ab. In der Flop-Spielrunde sind diese jedoch nahezu identisch, wobei in der Turn-Spielrunde eine Verbesserung von durchschnittlich 0.05 und in der River-Spielrunde von 0.12 erreicht wird.

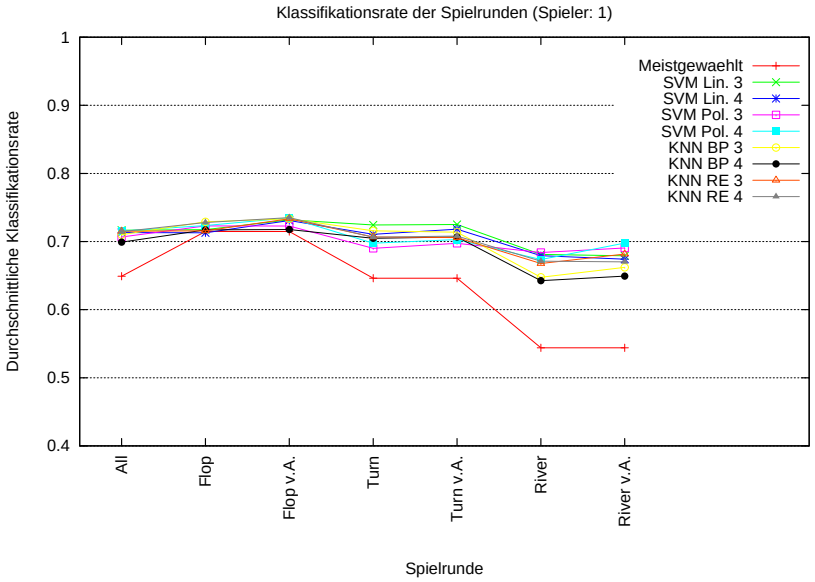

Abb. 5.30: *Klassifikationsraten der verschiedenen Spielrunden für Sp. 1: „v.A.“ zeigt an, dass der All-Klassifizierer angewendet worden ist. Ohne den Zusatz ist der spielrunden-spezifische Klassifizierer angewendet worden.*

Um die Klassifikationsraten der spielrunden-spezifischen Klassifikatoren und des All-Klassifikators direkt vergleichen zu können, sind in Abbildung 5.31 die Boxplots für die Flop-Spielrunde direkt nebeneinander dargestellt. *v.A.* zeigt an, dass der All-Klassifizierer angewendet worden ist und ohne den Zusatz ist der spielrunden-spezifische Klassifizierer angewendet worden (die Boxplots für die übrigen Spielrunden sind im Anhang A zu finden). Bei *Spieler 2* ist direkt zu erkennen, dass die Ergebnisse des spielrunden-spezifischen Klassifikators nur minimal von denen des All-Klassifikators abweichen. Die SVM mit linearem Kernel erreicht sogar genau die identische Verteilung. Bei *Spieler 0* ist eine deutliche Verschlechterung durch den Einsatz des spielrunden-spezifischen Klassifikators beim Backpropagation-Verfahren zu sehen. Dagegen haben sich sowohl bei *Spieler 4* als auch *Spieler 6* die Klassifikationsraten der SVM mit linearem Kernel durch den Einsatz des Flop-Klassifikators stark verbessert. Der große Unterschied der Extremwerte bei den KNN mit Backpropagation bei *Spieler 5*, der zuvor schon aufgefallen ist, tritt auch beim Flop-Klassifizierer auf. Für *Spieler 1* bestätigen die Boxplots die Aussage, die zur Abbildung 5.30 getroffen worden ist, dass der Einsatz des Flop-Klassifikators die Klassifikationsrate gegenüber dem All-Klassifizierer verschlechtert.

Wie in Tabelle 5.9 zu sehen ist, wird der All-Klassifizierer ungefähr zu einem Viertel von River-Mustern, einem Drittel von Turn-Mustern und zum restlichen, größten Anteil von Flop-

Tab. 5.16: *Maximale Verbesserung und Verschlechterung der Klassifikationsraten durch den Einsatz der spielrunden-spezifischen Klassifizierer im Vergleich zu den Klassifikationsraten, die der All-Klassifizierer für die Muster erreicht.*

Sp.	Spielrunde	Max. Verb.	Max. Verschl.
0	Flop	0.0042	0.0421
	Turn	0.0385	*-0.0045*
	River	0.0256	0.0217
1	Flop	0.0001	0.0180
	Turn	0.0020	0.0075
	River	0.0051	0.0242
2	Flop	0.0112	0.0148
	Turn	0.0128	0.0138
	River	**0.0402**	0.0374
3	Flop	0.0034	0.0115
	Turn	0.0154	0.0280
	River	0.0143	0.0273
4	Flop	0.0573	0.0139
	Turn	*-0.0122*	**0.0472**
	River	0.0033	0.0315
5	Flop	0.0383	0.0016
	Turn	-0.0021	0.0215
	River	0.0296	0.0073
6	Flop	0.0381	0.0033
	Turn	0.0285	-0.0030
	River	0.0310	0.0097

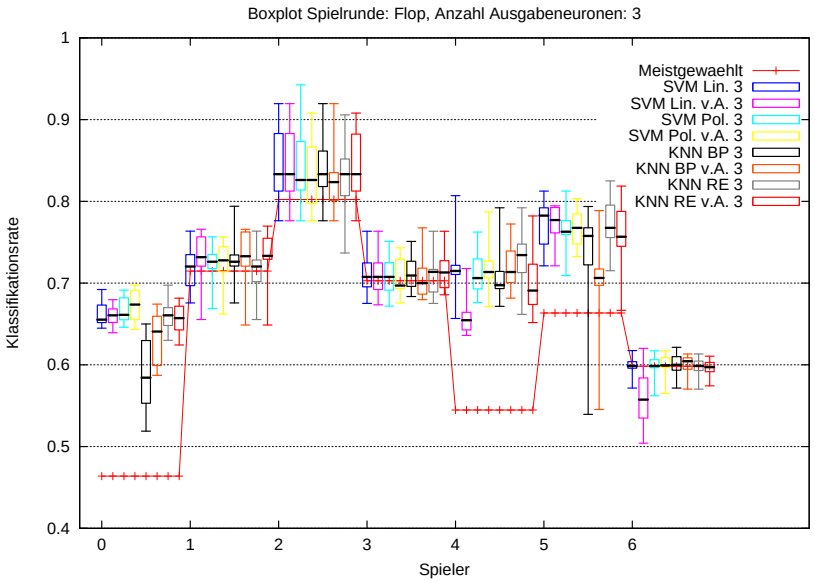

Abb. 5.31: *Boxplots der Spielrunde Flop, 3 Ausgabeneuronen: v.A. zeigt an, dass der All-Klassifizierer angewendet worden ist. Ohne den Zusatz ist der spielrunden-spezifische Klassifizierer angewendet worden.*

Mustern angelernt. Es ist jedoch nicht zu erkennen, dass der große Anteil der Flop-Muster beim Training des All-Klassifizierer die Klassifikation für die Muster der einzelnen Spielrunden stark beeinflusst. Das ist daran zu sehen, dass die Klassifikationsrate der spielrunden-spezifischen Klassifizierer teilweise besser und teilweise schlechter sind.

Die Anzahl der Epochen bei den KNN und die Parameterbestimmung für die Lernverfahren sind in den vorexperimentellen Tests nur an den All-Mustern ermittelt worden. Eventuell ist eine Verbesserung der Klassifikationsrate zu erreichen, wenn die Parameter für die Flop-, Turn- und River-Klassifizierer durch gesonderte Vortests ermittelt werden, bei denen auch nur Muster der jeweiligen Spielrunde betrachtet werden. Zusammenfassend lässt sich sagen, dass der Einsatz von spielrunden-spezifischen Klassifikatoren nur teilweise zu Verbesserungen führt. Da jedoch auch Verschlechterungen auftreten, kann nicht allgemein gesagt werden, dass spielrunden-spezifische Klassifikatoren eingesetzt werden sollen oder nicht.

5.7 Verwendbarkeit der Ergebnisse

Dieses Unterkapitel beschreibt, wie die Ergebnisse, welche die Klassifikationsverfahren liefern, einem Poker-Bot zur Gegnermodellierung nützen. Es muss jedoch immer bedacht werden, dass die Vorhersagen immer nur für die erste Aktion eines Spielers in einer Spielrunde getroffen werden. Diese Problematik ist in Abschnitt 5.2.2 angesprochen worden. Für die Verwendbarkeit der Ergebnisse wird jedoch angenommen, dass sich diese Vorhersagen auch auf die Aktionen innerhalb einer Spielrunde übertragen lassen können. Dadurch, dass pro Spiel nur höchstens

drei Muster erzeugt werden können (und das auch nur, wenn der Spieler bis zum River im Spiel bleibt), werden sehr viele Spiele benötigt, um eine geeignete Anzahl an Mustern zum Trainieren eines Spielers zu erhalten.

5.7.1 Probability-Triple

Die Ausgabe jedes Klassifikationsverfahrens ergibt die Verteilung für die einzelnen Aktionen. Diese werden so normiert, dass die Summe der Werte für Fold, Call und Raise 1 ergibt. So wird das Probability-Triple für die untersuchte Spielsituation erstellt. Dieses Triple kann nun zur Aktualisierung der Gewichtsmatrix verwendet werden, welche die Annahmen über die Hole-Cards des Gegners enthält. Zudem kann es für Zuweisung der Aktion eines Gegners bei der Simulation-Based-Betting-Strategy genutzt werden. Die Aktualisierung der Gewichtsmatrix und die Simulation-Based-Betting-Strategy sind in Unterkapitel 2.3 vorgestellt worden.

5.7.2 Meta-Entscheider

Da sich keines der Klassifikationsverfahren als „bestes“ herausgestellt hat, ist es bei der Menge an verschiedenen Klassifikationsverfahren schwierig zu entscheiden, welches Ergebnis von welchem Entscheider nun gewählt werden soll. Zudem haben die Trennung von Check und Call und der Einsatz der spielrunden-spezifischen Klassifizierer sowohl Verbesserungen als auch Verschlechterungen der Klassifikationsraten ergeben. Daher werden Meta-Entscheider eingesetzt. Ein Meta-Entscheider vereint die Vorhersagen aller getesteten Verfahren (KNN, SVM, All-Klassifizierer, spielrunden-spezifische Klassifizierer, 3 und 4 Ausgabeneuronen), indem er die Probability-Triple kombiniert, die jedes einzelne Klassifikationsverfahren für ein Muster ermittelt.

Es werden zwei verschiedene Meta-Entscheider erstellt. Der erste Meta-Entscheider, *Meta 0*, addiert jeweils die Fold-, Call- und Raise-Werte jedes Probability-Triples, welches die verschiedenen Klassifikationsverfahren für ein Muster liefern, und normiert diese wieder so, dass die Summe innerhalb eines Probability-Triples 1 ergibt. Der zweite Meta-Entscheider, *Meta 1* ermittelt den jeweils höchsten Wert für eine Aktion aus allen Probability-Triple und normiert diese. Beide Entscheider geben abschließend die Aktion als Vorhersage aus, für den der höchsten Wert im Probability-Triple steht.

Durch die Kombination der Einzelvorhersagen wird die Gesamtvorhersage robuster, da sie nicht von der Klassifikationsfähigkeit eines einzelnen Verfahrens abhängt. Eventuell kann die Klassifikationsrate sogar noch verbessert werden. Entscheider *Meta 0* orientiert sich demnach an der durchschnittlichen Entscheidung der einzelnen Verfahren, wobei der Entscheider *Meta 1* sich an den maximalen Werten der Probability-Triple der einzelnen Verfahren orientiert. In Tabelle 5.17 ist beispielhaft die Auswertung der Meta-Entscheider zu sehen. In den ersten vier Spalten sind die Werte des Probability-Triples beispielhafter Einzelverfahren zu sehen, in den letzten beiden die ermittelten der Meta-Entscheider.

Tabelle 5.18 stellt eine Übersicht der durchschnittlichen Klassifikationsraten der Meta-Entscheider im Vergleich zu dem besten Ergebnis eines Einzelverfahrens dar. Letzteres ist in der dritten Spalte *Best. Klas.* dargestellt. In den beiden hinteren Spalten sind die Klassifikationsraten der Meta-Entscheider zu sehen. Die fett-gedruckten Werte in der Tabelle zeigen an, dass

Tab. 5.17: *Beispielhafte Berechnung der Klassifikationsrate der Meta-Entscheider: Meta 0 orientiert sich an den durchschnittlichen Werten der Einzelverfahren, Meta 1 an den maximalen Werten. Die Werte sind normiert, so dass die Summe der Werte für Fold, Raise und Call 1 ergeben.*

	Verf. 1	Verf. 2	Verf. 3	Verf. 4	Meta 0	Meta 1
Fold	0.4	0.2	0.6	0.3	0.38	0.43
Call	0.5	0.5	0.3	0.5	0.45	0.36
Raise	0.1	0.3	0.1	0.2	0.18	0.21
Vorhersage	Call	Call	Fold	Call	Call	Fold

ein Meta-Verfahren einen besseren Wert als das beste Einzelverfahren erreicht hat. Es ist zu erkennen, dass die Klassifikationsrate der Meta-Entscheider sich meist nur minimal von den Ergebnissen des besten Klassifikationsverfahren unterscheiden. Dabei sind die Ergebnisse des Entscheiders *Meta 0* überwiegend besser als die des Entscheiders *Meta 1*. Entscheider *Meta 1* lässt sich eventuell durch Ausreißer einzelner Verfahren zu sehr beeinflussen, wogegen durch die Mittlung der Ergebnisse bei Entscheider *Meta 0* diese Ausreißer nicht so stark ins Gewicht fallen.

Teilweise liefern die Meta-Entscheider sogar bessere Ergebnisse als das besten Ergebnis der getesteten Klassifikatoren. Dies ist beispielsweise bei *Spieler 0* beim Turn, bei *Spieler 1* in den Spielrunden All und Flop, sowie bei *Spieler 6* in den Spielrunden All und Turn der Fall. Eine relativ hohe Negativ-Abweichung zum besten Ergebnis eines Einzelentscheiders ist bei *Spieler 3* in der Spielrunde River zu sehen (0.0295). Jedoch schwankt in dieser Spielrunde bei dem Spieler die Klassifikationsrate der Einzelverfahren stark von 0.6577 bis 0.7017 und hat damit Einfluss auf die schlechtere Klassifikationsrate (vergleiche Tabellen A.7 und A.8 im Anhang).

Abbildung 5.32 stellt die Boxplots der Meta-Entscheider denen der Einzelverfahren mit 3 Ausgabeneuronen und der Spielrunde All gegenüber. Bei den *Spielern 0* und *1* ist deutlich zu erkennen, dass die Meta-Entscheider gute Klassifikationsraten im Vergleich zu den Einzelverfahren erreichen, bei *Spieler 1* sogar bessere Ergebnisse als die Einzelverfahren ermittel werden. Auch bei den *Spielern 2*, *3* und *4* werden durch die Meta-Entscheider Klassifikationsraten erreicht, die mit den Ergebnissen der besten Einzelverfahren gleichzusetzen sind. Das für *Spieler 5* bekannte Problem mit der starken Abweichung der Extremwerte beim KNN und dem Backpropagation-Verfahren ist auch bei *Meta 1* zu sehen. Beide Meta-Entscheider ermitteln bei *Spieler 6* sogar bessere Ergebnisse als die Einzelverfahren.

Zusammenfassend lässt sich sagen, dass sich die Meta-Entscheider zur Kombination der Einzelentscheidungen eignen und dadurch gute Ergebnisse im Vergleich zu den Einzelverfahren erreicht werden. Die Abweichungen einzelner Verfahren werden durch die Meta-Entscheider meist kompensiert, was sie robuster gegenüber Ausreißern macht.

5.7.3 Weitere Verbesserungsmöglichkeit

Die Confusion-Matrix 5.8 (siehe Seite 62) ist in Tabelle 5.19 in relativer Form dargestellt. Die Werte innerhalb der Matrix summieren sich zu 1. In der letzten Spalte bzw. Zeile ist die

Tab. 5.18: *Durchschnittliche Klassifikationsraten der Meta-Entscheider: Bessere Ergebnisse eines Meta-Verfahrens im Vergleich zu einem Einzelverfahren sind fett dargestellt.*

Sp.	Spielrunde	Best. Klas.	Meta 0	Meta 1
0	All	0.7046	0.7013	0.6998
	Flop	0.6747	0.6713	0.6635
	Turn	0.7566	**0.7589**	**0.7571**
	River	0.7091	0.6985	0.6967
1	All	0.7166	**0.7196**	**0.7195**
	Flop	0.7351	**0.7376**	**0.7394**
	Turn	0.7250	0.7214	0.7231
	River	0.6978	0.6875	0.6857
2	All	0.7645	0.7607	0.7617
	Flop	0.8428	0.8404	0.8365
	Turn	0.8046	0.8008	0.7946
	River	0.6179	0.6148	0.6013
3	All	0.7168	0.7125	0.7153
	Flop	0.7179	0.7171	0.7144
	Turn	0.7317	0.7283	0.7302
	River	0.7017	0.6896	0.6763
4	All	0.6998	0.6901	0.6848
	Flop	0.7334	0.7222	0.7039
	Turn	0.6842	0.6714	0.6759
	River	0.6656	0.6619	0.6613
5	All	0.7870	0.7806	0.7787
	Flop	0.7804	0.7707	0.7627
	Turn	0.7974	0.7903	0.7841
	River	0.8077	0.7934	0.7948
6	All	0.6092	**0.6097**	**0.6092**
	Flop	0.6010	0.5996	0.5919
	Turn	0.6457	**0.6470**	**0.6466**
	River	0.6125	0.6006	0.5996

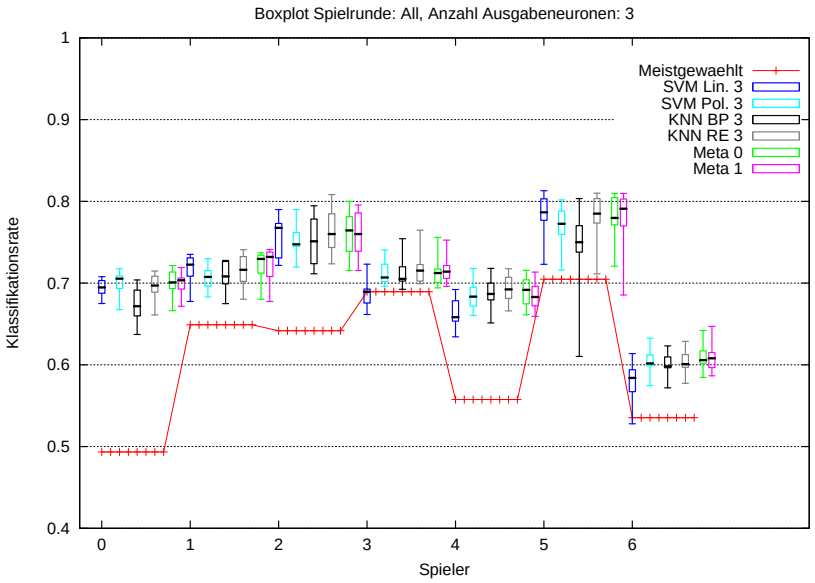

Abb. 5.32: *Boxplots der Spielrunde All mit Meta-Entscheider, 3 Ausgabeneuronen*

Frequenz der durchgeführten bzw. vorhergesagten Aktion abzulesen. Tabelle 5.20 stellt die Verlässlichkeit einer vorhergesagten Aktion dar. Hieran ist die Art des Fehlers zu erkennen, die das Klassifikationsverfahren macht. Beispielsweise ist bei einem vorhergesagten Fold die durchgeführte Aktion zu $\frac{0.087}{0.165} \approx 0.526 = 52.6\%$ auch tatsächlich ein Fold, jedoch zu 35.1% ist sie ein Call und zu 12.3% ein Raise.

Tab. 5.19: *Beispielhafte Confusion-Matrix in relativer Form: Die Werte innerhalb der Matrix summieren sich zu 1.*

	vorherg. Fold	vorherg. Call	vorherg. Raise	Freq.
Fold	0.087	0.011	0.006	0.104
Call	0.058	0.339	0.073	0.470
Raise	0.020	0.157	0.249	0.426
Freq.	0.165	0.507	0.328	0.675

Davidson [Dav02] schlägt eine Erweiterung vor, welche die Verlässlichkeit einer voherhergesagten Aktion mit in die endgültige Berechnung des Probability-Triples einfließen lässt. Angenommen, das Verfahren, welches die Confusion-Matrix 5.19 liefert, ermittelt die Ausgabe $(0.5, 0.3, 0.2)$ für eine Spielsituation. Wie in Tabelle 5.20 abzulesen, ist die Verlässlichkeit der Vorhersage $(0.526, 0.668, 0.761)$, also ein vorhergesagter Fold ist zu ca. 53% ein Fold, ein vorhergesagter Call zu ca. 67% ein Call und ein vorhergesagtes Raise zu ca. 76% ein Raise. Das Skalarprodukt dieser beiden Tupel ergibt $(0.5 \cdot 0.526, 0.3 \cdot 0.668, 0.2 \cdot 0.761) \approx (0.26, 0.2, 0.15)$ und führt normiert zum Probability-Triple $(0.43, 0.33, 0.24)$, welches der Aktion Fold ent-

Tab. 5.20: *Verlässlichkeit einer vorhergesagten Aktion: Ein vorhergesagter Fold ist zu 0.526 tatsächlich ein Fold, zu 0.351 ein Call und zu 0.123 ein Raise.*

	vorherg. Fold	vorherg. Call	vorherg. Raise
Fold	0.526	0.023	0.018
Call	0.351	0.668	0.221
Raise	0.123	0.309	0.761
Summe	1.000	1.000	1.000

spricht. Möglicherweise lässt sich die Klassifikationsrate der einzelnen Verfahren und der Meta-Entscheider durch die erweiterte Berechnung des Probability-Triples noch verbessern.

5.7.4 Anpassung des Gegnermodells

Die durchgeführten Tests stellen die Situation bis zu einem bestimmten Zeitpunkt in einem Pokerspiel dar, bis zu dem die Muster bekannt sind. Bis dahin ist ein Gegnermodell anhand der Spielweise und der durchgeführten Aktionen des Gegners erstellt worden. Es kann von einem Poker-Bot, beispielsweise Poki, in der in Unterkapitel 4.2 beschriebenen Art und Weise genutzt werden. Dieses Modell muss nun während eines realen Spiels ständig aktualisiert werden. Dazu müssen die KNN mit den aus den aktuellen Spielsituationen gewonnenen Informationen, also den neuen Mustern, nachtrainiert werden. Auch die Berechnungen der SVM für die optimale trennende Hyperebene müssen ständig aktualisiert werden.

5.8 Mögliche Probleme

In diesem Unterkapitel werden mögliche Ursachen genannt, welche bessere Klassifikationsraten verhindert haben könnten. Das Hauptproblem ist die Unvollständigkeit der Information, dass die Hole-Cards der Gegner und die Community-Cards, die während des Spiels aufgedeckt werden, nicht bekannt sind. Ein Spieler kann demnach bei identischer Spielsituation, die durch die Muster beschrieben wird, unterschiedlich reagieren, je nachdem welche Hand er aus seinen Hole-Cards und den Community-Cards bilden kann. Desweiteren fehlen in der Datenbasis Informationen, die zur besseren Spielsituationsbeschreibung und damit zur eventuell besseren Klassifikation nützlich wären. Es ist auch nicht zu gewährleisten, dass die verwendete Software zur Klassifikation, SNNS und RapidMiner, immer korrekt arbeitet. Desweiteren könnte die Auswahl der Merkmale zur Beschreibung der Spielsituation suboptimal sein. Durch das Hinzufügen weiterer oder die Entfernung einiger Merkmale kann eventuell eine bessere Klassifikation erreicht werden. Bei der Extraktion der Merkmale aus der Datenbasis können ebenfalls Fehler aufgetreten sein. Ein weiteres Problem kann sich dadurch ergeben haben, dass ein allgemeines Modell für alle Pokerspieler erstellt worden ist. Für jeden Spieler werden die Klassifikationsverfahren mit den gleichen Einstellungen durchgeführt, die in Vortests ermittelt worden sind. Die Parametereinstellungen sind experimentell an einer Menge von Spielern ermittelt worden. Diese Parameter können für einige Spieler suboptimal sein.

5.9 Zusammenfassung

Dieses Kapitel hat einen Überblick über den Aufbau und die Durchführung der Tests mit den Klassifikationsverfahren KNN und SVM gegeben. Die Bestimmung der Parameter der Lernverfahren ist beschrieben worden. Daran hat sich die Darstellung der Ergebnisse und die Beobachtungen dazu angeschlossen. Abschließend sind die Verwendbarkeit der Ergebnisse, mögliche Erweiterungen und auftretende Probleme aufgezeigt worden.

6 Fazit

In diesem Kapitel wird eine Zusammenfassung der vorliegenden Arbeit gegeben. Dabei werden die Ergebnisse und Schlussfolgerungen dargestellt, die aus dieser Arbeit hervorgegangen sind. Abschließend wird in einem Ausblick aufgezeigt, wo Potential zur Erweiterung, Veränderung und Verbesserung zu finden ist.

6.1 Zusammenfassung

In der vorliegenden Arbeit sind zwei Klassifikationsverfahren getestet worden, die künstlichen neuronalen Netze und die Support-Vector-Machines. Diese beiden Verfahren werden wegen ihrer hohen Leistungsfähigkeit häufig zur Lösung von Klassifikationsproblemen eingesetzt. Beim Pokerspiel besteht ein solches Problem darin, zwischen den Spielaktionen eines Spielers zu unterscheiden, um ein Modell für das Bietverhalten eines Spielers zu erstellen. Dies hat sich als schwieriges Problem erwiesen. Es hat sich jedoch gezeigt, dass die KNN und SVM eine gute Klassifikation erreichen. Diese Ergebnisse können von Poker-Bots genutzt werden, um sich ein Gegnermodell zu erstellen und die eigene Bietstrategie daran auszurichten.

Zu Beginn ist in das Pokerspiel eingeführt und die Regeln, Spielabläufe und Ansätze zur Wahl der Bietstrategie sind beschrieben worden. Dabei ist die Wichtigkeit der Gegnermodellierung herausgestellt worden. Daran hat sich die Beschreibung der verwendeten Klassifikationsverfahren angeschlossen. Hierbei sind die Grundlagen ausführlich erklärt und mögliche auftretende Probleme und deren Lösungen erläutert worden. Weiterhin ist ein Einblick in andere wissenschaftliche Arbeiten zum Thema Poker gegeben und aufgezeigt worden, wie sich diese von der vorliegenden Arbeit abgrenzen.

Für die durchgeführten Tests sind bei den KNN als Lernverfahren der Backpropagation-Algorithmus und der Resilient-Propagation-Algorithmus eingesetzt worden. Der Backpropagation-Algorithmus ist das Standardverfahren für KNN und Resilient-Propagation eine Variante dessen. Beide Verfahren haben annähernd gleich gute Ergebnisse geliefert. Bei den SVM wird die Trennung zwischen den Spielsituationen mit so genannten Kernel vollzogen. In dieser Arbeit sind ein linearer und ein polynomieller Kernel zum Einsatz gekommen. Die Klassifikationsergebnisse der SVM mit beiden Kernel sind vergleichbar zu denen der KNN ausgefallen, ohne dass sich ein Verfahren als durchweg bestes herausstellen lässt. Die Ergebnisse der Klassifikationsverfahren sind mit denen eines simplen Klassifikators verglichen worden, der immer die Aktion vorhersagt, die ein Spieler in einer Spielrunde am häufigsten wählt. Im Vergleich zu diesen Ergebnissen haben die getesteten Klassifikationsverfahren stets besser abgeschnitten und teilweise Verbesserungen der Klassifikationsrate von 0.2 erreicht. Die durchschnittliche Klassifikationsrate, die bei allen Spielern in den jeweiligen Spielrunden erreicht worden ist, liegt bei 0.72. Die für einen Spieler beste Klassifikationsrate ist mit 0.84 ermittelt worden.

Die Aktionen Check und Call bezeichnen das „Mitgehen“ beim Pokerspiel, also der Verbleib

im Spiel. Der Unterschied zwischen den Aktionen besteht darin, dass bei einem Call noch die Differenz zum bisher höchsten Einsatz eines Spielers am Tisch nachbezahlt werden muss. Beim Check hingegen muss nicht nachgezahlt werden, da der eigene Einsatz bereits der höchste am Tisch ist. Gemeinhin wird das „Mitgehen“ jedoch als eine Aktion betrachtet, da sie nicht wie das Raise die übrigen Spieler in Aktionszwang bringt. In der vorliegenden Arbeit ist sowohl die Betrachtung von Check und Call als zwei unterschiedliche Aktionen also auch die Zusammenfassung zu einer Aktion untersucht worden. Dabei hat sich herausgestellt, dass die Klassifikationsverfahren durch die Trennung von Check und Call eine genauere Unterscheidung der Aktion Fold vom Check erzielen. Außerdem ist die Aktion Check nahezu nie als Call klassifiziert worden. Das gilt auch für den umgekehrten Fall. Dies macht insofern Sinn, dass in einer gegebenen Spielsituation jeweils nur eine der beiden Aktionen möglich ist. Das Klassifikationsverfahren beachtet diese Regel also implizit, auch ohne entsprechendes explizites Vorwissen.

Beim Pokerspiel gibt es die Spielrunden Flop, Turn und River. Die Strategie zur Aktionsauswahl eines Spielers kann sich in jeder Spielrunde unterscheiden. Daher ist untersucht worden, ob es zu einer Verbesserung der Klassifikationsrate führt, wenn die Spielrunden seperat klassifiziert werden. Dem gegenüber steht der Klassifikator, der alle Spielrunden zusammen betrachtet. Es ist beobachtet worden, dass durch die spielrunden-spezifischen Klassifikatoren teilweise bessere Klassifikationsraten erreicht werden als durch den Klassifikator, der alle Spielrunden zusammen betrachtet. Jedoch sind dadurch auch Verschlechterungen aufgetreten, so dass nicht allgemein gesagt werden kann, dass der Einsatz der Klassifikatoren für die einzelnen Spielrunden lohnenswert ist.

Da keines der Verfahren durchgängig bessere Ergebnisse als die übrigen Verfahren ermittelt hat, werden die Ergebnisse der einzelnen Verfahren kombiniert. Dazu sind so genannte Meta-Entscheider eingesetzt worden. Es hat sich gezeigt, dass die Ergebnisse der Meta-Entscheider zumeist dem Ergebnis des besten Einzelverfahrens entspricht und teilweise sogar eine bessere Klassifikationsrate erreicht wird.

6.2 Ausblick

Es gibt weitere Möglichkeiten, die Aktionsvorhersage eines Spielers zu verbessern, die jedoch durch den begrenzten Zeitrahmen dieser Arbeit nicht untersucht worden sind. Verbesserung kann durch die in Abschnitt 5.7.3 vorgestellte Erweiterung durch die Hinzunahme der Verlässlichkeit der einzelnen Klassifikationsverfahren zur Berechnung der Probability-Triple erreicht werden. Zudem kann das in Unterkapitel 5.8 angesprochene Verändern der Muster, also das Hinzufügen weiterer Merkmale und das Entfernen anderer Merkmale, eventuell die Klassifikationsraten verbessern.

Da ein Pokerspieler seine Spielweise über die Zeit gesehen verändert, wäre eine andere mögliche Erweiterung, dass weitere KNN bzw. SVM erstellt werden, die den zeitlichen Aspekt mit berücksichtigen. Dabei wird die Klassifikation nicht auf allen Mustern durchgeführt, sondern auf Mustern eines kurzfristigen, eines mittelfristigen und eines langfristigen Zeitraums. Desweiteren kann ein allgemeines Gegnermodell erstellt werden, welches an vielen verschiedenen Spielern lernt. Mit einem solchen Modell können auch für einen neuen Spieler am

Tisch, über den bisher nur wenige oder gar keine Informationen und Daten vorliegen, Vorhersagen getroffen werden, welche Aktion er ausführen wird. Auch für bestimmte Spielertypen, also solche, die sich in ihrer Spielaggressivität und Neugierde unterscheiden (vergleiche Unterkapitel 2.3), können Modelle entwickelt werden, die eine Vorhersage der Aktion durchführen.

Eine weitere Möglichkeit ist die Verwendung anderer Klassifikationsverfahren, wie beispielsweise Entscheidungsbäumen oder Verfahren wie Random-Forest oder Boosting. Die Vorhersagen der einzelnen Verfahren können noch gewichtet werden, so dass die Vorhersage eines in der Vergangenheit häufig korrekten Verfahrens stärker in die Entscheidungsfindung einfließt.

Es muss auch immer beachtet werden, dass Poker ein Echtzeitspiel ist, in dem einem Spieler nur eine begrenzte Zeitspanne zur Verfügung steht, in der die eigene Aktion gewählt werden muss. Das schränkt die Wahl und Durchführung einzelner Verfahren aufgrund ihrer hohen Laufzeit ein. Die Modelle müssen mit den neuen Informationen, die jeder Spielzug eines Spielers preisgibt, nachgelernt werden.

Insgesamt hat diese Arbeit gezeigt, dass sich künstliche neuronale Netze und Support-Vector-Machines zur Gegnermodellierung beim Pokerspiel eignen. Auch wenn keine einzelne Variante der Verfahren völlig verlässliche Voraussagen liefern kann, erreichen manche dieser und speziell auch deren Kombination durch Meta-Entscheider überraschend gute Klassifikationsraten für die Vorhersage der Aktion des menschlichen Gegners. Zusätzlich dazu sind Verbesserungen durch die genannten Erweiterungsmöglichkeiten zu erwarten.

Literaturverzeichnis

[Ach07] Achenbach, Florian: *NL-Hold'em Poker*. Heel, 2007

[BBD+03] Billings, Darse ; Burch, Neil ; Davidson, Aaron ; Holte, Robert ; Schaeffer, Jonathan ; Schauenberg, Terence C. ; Szafron, Duane: Approximating game-theoretic optimal strategies for full-scale poker. In: *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, S. 661–668

[BDS02] Billings, Darse ; Davidson, Aaron ; Szafron, Duane: The challenge of poker. In: *Artificial Intelligence* 134 (2002), Nr. 1–2, S. 201–240

[Bra97] Braun, Heinrich: *Neuronale Netze: Optimierung durch Lernen und Evolution*. Heidelberg : Springer, 1997

[Car03] Caro, Mike: *Caro's Book Of Poker Tells*. Cordoza Publishing, 2003

[CL01] Chang, Chih-Chung ; Lin, Chih-Jen: *LIBSVM: a library for support vector machines*, 2001. – Software verfügbar unter: `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, zuletzt gesichtet: 12.10.2008

[Dav99] Davidson, Aaron: CMPUT 499 – Research Project Review: Using Artifical Neural Networks to Model Opponents in Texas Hold'em. 1999. – Forschungsbericht

[Dav02] Davidson, Aaron: *Opponent modeling in poker: Learning and acting in a hostile environment*. 2002. – Masterthesis

[DHS00] Duda, Richard O. ; Hart, Peter E. ; Stork, David G.: *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000

[Fah88] Fahlman, Scott E.: An empirical study of learning speed in back-propagation networks. 1988. – Forschungsbericht

[Gre06] Greenstein, Barry: *Ace On The River – An Advanced Poker Guide*. Last Knight Publishing Company, 2006

[Heb49] Hebb, Donald O.: *The organization of behavior*. New York : Wiley, 1949

[IBM97] IBM: *Deep Blue*. 1997. – `http://www.research.ibm.com/deepblue/home/html/b.shtml`, zuletzt gesichtet: 15.08.2008

[jav] *Java*. – Software verfügbar unter: `http://java.com/de/`, zuletzt gesichtet: 12.10.2008

[KP97] Koller, Daphne ; Pfeffer, Avi: Representations and Solutions for Game-Theoretic Problems. In: *Artificial Intelligence* 94 (1997), S. 167–215

[MP43] Mcculloch, Warren ; Pitts, Walter: A logical calculus of the ideas immanent in nervous activity. In: *Bulletin of Mathematical Biology* 5 (1943), December, Nr. 4, S. 115–133

[MP69] Minsky, Marvin L. ; Papert, Seymour: *Perceptrons*. MIT Press, 1969

[MWK+06] Mierswa, Ingo ; Wurst, Michael ; Klinkenberg, Ralf ; Scholz, Martin ; Euler, Timm: YALE: Rapid Prototyping for Complex Data Mining Tasks. In: Ungar, Lyle (Hrsg.) ; Craven, Mark (Hrsg.) ; Gunopulos, Dimitrios (Hrsg.) ; Eliassi-Rad, Tina (Hrsg.): *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, August 2006, 935–940

[mys] *MySQL*. – Software verfügbar unter: `http://www.mysql.de/`, zuletzt gesichtet: 12.10.2008

[NKK94] Nauck, Detlef ; Klawonn, Frank ; Kruse, Rudolf: *Neuronale Netze und Fuzzy-Systeme*. Wiesbaden : Vieweg, 1994

[pok] *Michael Maurer's IRC Poker Database*. – `http://games.cs.ualberta.ca/poker/IRC/`, zuletzt gesichtet: 12.10.2008

[rap] *RapidMiner*. – Software verfügbar unter: `http://rapid-i.com/`, zuletzt gesichtet: 12.10.2008

[RHW86] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning internal representations by error propagation. (1986), S. 318–362

[Sch06] Schauenberg, Terence C.: *Opponent modelling and search in poker*. 2006. – Masterthesis

[Sch07] Schilling, Frieder: Mit ein paar Karten zum großen Geld. In: *SPIEGEL ONLINE* (2007). – `http://www.spiegel.de/sport/sonst/0,1518,492529,00.html`, zuletzt gesichtet: 17.10.2008

[Sch08] Schulz, Norbert: Skript zur Vorlesung Spieltheorie / Universität Würzburg. 2008. – Forschungsbericht

[Sim72] Simmons, George F.: *Differential Equations With Applications and Historical Notes*. McGraw-Hill, 1972

[Skl99] Sklansky, David: *Hold'em Poker: For Advanced Players*. Two Plus Two Pub, 1999

[Skl04] Sklansky, David: *The Theory Of Poker*. Two Plus Two Pub, 2004

[SNN] *SNNS – Stuttgart Neural Network Simulator*. – Software verfügbar unter: `http://www.ra.cs.uni-tuebingen.de/SNNS/`, zuletzt gesichtet: 12.10.2008

[SS01] Scholkopf, Bernhard ; Smola, Alexander J.: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA, USA : MIT Press, 2001

[UAC] *The University of Alberta Computer Poker Research Group.* – `http://poker.cs.ualberta.ca/`, zuletzt gesichtet: 12.10.2008

[Wer88] Werbos, Paul J.: Backpropagation: past and future. In: *Neural Networks, 1988., IEEE International Conference on* (1988), Jul, S. 343–353 vol.1

Abbildungsverzeichnis

Tabellenverzeichnis

A Ergebnistabellen und Plots der Tests

Beschreibung zum Aufbau der Tabellen und der Plots sind in Unterkapitel 5.6 (siehe Seite 60) nachzulesen.

Tab. A.1: *Durchschnittliche Klassifikationsraten, Sp. 0, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.6739	0.6817	0.6966	0.6940	0.4935
Flop	*0.5895*	0.6102	0.6606	0.6591	0.4639
Flop v.A.	0.6316	0.6386	0.6563	0.6551	0.4639
Turn	0.7533	0.7498	**0.7562**	0.7487	0.5332
Turn v.A.	0.7148	0.7154	0.7488	0.7393	0.5332
River	0.6915	0.6829	0.7081	0.7079	0.5255
River v.A.	0.6907	0.7046	0.6962	0.7002	0.5255

Tab. A.2: *Durchschnittliche Klassifikationsraten, Sp. 0, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.6935	0.6975	0.7003	0.7046	0.4935
Flop	0.6620	0.6700	0.6674	0.6727	0.4639
Flop v.A.	*0.6595*	0.6686	0.6718	0.6747	0.4639
Turn	0.7553	**0.7566**	0.7532	0.7526	0.5332
Turn v.A.	0.7492	0.7352	0.7415	0.7481	0.5332
River	0.7091	0.7076	0.6844	0.6847	0.5255
River v.A.	0.6835	0.7031	0.6999	0.7039	0.5255

Tab. A.3: *Durchschnittliche Klassifikationsraten, Sp. 1, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.7101	0.6991	0.7150	0.7146	0.6490
Flop	0.7290	0.7171	0.7158	0.7279	0.7147
Flop v.A.	0.7335	0.7176	**0.7334**	**0.7351**	0.7147
Turn	0.7158	0.7048	0.7067	0.7063	0.6462
Turn v.A.	0.7138	0.7067	0.7058	0.7081	0.6462
River	0.6476	*0.6427*	0.6678	0.6712	0.5439
River v.A.	0.6622	0.6492	0.6816	0.6701	0.5439

Tab. A.4: *Durchschnittliche Klassifikationsraten, Sp. 1, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.7166	0.7132	0.7068	0.7154	0.6490
Flop	0.7175	0.7128	0.7229	0.7236	0.7147
Flop v.A.	0.7318	0.7308	0.7229	**0.7347**	0.7147
Turn	0.7243	0.7107	0.6901	0.6976	0.6462
Turn v.A.	0.7250	0.7182	0.6975	0.7028	0.6462
River	0.6811	0.6792	0.6839	*0.6736*	0.5439
River v.A.	0.6791	0.6741	0.6905	0.6978	0.5439

Tab. A.5: *Durchschnittliche Klassifikationsraten, Sp. 2, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.7507	0.7499	0.7645	0.7601	0.6418
Flop	0.8384	0.8394	0.8255	0.8338	0.8023
Flop v.A.	0.8278	0.8347	**0.8403**	0.8403	0.8023
Turn	0.8012	0.7948	0.7810	0.7808	0.6872
Turn v.A.	0.7883	0.7865	0.7948	0.7844	0.6872
River	0.6179	0.6172	0.6098	0.6029	0.5281
River v.A.	0.5933	*0.5823*	0.6164	0.6066	0.5281

Tab. A.6: *Durchschnittliche Klassifikationsraten, Sp. 2, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.7562	0.7490	0.7535	0.7505	0.6418
Flop	0.8424	0.8404	**0.8428**	0.8364	0.8023
Flop v.A.	0.8424	0.8387	0.8316	0.8356	0.8023
Turn	0.8019	0.7870	0.7851	0.7692	0.6872
Turn v.A.	0.8046	0.7856	0.7754	0.7603	0.6872
River	0.6156	0.5864	0.5780	0.5903	0.5281
River v.A.	*0.5754*	0.5769	0.6154	0.6153	0.5281

Tab. A.7: *Durchschnittliche Klassifikationsraten, Sp. 3, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.7120	0.7125	0.7168	0.7118	0.6894
Flop	0.7119	0.7063	0.7085	0.7125	0.7029
Flop v.A.	0.7086	0.7179	0.7143	0.7141	0.7029
Turn	0.7190	0.7267	0.7251	0.7218	0.6966
Turn v.A.	0.7234	0.7224	**0.7317**	0.7174	0.6966
River	*0.6675*	0.6808	0.6828	0.7008	0.6586
River v.A.	0.6948	0.6944	0.6947	0.6865	0.6586

Tab. A.8: *Durchschnittliche Klassifikationsraten, Sp. 3, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.6873	0.6994	0.7128	0.7136	0.6894
Flop	0.7117	0.7144	0.7083	0.7073	0.7029
Flop v.A.	0.7097	0.7174	0.7094	0.7075	0.7029
Turn	0.7009	0.7082	0.7093	0.7021	0.6966
Turn v.A.	0.6855	0.7009	**0.7315**	0.7301	0.6966
River	0.6577	0.6785	0.6755	0.6822	0.6586
River v.A.	*0.6533*	0.6684	0.6939	0.7017	0.6586

Tab. A.9: *Durchschnittliche Klassifikationsraten, Sp. 4, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.6875	0.6921	0.6933	0.6998	0.5576
Flop	0.7064	0.7302	0.7289	0.7334	0.5448
Flop v.A.	0.7203	0.7266	0.7009	0.7297	0.5448
Turn	0.6609	0.6618	0.6608	0.6613	0.5371
Turn v.A.	0.6731	0.6757	0.6808	0.6842	0.5371
River	0.6370	*0.6294*	0.6369	0.6299	0.6037
River v.A.	0.6364	0.6376	0.6593	0.6614	0.6037

Tab. A.10: *Durchschnittliche Klassifikationsraten, Sp. 4, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.6648	0.6658	0.6845	0.6834	0.5576
Flop	0.7163	**0.7184**	0.7099	0.7041	0.5448
Flop v.A.	0.6590	0.6685	0.7171	0.7149	0.5448
Turn	0.6585	0.6607	0.6333	0.6216	0.5371
Turn v.A.	0.6709	0.6743	0.6700	0.6689	0.5371
River	0.6619	0.6526	0.6306	*0.6203*	0.6037
River v.A.	0.6656	0.6493	0.6481	0.6489	0.6037

Tab. A.11: *Durchschnittliche Klassifikationsraten, Sp. 5, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.7459	0.7354	0.7819	0.7865	0.7047
Flop	0.7343	0.7333	0.7703	0.7804	0.6635
Flop v.A.	0.6975	*0.6950*	0.7622	0.7690	0.6635
Turn	0.7429	0.7589	0.7873	0.7884	0.7184
Turn v.A.	0.7644	0.7636	0.7922	0.7974	0.7184
River	0.7808	0.7717	**0.8032**	0.7985	0.7499
River v.A.	0.7838	0.7421	0.7851	0.7918	0.7499

Tab. A.12: *Durchschnittliche Klassifikationsraten, Sp. 5, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.7838	0.7870	0.7706	0.7780	0.7047
Flop	0.7707	0.7794	*0.7662*	0.7715	0.6635
Flop v.A.	0.7719	0.7698	0.7666	0.7731	0.6635
Turn	0.7687	0.7770	0.7671	0.7668	0.7184
Turn v.A.	0.7829	0.7912	0.7693	0.7802	0.7184
River	0.7986	0.8004	0.7965	0.7994	0.7499
River v.A.	0.8027	**0.8077**	0.7789	0.7828	0.7499

Tab. A.13: *Durchschnittliche Klassifikationsraten, Sp. 6, KNN*

Spielrunde	Backprop. 3	Backprop. 4	R-Prop. 3	R-Prop. 4	Referenz
All	0.6004	0.5997	0.6039	0.6086	0.5353
Flop	0.6003	0.5926	0.5974	0.5977	0.5983
Flop v.A.	0.5995	0.5959	0.5964	0.5974	0.5983
Turn	0.6429	0.6296	**0.6452**	0.6450	0.5466
Turn v.A.	0.6144	0.6179	0.6217	0.6320	0.5466
River	0.5748	0.5922	0.5900	0.5919	0.4876
River v.A.	*0.5721*	0.5727	0.5779	0.5868	0.4876

Tab. A.14: *Durchschnittliche Klassifikationsraten, Sp. 6, SVM*

Spielrunde	Lin. Kernel 3	Lin. Kernel 4	Pol. Kernel 3	Pol. Kernel 4	Referenz
All	0.5789	0.5910	0.6051	0.6092	0.5353
Flop	0.5983	0.5983	0.5985	0.5982	0.5983
Flop v.A.	0.5601	0.5605	0.5988	0.6010	0.5983
Turn	0.6441	0.6398	**0.6457**	0.6383	0.5466
Turn v.A.	0.6198	0.6368	0.6299	0.6261	0.5466
River	0.5848	0.6125	0.5822	0.5915	0.4876
River v.A.	*0.5577*	0.5816	0.5850	0.6012	0.4876

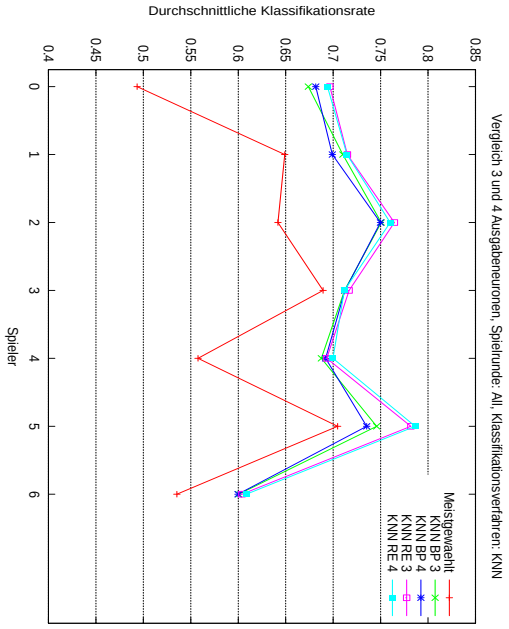

Abb. A.1: *Vergleich 3 und 4 Ausgabeneuronen, All, KNN*

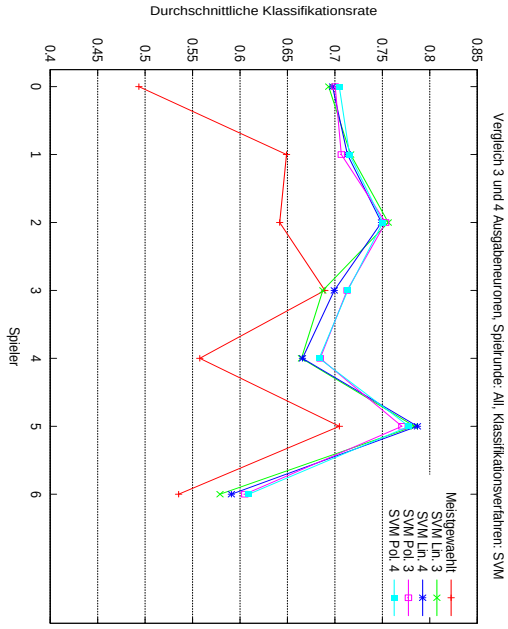

Abb. A.2: *Vergleich 3 und 4 Ausgabeneuronen, All, SVM*

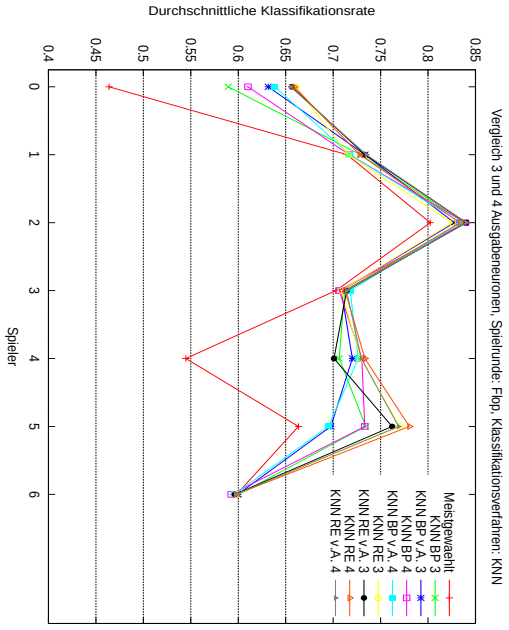

Abb. A.3: *Vergleich 3 und 4 Ausgabeneuronen, Flop, KNN*

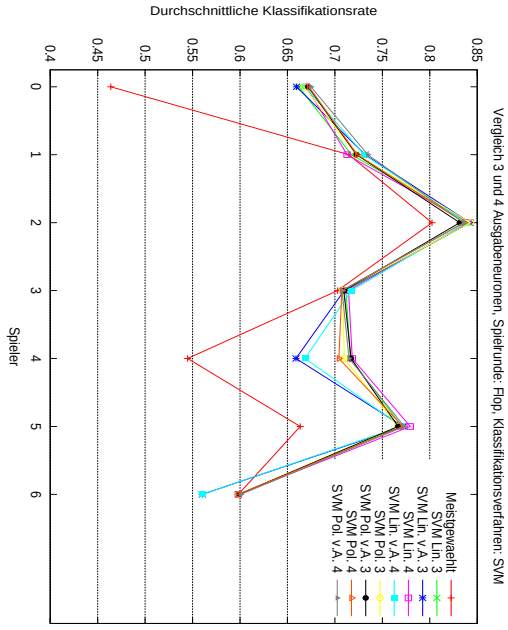

Abb. A.4: *Vergleich 3 und 4 Ausgabeneuronen, Flop, SVM*

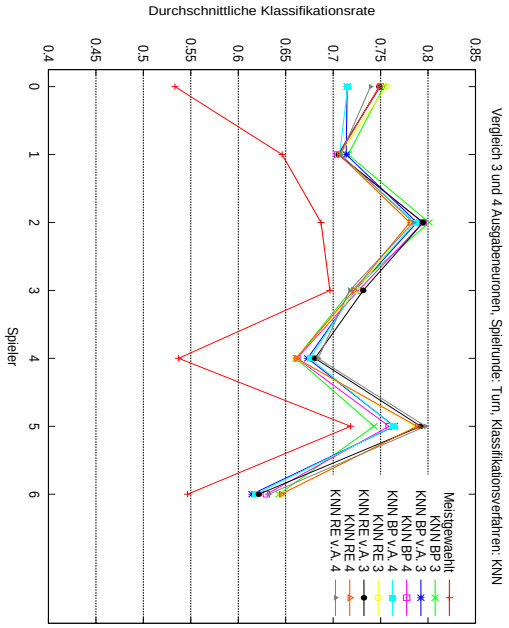

Abb. A.5: *Vergleich 3 und 4 Ausgabeneuronen, Turn, KNN*

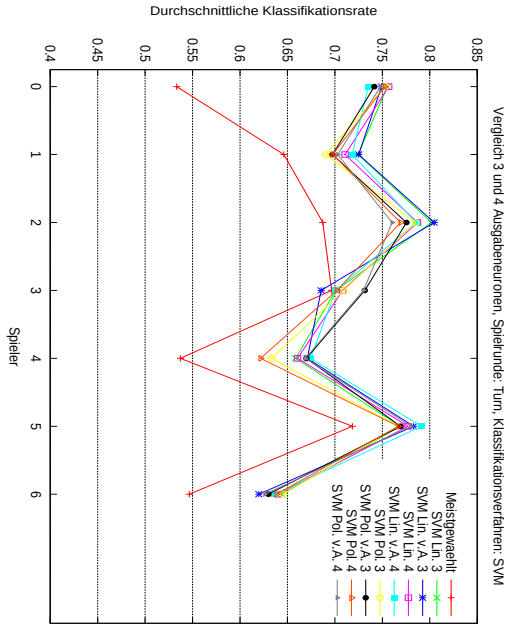

Abb. A.6: *Vergleich 3 und 4 Ausgabeneuronen, Turn, SVM*

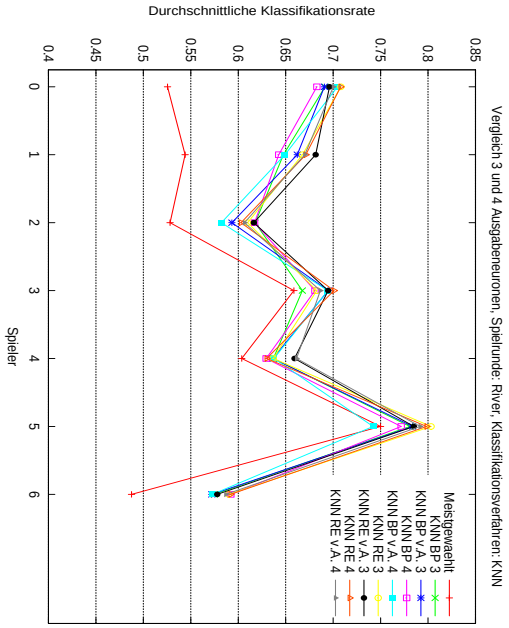

Abb. A.7: *Vergleich 3 und 4 Ausgabeneuronen, River, KNN*

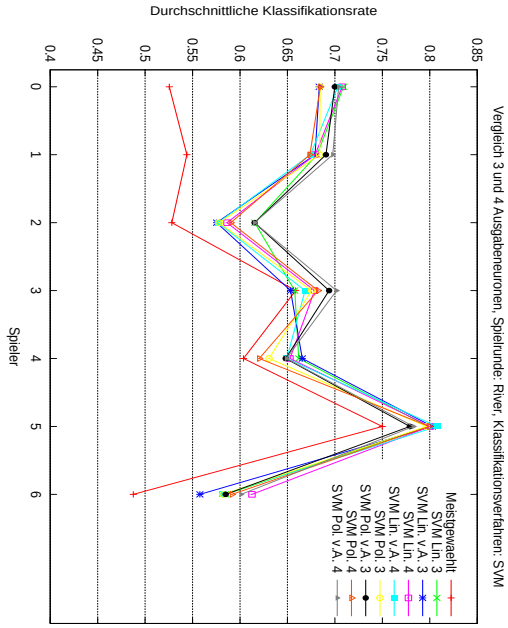

Abb. A.8: *Vergleich 3 und 4 Ausgabeneuronen, River, SVM*

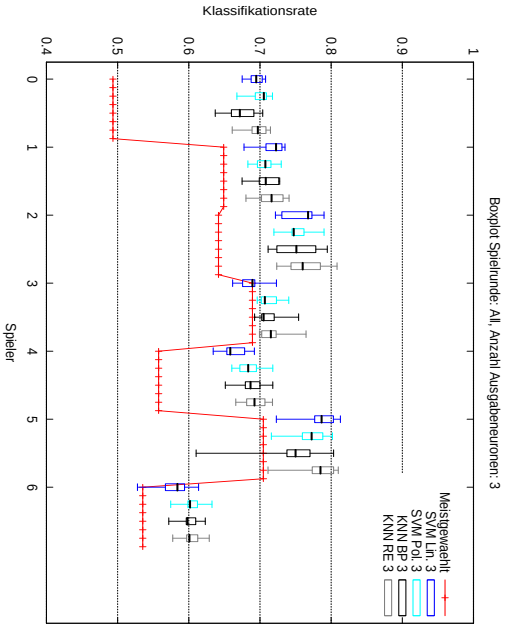

Abb. A.9: *Boxplots der Spielrunde All, 3 Ausgabeneuronen*

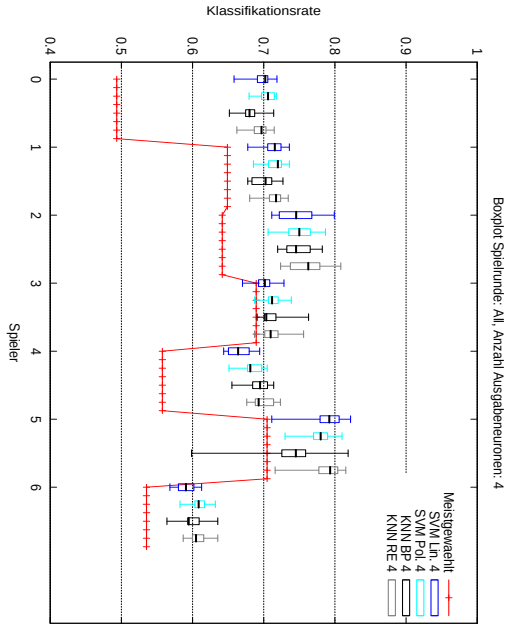

Abb. A.10: *Boxplots der Spielrunde All, 4 Ausgabeneuronen*

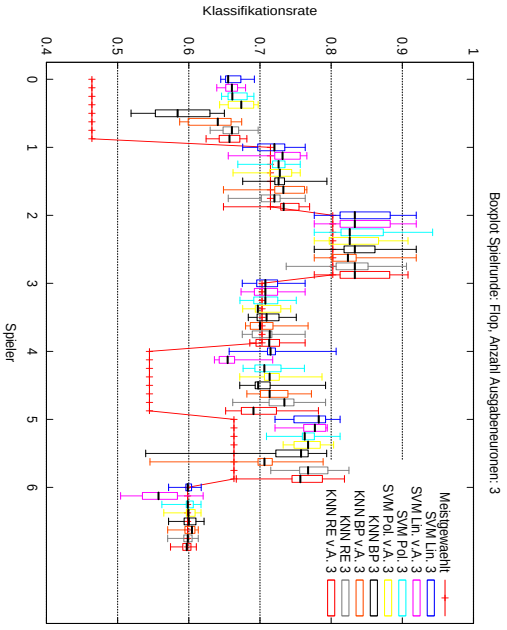

Abb. A.11: *Boxplots der Spielrunde Flop, 3 Ausgabeneuronen*

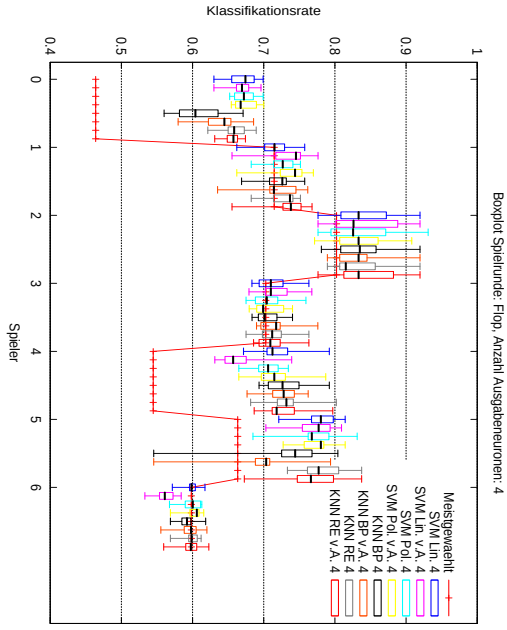

Abb. A.12: *Boxplots der Spielrunde Flop, 4 Ausgabeneuronen*

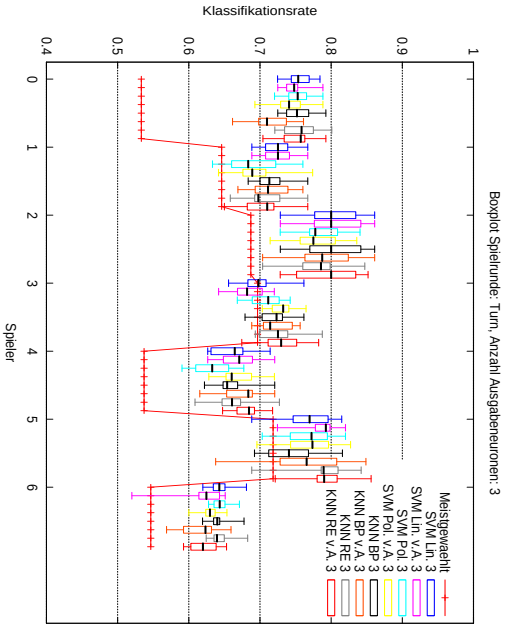

Abb. A.13: *Boxplots der Spielrunde Turn, 3 Ausgabeneuronen*

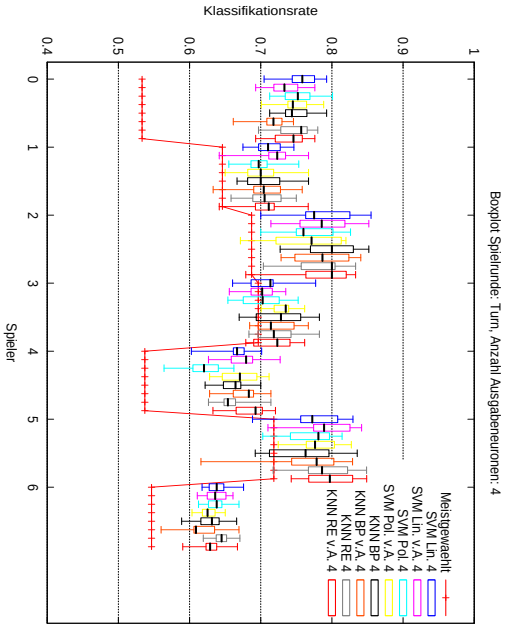

Abb. A.14: *Boxplots der Spielrunde Turn, 4 Ausgabeneuronen*

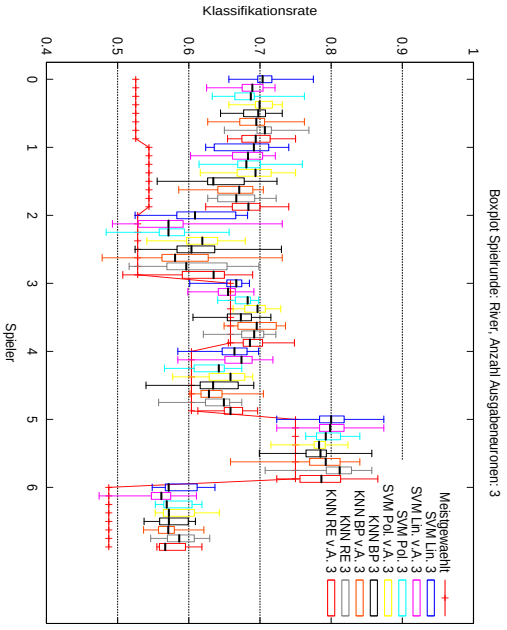

Abb. A.15: *Boxplots der Spielrunde River, 3 Ausgabeneuronen*

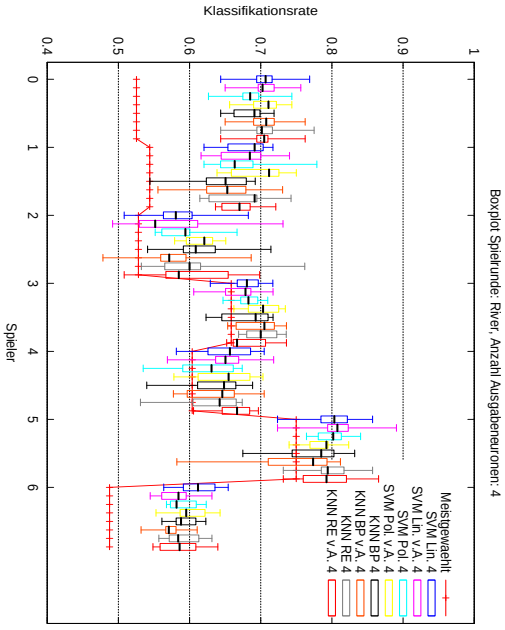

Abb. A.16: *Boxplots der Spielrunde River, 4 Ausgabeneuronen*