Diplomarbeit

Untersuchung latenter Strukturen im Parameterraum maschineller Lernverfahren

Sarah Risse
September 2015

Gutachter:
Prof. Dr. Katharina Morik
Dipl.-Inf. Nico Piatkowski

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS VIII)
http://www-ai.cs.uni-dortmund.de

Kurzzusammenfassung

In dieser Diplomarbeit wird eine Methode vorgestellt und durchgeführt, um Modelle mit einer kleineren Komplexität zu erzeugen. Diese sollen im Vergleich zum Ursprungsmodell eine ähnliche Güte, aber eine kleinere Komplexität aufweisen. Hierfür werden überwachte Lernverfahren in Verbindung mit unüberwachten verwendet. Diese Methode unterscheidet sich von dem häufig verwendeten Vorgehen, Datensätze vor Erlernen eines Modells zu clustern. Dieser Unterschied bringt eine Implementierung neuer Operatoren für das Programm Rapidminer mit sich.

Nach den relevanten Grundlagen zum Thema maschinelles Lernen und einer Einführung der verschiedenen Lernverfahren, wird das ausgearbeitete Vorgehen vorgestellt. Darauf folgt eine Beschreibung der Implementierung der verschiedenen Operatoren und eine Evaluation mithilfe diverser Experimente auf verschiedenen Datensätzen.

Anhand der Ergebnisse wird sich zeigen, dass die vorgestellte Idee dieser Arbeit plausibel und effektiv ist, jedoch nicht für alle vorgestellten Lernverfahren gleich gute Ergebnisse liefert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1 Einleitung

Mithilfe maschineller Lernverfahren können Strukturen innerhalb einer Datenmenge gefunden und untersucht werden. Ein Lernverfahren kann man sich als Algorithmus vorstellen, welcher anhand von gegebenen Beispielen eine vorgegebene Lernaufgabe lösen kann. Es gibt verschiedene Arten von Lernverfahren, welche in Abschnitt 2.2 ausführlicher erläutert und diskutiert werden.

Aus einem gegebenen Datensatz und dem verwendeten Lernverfahren ergibt sich ein Parameterraum, der in Abhängigkeit von den Daten und des gewählten Lernverfahrens größer oder kleiner werden kann. Der Parameterraum besteht aus der Menge aller möglichen Parameter, die das Lernproblem beschreiben. Diese Parameter sind abhängig vom gewählten Lernverfahren und bestimmen, wie genau die Abbildung der Eingabe auf die Ausgabe aussieht.

Innerhalb des Parameterraumes soll nun eine latente Struktur gefunden werden. In diesem Fall ist die Struktur eine Teilmenge des Parameterraums, wird aber nicht weiter spezifiziert, da sie jeweils vom Parameterraum und vom gewählten Lernverfahren abhängig ist. Latente Variablen sind manchmal nicht sicht- und/oder messbar. Häufig ist es aber auch einfach zu aufwendig oder zu teuer, um sie zu bestimmen. Das Hauptproblem hierbei ist die Frage, ob überhaupt eine latente Struktur vorhanden ist. Ein Beispiel für eine latente Struktur könnte die Eigenschaft sein, dass die eine Hälfte der Parameter ein Vielfaches von einem bestimmten α und die andere Hälfte von einem bestimmten β ist, basierend auf der Idee von Piatkowski/et al. [1]. Um eine latente Struktur innerhalb der Daten finden zu können, ist es notwendig, dass diese Daten normalisiert sind. Das heißt, dass z. B. alle verwendeten Einheiten auf die gleiche Größe angepasst werden müssen. So können z. B. 2 g nicht mit 2 kg verglichen werden, da die Maßeinheiten nicht identisch sind. Das bedeutet, dass die 2 kg zunächst in 2000 g umgerechnet werden müssen, um eine geeignete Vergleichsgrundlage zu schaffen.

Aus diesen Überlegungen ergibt sich der Titel, den diese Arbeit trägt: „Untersuchung latenter Strukturen im Parameterraum maschineller Lernverfahren“. Wobei das Untersuchen in diesem konkreten Fall bedeutet, dass unüberwachte Lernverfahren angewendet werden, um latente Strukturen in den Parameterräumen von überwachten Lernverfahren zu finden. Und wie sich beispielsweise die Einschränkung des Parameterraums auf einen Unterraum auf das Lernergebnis auswirkt.

Angenommen, eine nicht triviale und umfangreiche Datenmenge D soll untersucht werden. Diese Datenmenge kann mithilfe von maschinellen Lernverfahren modelliert werden, um damit Vorhersagemethoden aus den Daten abzuleiten oder zu klassifizieren. Bekannte Methoden hierfür sind z. B. Stützvektormaschinen (s. Abschnitt 2.2.2), lineare Regression (s. Abschnitt 2.2.4) oder probabilistische grafische Modelle wie das *Markov Random Field* (MRF), welche jedoch in dieser Arbeit nicht betrachtet werden. Stattdessen wird der naive Bayes Algorithmus (s. Abschnitt 2.2.3) betrachtet, der die gleiche Struktur wie ein MRF hat, sich aber einfacher anwenden lässt. Qualitativ gibt es

zwischen beiden Verfahren kaum einen nennenswerten Unterschied, da alle grafischen Modelle eine ähnliche Struktur haben und sich die Experimente, die im Zuge dieser Arbeit durchgeführt werden, sehr gut auf den naiven Bayes Algorithmus anwenden lassen. Des Weiteren ist es vonnöten, eine geeignete und effiziente Datenstruktur zu finden, die die Daten und das Modell sowohl komprimiert abspeichert, als auch einen schnellen Zugriff auf die einzelnen Werte liefert. Das Ganze funktioniert nur, wenn es eine latente Struktur innerhalb des Modells gibt. Stichwort hierbei soll *Minimum Description Length* (MDL) (s. Abschnitt 2.1) sein, was ein Maß für die Komplexität eines Modells darstellt. Motiviert ist MDL durch das *Sparsamkeitsprinzip* (*Occam's razor* oder *Ockhams Rasiermesser*), welches besagt, dass von mehreren Theorien zur Lösung eines Problems, die alle die gleiche Lösung bereitstellen, die einfachste als richtig angenommen wird.

Bemerkung 1. Von mehreren Theorien, die alle die gleiche Lösung präsentieren, nutze die einfachste. (William of Occam)

Es wird also eine Datenstruktur und eine Methode gesucht, welche nicht nur auf die *sparsity,* also die Komprimierbarkeit der Modelle reduziert ist, sondern auch die Abbildung, die weniger Werte auf die eigentliche Dimension des Parameterraums innerhalb eines Vektors abbildet, effizient abspeichern kann.

1.1 Übersicht

In Kapitel 2 werden einige Grundlagen und Grundbegriffe des machinellen Lernens erläutert. Es werden die unterschiedlichen Lernverfahren, die im weiteren Verlauf der Arbeit verwendet werden, eingeführt. Das darauf folgende Kapitel beinhaltet die grundsätzliche Idee, die hinter dieser Arbeit steht und beschreibt inwieweit es latente Strukturen innerhalb von überwachten Lernverfahren gibt. In Kapitel 4 werden die, im Laufe dieser Arbeit, entwickelten Operatoren vorgestellt und es wird erläutert welche Probleme es zu beachten gibt. Die Verwendung dieser Operatoren innerhalb einer Reihe von Experimenten wird in Kapitel 5 beschrieben. Abschließend folgt eine Zusammenfassung sowie ein Ausblick, inwiefern die Ergebnisse weiter genutzt werden können.

2 Grundlagen

In diesem Kapitel werden einige Grundlagen des maschinellen Lernens beschrieben. Maschinelles Lernen bezeichnet ein Forschungsgebiet der Informatik, welches sich mit „computergestützter Modellierung und Realisierung von Lernphänomenen“ [2] auseinandersetzt. Hat man eine große Menge von Daten, kann man mithilfe des maschinellen Lernens aus dieser Menge Wissen extrahieren und adaptive Systeme, also Systeme die in der Lage sind ihr Verhalten an die Gegebenheiten anzupassen, realisieren, indem das genutzte Modell aus den gegebenen Daten lernt. Im Allgemeinen ist es nicht so einfach, eine genaue Definition von „lernen“ zu geben. Im normalen Sprachgebrauch versteht man unter dem Begriff „lernen“ das Aneignen und Verwenden von Wissen, Kenntnissen und Fertigkeiten.

Computer können auch lernen, wenn auch in einem anderen Sinne. Hier hilft die Definition von Ryszard Michalski von 1986: „Lernen ist das Konstruieren oder Verändern von Repräsentationen von Erfahrungen“. Das heißt ein vorhandenes System hat eine gewisse Menge an Daten (Erfahrungen) und soll aus diesen, neues Wissen konstruieren oder schlussfolgern. Womit man zu den verschiedenen Inferenzarten *Deduktion, Induktion* und *Abduktion* kommt, welche sich in dem Bereich der Logik wiederfinden und das Schlussfolgerungswesen beschreiben. Hier ist neues oder auch erlerntes Wissen logisch aus gegebenen Wissen oder Erfahrungen geschlussfolgert. Der Begriff *Deduktion* beschreibt eine sichere Schlussfolgerung B aus gegebenen Wissen W. *Abduktion* und *Induktion* sind jeweils unsichere Schlussweisen, hier wird eine Erklärung E für eine Beobachtung B mit dem gegebenen Wissen W abgegeben bzw. aus vielen Beispielen wird unsicheres Wissen abgeleitet. Diese Herangehensweise kommt aus dem Bereich der Logik und eignet sich als Verstehensgrundlage. Das moderne maschinelle Lernen befasst sich eher mit der Optimierung und dem Finden einer Zielfunktion [3].

Innerhalb des maschinellen Lernens gibt es verschiedene Lernaufgaben. Hier sind das überwachte, das unüberwachte Lernen und das Verstärkungslernen zu nennen. Auf letzteres wird hier aber nicht weiter eingegangen. Die am häufigsten untersuchte Lernaufgabe ist die des überwachten Lernens. Für die einzelnen Lernverfahren gibt es unterschiedliche Lernvarianten, z. B. Entscheidungsbäume (s. Abschnitt 2.2.1), Support Vector Machine (s. Abschnitt 2.2.2) und probabilistische Modelle (Abschnitt 2.2.3) innerhalb des überwachten Lernens und Clusterverfahren (s. Abschnitte 2.2.5 und 2.2.6) innerhalb des unüberwachten Lernens. Eine genauere Erläuterung findet sich in Abschnitt 2.2 wieder.

Eine Lernaufgabe ist definiert durch die gegebenen Eingaben, die gewünschten Ausgaben und die gegebenen Randbedingungen. Man kann also sagen, dass ein System lernt, wenn es bei den Eingaben unter Einhaltung der Randbedingungen (z. B. minimale Kosten, Speicherverbrauch oder Laufzeit) die gewünschten Ausgaben erzeugt.

In der Praxis sind zwei große Anwendungsbereiche zu erkennen. Zum einen die „interaktive Analyse von vorher gesammelten Datenbeständen mithilfe von Lernverfahren“ [2] und zum anderen die „Verwendung von Lernverfahren zur Erzielung adaptiven Verhaltens“ [2]. Der zweite Bereich beschreibt ein System, welches fortlaufend Daten bekommt, mithilfe derer Prognosen getroffen werden müssen. Die Experimente (s. Ka-

pitel 5) sind im ersten Bereich anzusiedeln, da ein gegebener, sich nicht verändernder, Datensatz als Eingabe genutzt wird.

Maschinelles Lernen findet man in den verschiedensten Anwendungsgebieten. So kann es z. B. kognitionsorientiert, theoretisch oder algorithmisch-anwendungsorientiert sein. Im kognitionsorientierten Gebiet erhofft man sich das Lernen aus Sicht des Menschen besser zu verstehen und nutzt hierfür Computermodelle. Im theoretischen Teil stellt man sich die Frage, was für Lernaufgaben mit welchem Aufwand zu verwirklichen sind. Im algorithmisch-anwendungsorientierten Bereich möchte man einen praktischen Nutzen erhalten, indem hier die Systeme in der Praxis relevante Lernaufgaben lösen können. Auch in der täglichen Nutzung des Internets oder im interaktiven Analysieren von Datenbeständen, mithilfe dessen z. B. unbekannte Sachverhalte aus der bestehenden Datenmenge erforscht werden können, findet sich das maschinelle Lernen wieder [2]. Die in dieser Arbeit verwendeten Experimente sind im algorithmisch-anwendungsorientierten Bereich zu finden.

Konkrete Anwendungsgebiete sind z. B. Systeme, um Handschrift oder Sprache automatisch zu erkennen. Hier besteht das Problem darin, dass es viele unterschiedliche Darstellungen eines einzelnen Buchstabens gibt, da jeder Mensch eine individuelle Handschrift hat. Das Ziel ist es also, dem System beizubringen, welche spezifischen Merkmale die einzelnen Symbole haben, damit diese in verschiedensten Formen erkannt werden können. Ein ähnliches Beispiel ist die Erkennung von Gesichtern, die bei der Polizeiarbeit eine große Rolle spielt. Auch hier müssen Merkmale erkannt werden, um Gesichter aus unterschiedlichen Perspektiven zu bestimmen. Auch Sprache kann erkannt werden. Allerdings brauchen Computer für das, was ein Mensch ganz intuitiv heraushören und übersetzen kann, eine sehr große Datensammlung mit Beispielen, wie verschiedene Worte ausgesprochen werden.

Ein anderer Anwendungsfall ist das selbstständige Navigieren von Robotern oder Fahrzeugen. Hierfür müssen die Umgebungs- und Witterungsbedingungen gelernt und erkannt werden, damit das Fahrzeug entsprechend reagieren kann [4].

2.1 MDL und Occams Razor

Das *minimum description length principle* (MDL) wurde 1978 von Jorma Rissanen [5], basierend auf der *Kolmorogov-Komplexität,* vorgestellt und ist ein generelles Prinzip für die Anwendung induktiver Inferenz [6]. Die Kolmogorov-Komplexität gibt ein Maß für die Differenzierbarkeit einer Zeichenkette an, das durch die Länge des kürzesten Programms, welches diese Zeichenkette erzeugen kann, beschrieben wird. Mit der Komplexität kann man die beste Komprimierbarkeit der gegebenen Daten herausfinden. Hierbei ist allerdings zu beachten, dass die Wahrscheinlichkeit der Unkomprimierbarkeit der Zeichenkette steigt, wenn sich Komplexität und Länge der Zeichenkette annähern. Für diesen Fall kann man annehmen, dass es sich bei der Zeichenkette um eine zufällige Abfolge handelt, die keinerlei Regelmäßigkeiten inne hat. Die induktive Inferenz ist die Schlussfolgerung, die aus einer gegebenen Menge M von Beispielen automatisch erfolgt und Wissen bereitstellt, indem in M Gesetze und Regelmäßigkeiten gesucht werden. Die Idee von MDL ist, die Nutzung jeder Regelmäßgkeit zur Datenkomprimierung. Je

mehr Regelmäßigkeiten es in den Daten gibt, umso größer ist die Kompressionsrate der Daten. Daraus folgt wiederum, dass Daten, in denen es keine Regelmäßigkeiten gibt, auch nicht komprimiert werden können. Also kann man sagen, dass man die Fähigkeit zur Komprimierung als das Finden von Regelmäßigkeiten auffassen kann. Allerdings muss man zu jedem Zeitpunkt in der Lage sein, die Daten zu dekomprimieren und in ihre ursprüngliche Darstellung zurückzuführen.

Angenommen, man hat eine beliebige Zeichenkette, in der sich eine Regelmäßgkeit finden lässt. Zum Beispiel

$$0101010101010101010101 \ldots 010101010101$$

bei der 100 mal hintereinander die Zeichenfolge 0101 wiederholt wird. Dieses ist sprachlich sehr schnell zu beschreiben: „Schreibe 100 mal 0101 hintereinander auf“. Nun benötigt man eine Beschreibungsmethode, um den sprachlichen Ansatz zu formalisieren und in ein Alphabet zu übersetzen. Eine solche Methode formuliert also eine Datensequenz in eine Sequenz von Bits. Wenn man nun ein Programm schreiben möchte, welches genau diese Sequenz ausgibt, dann ist die kürzeste Lösung in Algorithmus 1 dargestellt. Hier wird einfach nur die Sequenz 01 200 mal ausgegeben.

Algorithmus 1 : Beispielcode

```
1 for i=1 to 200 do
2     print(01)
```

Die oben erwähnte Kolmogorov-Komplexität ist definiert durch die Länge eines solchen Programms, was die gewünschte Sequenz ausdruckt. Im Allgemeinen ist es jedoch schwierig die Kolmogorov-Komplexität auf reale Lernprobleme anzuwenden, da sie häufig nicht berechnet werden kann. Das heißt, dass es kein Programm gibt, das für einen beliebigen Datensatz D das kürzeste Programm K zurückgeben kann und in endlicher Zeit terminiert. Zudem gibt es entsprechend auch kein Programm, welches nur die gesuchte Länge von K ausgeben kann. Man hat zudem immer eine Konstante, die von der verwendeten Beschreibungsmethode abhängig ist. Die zweite zentrale Idee von MDL ist, die Gleichsetzung des Lernens mit dem Auffinden von Regelmäßigkeiten in Datensätzen. Es existieren unterschiedliche Arten von MDL, welche verschiedene Methoden zur induktiven Inferenz liefern. Aber es gibt keine „einzige, optimale MDL Methode/Prozedur“ [6]. Man kann festhalten, dass man an der Länge der Codierung einer Sequenz interessiert ist. Codierung bedeutet hier, dass eine gegebene Zeichenkette mithilfe eines Alphabets umgewandelt wird.

Wie schon zu Beginn der Arbeit erwähnt, ist die Motivation für MDL das Prinzip der Sparsamkeit, auch *Ockhams Rasiermesser* (Occams Razor (s. Abschnitt 1)) genannt, welches besagt, dass man bei mehreren vorhandenen Lösungen die einfachste nutzt. MDL hilft also für einen gegebenen Datensatz D und einer Menge an Hypothesen H, das H zu finden, bei dem D am besten komprimiert wird [6].

2.1.1 Kodierung und Entropie

Kodierung bedeutet, eine gegebene Sequenz von Symbolen mithilfe einer anderen Sequenz zu beschreiben. Die Symbole kommen aus einem Alphabet, welches aus einer endlichen Menge besteht. Angenommen man sucht die Länge der Kodierung einer Sequenz $\boldsymbol{x}\in\mathcal{X}^n$, wobei $\mathcal{X}^n$ die Menge aller Beobachtungen darstellt und entsprechend endlich und abzählbar ist. Ohne einen Verlust der Allgemeingültigkeit zu riskieren, lässt sich $\boldsymbol{x}^n$ mit den Symbolen aus dem Alphabet $\mathcal{X} = \{0, 1\}$ kodieren. Soll z. B. die Sequenz $cabbab$ mit $a = 0, b = 10, c = 11$ kodiert werden, so erhält man die eindeutige Kodierung 1101010010, welche sich ohne Verlust dekodieren lässt [6]. Um die Daten zu kodieren, benötigt man eine sogenannte *description methode*. Diese ordnet die Sequenzen den neuen Beschreibungen zu.

Die Kraft-Ungleichung besagt, dass für jede *description methode* C, für ein endliches Alphabet $A = 1, \ldots, m$, die Wortlänge des verwendeten Codes $L_c(1), \ldots, L_c(m)$ die Ungleichung $\sum_{a\in A} 2^{-L_c(a)} \leq 1$ erfüllen muss. Daraus ergibt sich die Existenz eines eindeutig dekodierbaren Codes für diese Wortlänge. Der Beweis hierfür findet sich in *The minimum description length principle* von Peter Grünwald ([6], S. 92).

Sei p eine Wahrscheinlichkeitsverteilung über $\mathcal{X}^n$, dann gibt es einen Code C für $\mathcal{X}^n$, sodass $\forall \boldsymbol{x}^n \in \mathcal{X}^n, L_c(\boldsymbol{x}^n) = -\log P(\boldsymbol{x}^n)$ gilt. Daraus folgt, dass zu einer großen Wahrscheinlichkeit eine kurze Codelänge zugeordnet wird und umgekehrt.

Definition 1. Entropie. Die *Entropie* misst die Unsicherheit in einer Wahrscheinlichkeitsverteilung p [7] und ist wie folgt definiert:

$$\mathcal{H}(p) \quad = \quad -\sum_{x\in X} p(\boldsymbol{x}) \log p(\boldsymbol{x})$$

Mit diesem Hintergrund lässt sich nun die Entropie erklären. In diesem Fall ist die Entropie von p, die erwartete Anzahl an Bits, die benötigt werden, um p mit dem optimalen Code zu kodieren. Die Entropie ist eine wichtige Größe zur Berechnung der Komplexität (s. Abschnitt 3.2) und wird innerhalb der Experimente ausgewertet.

2.2 Lernverfahren

Es gibt verschiedene Arten von Lernverfahren. Insbesondere gibt es die Unterscheidung zwischen überwachtem und unüberwachtem Lernen.

Innerhalb des überwachten Lernens wird aus einer gewissen Anzahl von Beispielen, die alle das gleiche Problem beschreiben, eine allgemeine Funktion gelernt, mit der sich weitere Beispiele klassifizieren lassen. Alle Modelle im überwachten Lernen sind parametrisiert[1]. Das heißt, dass man bei der Suche nach der abbildenden Funktion (s. Def. 2) nicht verschiedene Formeln ausprobiert, sondern, dass man eine Formel über

[1] Es gibt sogenannte *nicht-parametrische Modelle*, z. B. Gaußsche Prozesse. Diese stützen ihre Vorhersagen auf eine Teilmenge der Eingabedaten. Da diese Teilmengen aber ebenfalls abgespeichert werden müssen, können sie als Parameter des Verfahrens betrachtet werden.

Parameter θ hat. Ändert sich nun θ, dann ändert sich dadurch auch die Funktion. Um die Güte der Funktion zu messen, nutzt man eine sogenannte Verlustfunktion (s. Def. 3), die berechnet, wie gut oder schlecht θ auf den Daten ist. Ziel ist es die Verlustfunktion zu minimieren.

Definition 2. Überwachtes Lernen.
Gegeben sei ein *Datensatz* $\mathcal{D} = \{(\boldsymbol{x}, y)^i\}_{1 \leq i \leq N}$ mit $|\mathcal{D}| = N$ *Instanzen* $(\boldsymbol{x}, y)$, wobei jede Instanz aus einer Beobachtung $\boldsymbol{x} \in \mathcal{X}$ und einem *Label* $y \in \mathcal{Y}$ besteht. In der Regel ist die *Domäne* $\mathcal{X}$ der Beobachtungen d-dimensional, d.h. ein Kreuzprodukt verschiedener einzelner Domänen $\mathcal{X} = \mathcal{X}_1 \otimes \mathcal{X}_2 \otimes \cdots \otimes \mathcal{X}_n$ (z. B. $\mathcal{X} = \mathbb{R}^n$). Die Domäne des Labels ist dagegen nur im Fall von strukturellen Labeln mehrdimensional, ansonsten 1-dimensional. Die Aufgabe des *überwachten Lernens* besteht im Auffinden einer Funktion $f : \mathcal{X} \to \mathcal{Y}$ mit $f \in \mathcal{F}$, die die vorhandenen Daten in $\mathcal{D}$ möglichst gut „reproduzieren“ soll. Die Güte wird dabei mithilfe einer *Verlustfunktion* gemessen. Konkrete Kombinationen von Funktionenraum $\mathcal{F}$ und Verlustfunktion entsprechen einzelnen *Lernverfahren*.

Ein gutes Beispiel hierfür sind Filter für unerwünschte E-Mails. Der E-Mail Client bekommt vom Nutzer eine Vorgabe, welche Inhalte oder Absender als Spam eingestuft werden sollen und anhand dieser Information kann das Programm die betroffenen Nachrichten filtern und als Spam deklarieren. Es wird also aus gegebenen Ein- und Ausgabedaten einer Trainingsmenge gelernt und danach kann das Gelernte auf eine Testmenge angewendet werden. Hierbei ist es wichtig, dass es nicht zu einer Überanpassung (*Overfitting)* kommt. Von einer Überanpassung eines Modells spricht man, wenn sich das Modell zu genau den gegebenen Daten anpasst und Abweichungen schon als Ausreißer bezeichnet bzw. falsche Daten miteinbezieht. Hier entsteht das Problem, dass neue Testdaten nicht richtig klassifiziert werden können. Nur die Trainingsdaten werden als „richtig“ eingestuft, da das Modell diese auswendig gelernt hat. Dadurch hat das Modell zusätzliche, irrelevante Variablen inne, die nicht zum Einordnen der Ergebnisse genutzt werden können. Overfitting verschlechtert die Schätzeigenschaften eines Modells ungemein, da es nur auf die gegebene Stichprobe, aber nicht auf die Allgemeinheit ausgerichtet ist. Vermeiden lässt sich dieses Problem, indem eine möglichst geringe, jedoch trotzdem möglichst umfassende Anzahl an Parametern zur Modellierung genutzt wird. Die Auswahl eines geeigneten Trainingsdatensatzes mit möglichst umfangreichen Daten ist eine wichtige Voraussetzung, um ein allgemeingültiges Modell zu generieren.

Definition 3. Verlustfunktion.
Gegeben sei ein beliebiger Eingabevektor $\boldsymbol{x} \in \mathbb{R}^n$ und eine Ausgabevariable $y \in \mathbb{R}$ mit der Wahrscheinlichkeit $p(\boldsymbol{x}, y)$. Gesucht ist nun eine Funktion $f(\boldsymbol{x})$, um y vorhersagen zu können. Hier wird eine *Verlustfunktion* $L(y, f(\boldsymbol{x}))$ benötigt, welche die Diskrepanz zwischen der Vorhersage und den wirklichen Daten darstellt. Sehr häufig wird die *quadratische Abweichung* (oder auch der quadratische Fehler), also die Summe der kleinsten Abweichungen mit $L(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$, als Verlustfunktion verwendet. Alternativ zum quadratischen Fehler wird auch die Maximum-Likelihood-Methode genutzt. Hier wird von den gegebenen Parametern jener gewählt, der die größte Wahrscheinlichkeit hat, sodass die vorhergesagten Ausgabedaten realisiert werden.

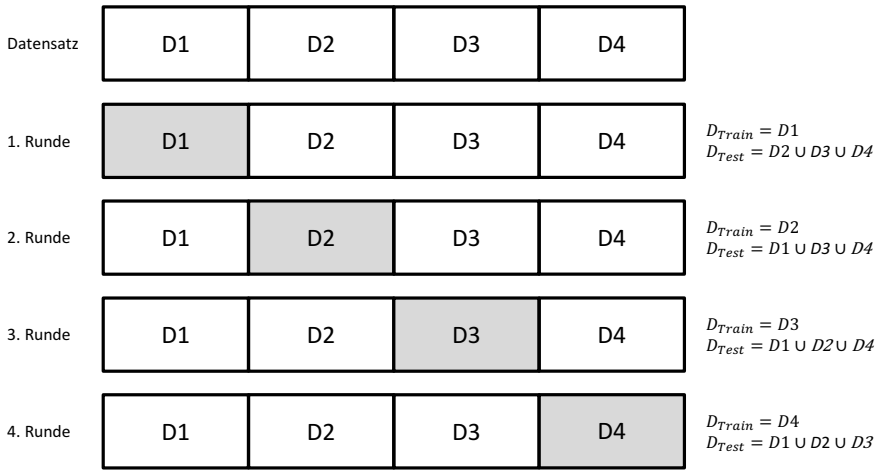

Abbildung 1: *Kreuzvalidierung. Hier wird in jeder Runde ein Teil des Datensatzes ausgelassen und der Rest als Testdatensatz verwendet.*

Für die in Abschnitt 2.2.2 beschriebene Support Vector Machine wird der sogenannte *Hinge Loss* genutzt. Diese Verlustfunktion ist definiert durch:

$$L(\boldsymbol{\beta}, \boldsymbol{x}, y) \quad = \quad \max(0, 1 - y \cdot (\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)),$$

mit $\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{w} \\ b \end{pmatrix}$.

Im Problemfeld des überwachten Lernens gibt es verschiedene Analyseformen für die gegebenen Daten. Hier sind Regression und Klassifikation zu nennen. Bei einem Regressionsproblem sind die Ausgabedaten quantitativ und somit Zahlwerte. Bei der Klassifikation hingegen handelt es sich um diskrete Werte und die Ausgabewerte sind in Klassen einzuteilen

Um bei den Experimenten (s. Kapitel 5) zu bestimmen, wie sich die jeweilige Methode auf bisher unbekannten Daten verhalten wird, bedient man sich dem Verfahren der Kreuzvalidierung. Hier werden die Daten in Trainings- und Testdatensätze aufgeteilt. Es ist die weitverbreitetste und einfachste Methode, um mit Vorhersagefehlern entsprechend umzugehen. Das Verfahren schätzt den Gesamtfehler der Vorhersage: $Err = E[L(y, f(\boldsymbol{x})]$, also den Erwartungswert der Verlustfunktion. Wenn genügend Daten zur Verfügung ständen, bräuchte man das Verfahren nicht, da man direkt einen Teil der Daten zur Verifizierung benutzen könnte. Häufig sind aber zu wenig Daten gegeben und es ist nicht sinnvoll, diese in Trainings- und Testmenge aufzuteilen, sodass man auf die Kreuzvalidierung zurückgreifen muss. Wie schon erwähnt, wird hierbei der Datensatz in Trainings- und Testdaten aufgeteilt. Dieser Vorgang wird mit unterschiedlichen Einteilungen wiederholt und für jede Runde wird dadurch der Fehler berechnet.

Die Anzahl der Runden wird meistens auf fünf oder zehn festgelegt. Abbildung 1 soll die Funktionsweise der Kreuzvalidierung darstellen. Gegeben ist ein Datensatz $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$, mit $D_i \neq 1$. Angenommen, man kreuzvalidiert nun über vier Runden, dann ist in jeder Runde jeweils ein anderer Datensatz als Trainingsdatensatz ausgewählt. Für die i-te Runde wird jeweils der i-te Datensatz ausgelassen und zur Verifizierung genutzt. Allerdings kann es hierbei zu einer hohen Varianz kommen, da sich die Trainingsdatensätze sehr ähneln [3]. Hier wird das gesuchte $\boldsymbol{\theta}$ auf dem Trainingsdatensatz gefunden und dann die Verlustfunktion für den Testdatensatz berechnet, um festzustellen, wie gut das Modell auf einem vorher nicht bekannten Datensatz ist. Wenn man ohne Kreuzvalidierung die Daten manuell in zwei Datensätze teilen würde, dann könnte es passieren, dass diese Aufteilung zufällig besonders gut oder besonders schlecht ist und der dadurch geschätzte Fehler unzuverlässig wird. Um dem entgegenzuwirken ist die Kreuzvalidierung unumgänglich und wird auch in den Experimenten dieser Arbeit genutzt. Innerhalb der Kreuzvalidierung werden die Modelle der überwachten und unüberwachten Verfahren gelernt. Wenn die Rundenanzahl $K = N$ und $D_i = 1$ entspricht, dann nennt man das auch *leave-one-out-Kreuzvalidierung.*

Beim unüberwachten Lernen hat man eine gegebene Datenmenge D, weiß allerdings nicht, welche Ausgabewerte es geben wird. Das heißt, dass anders als beim überwachten Lernen keine klassifizierten Beispiele existieren, anhand denen der Algorithmus lernen kann. Man bezeichnet das auch als „Lernen ohne Lehrer“ [3]. Das Ziel des unüberwachten Lernens ist es, Regelmäßigkeiten, Eigenschaften und interessante Strukturen in $p(\boldsymbol{x})$ herauszufinden. Die Dimension des Vektors $\boldsymbol{x}$ ist häufig viel größer, als im überwachten Lernen. Auch die Eigenschaften sind meist komplizierter. Mithilfe des unüberwachten Lernens können z. B. sogenannte Cluster innerhalb der Daten gefunden werden. Ein weiteres Beispiel für das unüberwachte Lernen sind auch Schätzungen von Wahrscheinlichkeitsverteilungen.

Unter einem Cluster versteht man eine Gruppe von Datenobjekten mit ähnlichen Eigenschaften. Das heißt Objekte innerhalb eines Clusters sind in gewisser Weise ähnlich und unterscheiden sich von Objekten, die sich in anderen Clustern befinden. Objekte, die keinem Cluster zugeordnet werden können, werden Ausreißer, Noise oder Outlier genannt. Die Größe und Eigenschaft der einzelnen Cluster hängt von dem gewählten Clusterverfahren ab. Es gibt verschiedene Verfahren zur Ermittlung solcher Cluster, welche in den Abschnitten 2.2.5 und 2.2.6 beschrieben werden.

Daten zu clustern bedeutet, dass sie in irgendeiner Art und Weise charakterisiert und in Kategorien eingeteilt werden. Hier ist wichtig zu erwähnen, dass es die Unterscheidung zwischen numerischen und kategorischen Modellen gibt. Das *frequent itemset mining* ist eine weitere Technik, um Daten zu typisieren. Angenommen, es sind verschiedene Mengen mit ähnlichen Elementen gegeben, dann beschreibt Frequent Itemset Mining das Problem, welche Elemente häufig zusammen auftreten. Ein beliebtes Beispiel ist hier der Einkaufswagen. Für die Betreiber eines Supermarkts ist es wichtig, welche verschiedenen Waren zusammen gekauft werden, damit sie entsprechend ihre Produkte gewinnbringend präsentieren können. Hierbei handelt es sich um ein kategorisches Verfahren, weshalb sich diese Arbeit im weiteren Verlauf nicht mit diesem Verfahren beschäftig. Hier werden nur numerischen Modelle untersucht. Da zunächst die Überlegung

bestand dieses Verfahren näher zu betrachten, ist es hier erwähnt.

In den Experimenten (s. Kapitel 5), die zu dieser Arbeit gehören, werden vier verschiedene überwachte Lernverfahren genutzt und innerhalb der Abschnitte 2.2.1 bis 2.2.4 beschrieben:

- Entscheidungsbaum
- Support Vector Machine
- Probabilistische Modelle
- Lineare Regression

Die hier genutzten unüberwachten Lernverfahren k-means und DBSCAN werden in den Abschnitten 2.2.5 und 2.2.6 vorgestellt.

2.2.1 Entscheidungsbäume

Das erste Verfahren des überwachten Lernens ist der sogenannte *Entscheidungsbaum*, welches ein häufig genutztes hierarchisches Verfahren ist, um Daten zu klassifizieren. Hier werden Objekte, die durch Paare von Wert und Attribut beschrieben werden, einer bestimmten Klasse zugeordnet [3]. Der Vorteil hierbei ist sowohl die Einfachheit der Handhabung als auch die der Interpretation des Ergebnisses, da das gelernte Modell intuitiv visualisiert und interpretiert werden kann. Zudem haben Entscheidungsbäume eine geringe Laufzeit. Mithilfe von Entscheidungsbäumen können komplexe Entscheidungsprozesse und alle zugehörigen Entscheidungsmöglichkeiten dargestellt werden. Dieses Verfahren wird für Regression (Regressionsbaum) und auch Klassifikation (Klassifikationsbaum) genutzt. Der Unterschied besteht in der Entscheidung, welches Attribut das geeignetste ist [8]. In dieser Arbeit wird dieses Verfahren aber nur für die Lernaufgabe der Klassifikation genutzt.

Als Beispiel dienen die Daten aus Tabelle 1. Das *Alter* bedeutet, ob der Probant jung (1), pre-altersweitsichtig (2) oder altersweitsichtig (3) ist. Die *Art der Sehschwäche* beschreibt, ob die Person kurz- (1) oder weitsichtig (2) ist. Die dritte Spalte beschreibt, ob die Person eine *Hornhautverkrümmung* (1) hat oder nicht (2). Die Spalte *Tränenproduktion* gibt an, ob diese reduziert (1) oder normal ist (2). In der letzten Spalte erfolgt dann die Klassifikation. Hier bedeutet 1, dass der Patient harte Kontaktlinsen tragen kann, 2, dass es weiche sein sollten oder 3, dass der Patient keine Kontaktlinsen benutzen kann.

Anhand der Tabelle kann ein Entscheidungsbaum erstellt werden, welcher die Attribute zur Klassifikation nutzt. Dieser ist in Abbildung 2 zu sehen. Ein Entscheidungsbaum besteht aus einer Wurzel, internen Knoten und terminalen Blättern. Der Baum wird top-down, also ausgehend von der Wurzel bis hin zu den Blättern, erstellt. Die Wurzel ist das erste Attribut, in diesem Fall das Alter (s. Abb.2). Hierfür gibt es nun drei Möglichkeiten (1, 2, 3) die zum nächsten Attribut führen. Die Möglichkeiten oder

ID	I	II	III	IV	V
1	1	1	1	1	3
2	1	1	1	2	2
3	1	1	2	1	3
4	1	1	2	2	1
5	1	2	1	1	3
6	1	2	1	2	2
7	1	2	2	1	3
8	1	2	2	2	1
9	2	1	1	1	3
10	2	1	1	2	2
11	2	1	2	1	3
12	2	1	2	2	1

ID	I	II	III	IV	V
13	2	2	1	1	3
14	2	2	1	2	2
15	2	2	2	1	3
16	2	2	2	2	3
17	3	1	1	1	3
18	3	1	1	2	3
19	3	1	2	1	3
20	3	1	2	2	1
21	3	2	1	1	3
22	3	2	1	2	2
23	3	2	2	1	3
24	3	2	2	2	3

Tabelle 1: *Beispieldaten. Alter (I), Art der Sehschwäche (II), Astigmatisch (III), Tränenproduktion (IV), Kontaktlinsenart (V). Aus den Daten I - IV ergeben sich die Daten in V. Ziel ist es, herauszufinden, welche Art der Kontaktlinse für welchen Patienten geeignet ist.*

Bedingungen werden an die jeweiligen Kanten geschrieben. Dieses Vorgehen wird für die folgenden Attribute wiederholt, bis man an dem Klassifikationsergebnis angelangt ist. Dieses wird in dem Baum als Blatt dargestellt und hat entsprechend keine Kinder, also nachfolgende Attribute, mehr. Hat man einen kompletten Entscheidungsbaum generiert, ist es ein leichtes, neue Beispiele und Daten mithilfe des Baums zu klassifizieren. Hier ist es allerdings wichtig, dass man nicht dem trivialen Ansatz nachgeht und für jedes Beispiel bzw. jeden Datensatz einen eigenen Pfad erzeugt. Dies würde einen Baum ergeben, welcher vollständig ist und die Daten nur auswendig gelernt hat. Hier käme das Problem des Overfittings (s. Abschnitt 2.2) zutage. Es gäbe keinerlei Generalisierung, die auf neue Fälle angewendet werden kann. Ziel ist es also, einen kompakten Baum zu erstellen, der vorhandene Muster in der gegebenen Trainingsdatenmenge erkennen kann, damit zukünftige Datensätze ebenfalls klassifiziert werden können [3, 9].

Ein einzelner Baum liefert in vielen Fällen allerdings eine zu geringe Konfidenz, da er sich eventuell zu sehr an den Trainingsdatensatz anpasst. Deshalb werden häufig sogenannte Ensemble [10] von Entscheidungsbäumen genutzt. Das bedeutet, dass nicht nur ein Baum betrachtet wird, sondern mehrere. Das Verfahren *Random Forest* ist eine solche Ansammlung von Bäumen und ist sehr weit verbreitet, da es einfach durchzuführen ist und, entsprechend eines einzelnen Entscheidungsbaums, eine geringe Laufzeit hat. Ein Random Forest berechnet, auf einer zufälligen Teilmenge der Attribute des gegebenen Datensatzes, eine bestimmte Anzahl zufälliger Bäume. Für das Ergebnis macht man dann eine Mehrheitsentscheidung über alle Ausgaben der Bäume, z. B. wenn 65 % der Bäume das gleiche Label bevorzugen wird dieses ausgewählt. Im ersten Moment liegt die Vermutung nahe, dass sich die Laufzeit hierdurch im Vergleich zu einem einzelnen Baum verschlechtert. Allerdings können die einzelnen Bäume auf modernen Multicore Rechnern parallel gelernt und ausgewertet werden.

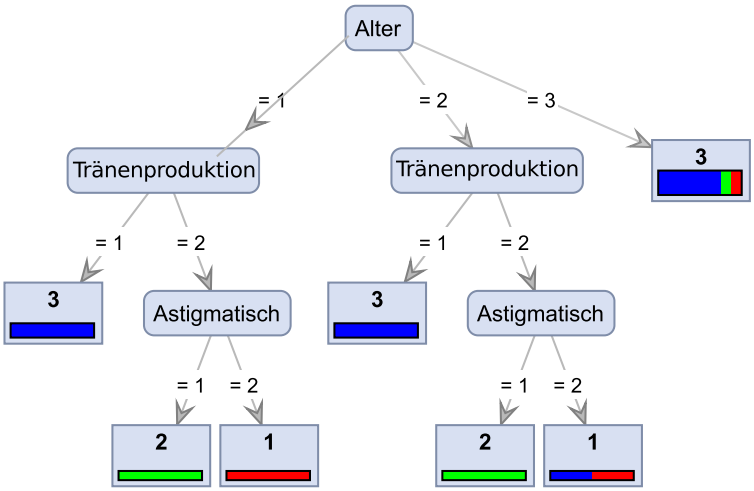

Abbildung 2: *Entscheidungsbaum. Dies ist ein Beispiel für einen Entscheidungsbaum, welcher sich aus den Daten aus Tabelle 1 ergibt. Zu oberst befindet sich die Wurzel, welche das erste Attribut, nach dem entschieden wird, darstellt. Darunter kommt man durch verschiedene Möglichkeiten zu verschiedenen Nachfolgern. Dieses wird bis zum letzten Blatt, der Klassifikation, wiederholt. Ein Blatt ist hierbei die geeignete Kontaktlinsenart. Dieser Baum wurde mit der Genauigkeit als Kriterium zur Auswahl des geeigneten Attributs genutzt.*

Der einem Entscheidungsbaum zugrunde liegende Algorithmus, muss für die Wurzel und jede folgende Verzweigung entscheiden, welches Attribut das Beste ist. Um diese Entscheidung zu treffen, gibt es verschiedene Maße: z. B. die Entropie, den Gini-Index oder den Klassifikationsfehler bzw. die *Accuracy*. Die Beispiele werden in Knoten zugeordnet. Ein Attribut ist geeignet, wenn der zugehörige Knoten m rein ist, d.h. wenn alle Instanzen, die eine bestimmte Verzweigung wählen, einer Klasse C_i zugehörig sind. Die Schätzung der Wahrscheinlichkeit der Klasse C_i lautet:

$$\hat{p}\left(C_i|m\right) \equiv p_m^i = \frac{N_m^i}{N_m}.$$

N_m entspricht der Anzahl aller Beispiele im Knoten m, N_m^i bezeichnet die Anzahl der Beispiele in Knoten m, die sich in Klasse C_i befinden. Der Knoten m ist rein, wenn $p_m^i \forall i \in \{0, 1\}$. Das Ergebnis ist 0, wenn keine Instanz, die m erreicht hat, zu der Klasse C_i gehört und 1, wenn alle erreichenden Instanzen zur Klasse C_i gehören [4]. Es gibt verschiedene Algorithmen zur Entscheidungsfindung bezüglich der genutzten Attribute in einem Baum. Zu nennen sind hier CART (*classification and regression tree*), ID3 (*Iterative Dichotomiser 3*) und C4.5 [11]. Wobei letzterer eine Erweiterung von ID3 ist. Alle Algorithmen gehören zu den sogenannten TDIDT-Verfahren (*Top-Down Induction of Decision Trees*) und bauen den Entscheidungsbaum schrittweise auf [3, 11]. Der CART Algorithmus wurde 1984 von Leo Breiman vorgestellt und hat als Ausgabe, ebenso wie ID3, einen Binärbaum.

Ziel des Algorithmus ist es eine optimale binäre Trennung zu finden.

Algorithmus 2 : ID3

```
1 if alle Elemente aus Datenmenge 𝒟 zur Klasse y gehören then
2     Konstruiere Blatt mit Klasse als Label
3 else
4     Wähle Merkmal mit höchstem Information Gain x_i
5 for alle Werte von x_i do
6     Konstruiere Teilbäume rekursiv
7 Konstruiere Knoten mit Label x_i und hänge erzeugte Teilbäume an
```

Die Auswahl der genutzten Attribute erfolgt mithilfe des Informationsgehalts (*information gain*). Dieser beschreibt die Auftrittswahrscheinlichkeit eines Ereignisses. Tritt ein Ereignis mit einer Wahrscheinlichkeit von $p_i = 1$ auf, dann ist der Informationsgehalt 0. Je kleiner die Wahrscheinlichkeit ist, dass ein Ereignis eintritt, desto größer ist der Informationsgehalt. Um also das nächste geeignete Attribut zu finden, muss der Informationsgehalt maximiert werden. Wie schon erwähnt, gibt es noch andere Maße, um den Gain zu berechnen (z. B. Gini-Index), auf die hier aber nicht weiter eingegangen wird.

Um während des Erstellens des Entscheidungsbaums Overfitting zu vermeiden, kann man ihn mithilfe von *Pruning* oder *Pre-Pruning* verkleinern [12]. Overfitting kann durch *Noise* (falsche Messwerte oder Datensätze) entstehen und generiert zu viele Abzweigungen innerhalb des Baums, womit dieser zu groß wird. Pruning bedeutet, dass der Baum zurechtgestutzt wird. Das *Pre-Pruning* geschieht während der Baum erstellt wird und soll eine vollständige Abarbeitung des gegebenen Trainingsdatensatz verhindern. Hierfür werden Stopp-Kriterien in den Algorithmus eingebaut, damit der Baum von Anfang an nicht zu groß werden kann. Ein Stopp-Kriterium könnte z. B. ein nicht erreichter Wert des Informationsgehalts sein. Problematisch ist es, dass durch das Pre-Pruning ein Algorithmus vorzeitig abbrechen kann und die Erzeugung des Baumes nicht beendet wird.

Pruning wird am häufigsten eingesetzt, um die Größe eines Baums zu verringern. Bei dieser Methode werden Teilbäume und innere Knoten durch Blätter ersetzt. Wenn ein gegebener Baum übersättigt (overfitted) ist, kann Pruning dazu beitragen die Güte für noch nicht genutzte Testdaten zu verbessern. Der Vollständigkeit halber soll hier erwähnt werden, dass es zwei verschiedene Pruning-Methoden gibt: das *Bottom-Up Pruning* (von den Blättern zur Wurzel) und das *Top-Down Pruning* (von der Wurzel zu den Blättern) [12]. Um eine ausreichende Basis für die Auswertung der Experimente zu bekommen, werden keine einzelnen Entscheidungsbäume genutzt, sondern ein Ensemble von Bäumen: das bereits erwähnte Verfahren Random Forest.

2.2.2 Support Vector Machine

Das zweite zu nennende Verfahren ist die *Support Vector Machine* (Stützvektormaschine oder kurz SVM). Dieses ist aus der statistischen Lerntheorie [13] hervorgegangen, welche beschreibt, wie aus einer gegebenen Menge von Beispieldaten auf kommende, aber ungesehene, Beispiele geschlossen werden kann [2]. Das Verfahren hatte seinen Durchbruch in den 90er Jahren des 20. Jahrhunderts. SVMs bieten die Möglichkeit, Lösungen für zwei-Klassen Klassifikationsprobleme zu finden. Das Ziel einer SVM ist das Erkennen einer Klassifikationsregel anhand einer Menge M von gegebenen Beispielen. Diese Trainingsbeipiele lassen sich wie folgt definieren: $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \,|x_i \in \mathcal{X}, y_i \in \{-1, 1\}\}$ mit dem Merkmalsvektor $\boldsymbol{x} \in \mathcal{X}$, welcher alle Merkmalswerte zusammenfasst und der Klasse $y \in Y$ mit -1, wenn das Beispiel negativ ist und $+1$ bei einem positiven Beispiel. SVMs werden häufig bei einer großen Menge von Merkmalen eingesetzt, welche ganz unterschiedlicher Natur sein können, z. B. Bildpixel bei der Bilderkennung, Wortvorkommen in einem Text oder Merkmale zum Erkennen eines handschriftlichen Buchstabens. Die einfachste Form des Verfahrens ist eine lineare Klassifikationsregel, bei der die Daten durch eine *Hyperebene [14]* in zwei Bereiche eingeteilt werden. Dazu ist es notwendig, dass die Daten in einem Vektorraum repräsentiert werden, in welchem es dann lineare Entscheidungen gibt. Die gegebenen Regeln haben dann die Form $h(\boldsymbol{x}) = sign\left(\sum_{1=0}^{n} \boldsymbol{w}_i \boldsymbol{x}_i\right) = sign\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\right)$, mit b als dem Bias (Punkt, an dem die Gerade im Koordinatensystem anfängt) und dem Gewichtsvektor $\boldsymbol{w}$, welcher jedem Merkmal ein Gewicht zuordnet[2]. Ziel ist es nun, den Gewichtsvektor und den Schwellwert so zu wählen, dass die Ungleichungen $y_1\left[\langle \boldsymbol{w}, \boldsymbol{x}_1 \rangle + b\right] > 0, \ldots, y_i\left[\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\right] > 0$ erfüllt sind, womit die Beispiele der richtigen Klasse zugeordnet wurden. Es gibt eine Ungleichung für alle Beispiele und der Wunsch ist es nun, diese Ungleichung für möglichst viele Beispiele zu erfüllen. Anschaulich bedeutet dies, dass die Beispiele nach Erfüllung auf der richtigen Seite der Hyperebene liegen. Die Hyperebene hat die Form $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ und ist im zweidimensionalen Raum eine Gerade. Es gibt jedoch nicht nur eine, sondern mehrere Hyperebenen mit dem gleichen Trainingsfehler (s. Abb. 3). Damit das Optimierungsproblem

$$\beta^* \quad = \quad arg\min_{\beta} \parallel \boldsymbol{\beta} \parallel_2^2 + \sum_{i=1}^{n} L\left(\boldsymbol{\beta}, \boldsymbol{x}_i, y_i\right)$$

mit der Verlustfunktion $L\left(\boldsymbol{\beta}, \boldsymbol{x}, y\right) = \max\left(0, 1 - y \cdot \left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\right)\right)$ aus Definition 3 und $\boldsymbol{\beta} = \langle \boldsymbol{w}, b \rangle$ eindeutig lösbar ist, wird also, wie in der Formel zu sehen, die Hyperebene mit der kleinsten Norm gesucht. Das Lösen des Optimierungsproblems, führt zu der, durch β^* spezifizierten, einzig optimalen Hyperebene, da die L_2- Norm streng konvex ist.

Häufig kommt es bei realen Daten vor, dass sich die Datenmenge D nicht direkt mithilfe linearer Methoden klassifizieren lässt. Für einen solchen Fall gibt es die sogenannten

[2] $sign$ bezeichnet das Vorzeichen einer Zahl, d.h. +1 oder -1.

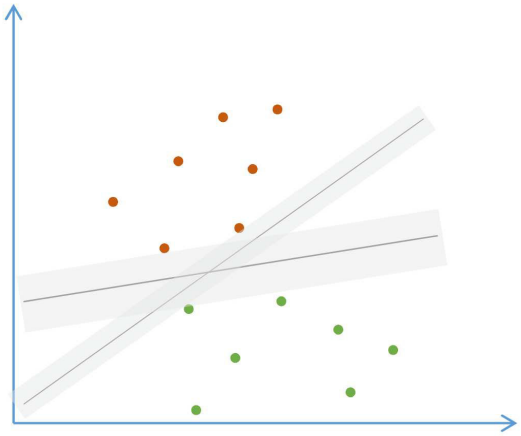

Abbildung 3: *Hyperebene einer SVM. Die Abbildung zeigt zwei mögliche Hyperebenen für die gegebenen Datenpunkte. Der Bereich um die Linien wird als* margin *bezeichnet.*

Kernelmethoden. Das ist eine Klasse von Algorithmen die sich des Kerneltricks bedienen. Hier verwendet man einen Kernel, um die Daten auf eine höhere Dimension (engl. *feature space*) abzubilden und sie damit dann separieren zu können. Es wird als Trick bezeichnet, weil das Mapping $\phi(x)$ (also die Zuordnung der Daten in die höhere Dimension) selbst niemals berechnet werden muss, da man mithilfe der Kernelfunktion $K(x, y) =< \phi(x), \phi(y) >$ den Algorithmus, der die Daten klassifizieren soll, einfach auf eine höhere Ebene legen kann. Die SVM arbeitet mit einem solchen Kernel, um eine Hyperebene in dem neu erzeugten Raum durch die Daten legen zu können. Allgemein ist eine Abbildung $K : X \times X \to \mathbb{R}$ ein Kernel, wenn es einen Prähilbertraum $H\,(< \cdot, \cdot >)$ mit dem feature space H, und eine Abbildung $\phi : X \to H$ gibt, wobei X den Eingaberaum und ϕ die sogenannte *feature map* darstellt. Gesucht ist also eine Funktion, die die Ursprungsdaten in einen Raum transferiert, in dem die Daten linear klassifiziert werden können. Die SVM nutzt verschiedene Arten von Kernel. Zum einen den linearen Kernel mit $K(x, y) =< x, y >$, der die Daten nicht in einen feature space verschiebt. Er wird genutzt, wenn die Anzahl der Feature $(x_1, x_2, \ldots, x_n)$ groß, die Anzahl der Daten im Trainingsset klein ist und wenn die sich D direkt klassifizieren lässt, ohne dass man ein ϕ suchen muss (s. Abb. 4). Der *radial-base-function-Kernel* (RBF-Kernel) $K(x, y) = \exp\left(-\frac{||x-y||^2}{\gamma}\right)$ ist eine Abwandlung des linearen Kernel und wird auch Gaussian Kernel genannt. Er wird genutzt, wenn $x^n \in \mathbb{R}^n$ und die Anzahl der Trainingsdaten n so groß ist, dass sie nicht einfach separiert werden können. Da ϕ unendlich dimensional ist, kann es hier nicht explizit berechnet werden. Aufgrund dessen muss hier der Kerneltrick verwendet werden, da eine Berechnung unendlich lange dauern würde. Ein Kompromiss zwischen dem linearen und dem RBF-Kernel ist der polynomielle Kernel $K(x, y) =< x, y >^d$. Dieser wird häufig bei normalisierten Trainingsdaten genutzt. Hierbei kann ϕ explizit berechnet und Abhängigkeiten zwischen den Attributen berücksichtigt werden. Kurz gesagt, die Daten werden mit ϕ in einen höherdimensionalen Raum projeziert [15], [16]. Außer der SVM gibt es noch andere, ähnliche Kernelmaschinen, wie z. B. die unüberwachte one-class SVM oder den pro-

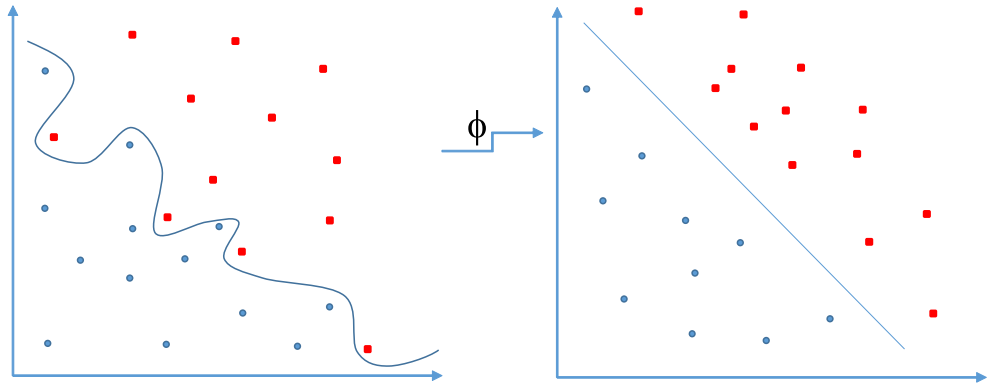

Abbildung 4: *Feature-Map ϕ. Die Grafik soll veranschaulichen, was die Aufgabe von ϕ ist. Auf der linken Seite ist der ursprünglich gegebene Datensatz zu sehen, welcher nicht linear separiert werden kann. Mithilfe von ϕ werden die Datenpunkte nun in einen höherdimensionierten Raum gebracht, wo sie linear klassifiziert werden können.*

babilistischen Gaußschen Prozess (benannt nach Johann Carl Friedrich Gauß (1777 - 1855)) [17], die aber nur erwähnt und nicht weiter ausgeführt werden sollen.

2.2.3 Probabilistische Modelle

Eine sehr allgemeine Art des überwachten Lernens, ist das Lernen der Wahrscheinlichkeitsverteilung $p_\theta(y|x)$ mit $\boldsymbol{\theta} = \min_\theta \prod_{i=1}^{n} p(y^i|x^i)$. Dieses Modell ist durch $\boldsymbol{\theta}$ parametrisiert. Dabei sucht man die Parameter, die den gesamten Daten die größte Wahrscheinlichkeit zuordnen, auch *Maximum-Likelihood-Schätzung* genannt. Man sucht also das p_θ, das die Wahrscheinlichkeit von $\mathcal{D}$ maximiert. Wenn so ein p_θ existiert, dann wird $\boldsymbol{\theta}$ als *Maximum-Likelihood-Schätzer* für $\mathcal{D}$ bezeichnet und oft auch $\hat{\boldsymbol{\theta}}$ genannt [6]. Die Klasse der probabilistischen Modelle ist sehr vielfältig. In dieser Arbeit wird zur Repräsentation ein einfaches probabilistisches Modell genutzt: das Verfahren *Naive Bayes*.

Der naive Bayes Klassifizierer basiert auf dem Theorem von Bayes, 1763 von Thomas Bayes entwickelt, und ist aufgrund seiner guten Erkennungsrate und der schnellen Berechenbarkeit sehr beliebt. Zudem kann er Datensätze klassifizieren, die sowohl numerische als auch diskrete Werte besitzen.

Definition 4. Satz von Bayes. Mithilfe des *Satz von Bayes* lassen sich Berechnungen bedingter Wahrscheinlichkeiten durchführen. Mit einer bedingten Wahrscheinlichkeit kann man vorhandene Informationen zum Vorteil verwenden. Sind die zwei Ereignisse A und B gegeben, dann ist die bedingte Wahrscheinlichkeit definiert durch:

$$p(A|B) = \frac{p\,(A)\,p\,(B|A)}{p\,(B)}$$

mit $p\,(A|B)$ als *a-posteriori-Wahrscheinlichkeit* und $p\,(A)$ als *a-priori-Wahrscheinlichkeit.* Unter der a-posteriori-Wahrscheinlichkeit versteht man die

Wahrscheinlichkeit nachdem B eingetreten ist. Die a-priori-Wahrscheinlichkeit beschreibt die Anfangswahrscheinlichkeit des Ereignisses A ([18]).

Das Verfahren Naive Bayes liefert in manchen Gebieten ähnlich gute Ergebnisse wie Entscheidungsbäume. Wenn man dieses Verfahren auf mehrere Variablen verallgemeinert, dann nennt man es auch Conditional Random Field (CRF) [19]. Wenn es keine Beobachtungen gibt und somit nur $p(y)$ geschätzt wird, dann wird die Methode Markov Random Field (MRF) genannt. Die Aufgabe des Klassifikators ist die Vorhersage der wahrscheinlichsten Klasse für eine neue Instanz x. Eine Instanz ist in diesem Fall ein neuer Datensatz und wird folgendermaßen dargestellt $x = (a_1, a_2, ..., a_n)$, beschreibt also die Verbindung verschiedener Attributwerte [20]. Die Daten können sowohl numerisch als auch diskret oder auch innerhalb eines Datensatzes gemischt sein (s. Tabelle 2). Bei diskreten Werten werden die einzelnen Einträge einfach abgezählt und daraus die entsprechende Wahrscheinlichkeit berechnet. Handelt es sich um numerische Werte, dann wird in der Regel eine Normalverteilung angenommen und man benötigt den entsprechenden Mittelwert und die Standardabweichung.

Die Grundannahmen des Verfahrens sind, dass jedes Attribut nur vom Klassenattribut abhängt und alle Attribute gleich wichtig sind. Weiterhin nimmt man an, dass die Attribute untereinander statistisch unabhängig sind, sofern das Label bekannt ist. Das heißt, wenn der Wert eines Attributs bekannt ist, dann sagt dies nichts über den Wert eines anderen Attributs aus. Diese Tatsache macht die Einfachheit von Naive Bayes aus. Zu erwähnen ist, dass diese Unabhängigkeitsannahme in der Realität niemals stimmt, aber sie ist wichtig, um das Verfahren ausführen zu können und eine geringe Komplexität der Methode zu garantieren. Die Klassifikation wird durch

$$y_{NB} = arg\max_y p(y) \prod_{i=1}^{n} p(a_i|y)$$

erreicht. Voraussetzung hierfür ist die stochastische Abhängigkeit zwischen y und den a_i.

Beispiel. Als Beispiel sind die Daten in Tabelle 2 gegeben und es soll nun eine neue Instanz $< sonnig, 20, 90, stark >$ zugefügt werden. Die Frage ist nun, welcher Klasse diese Instanz zugeordnet wird ($Golf = ja$, $Golf = nein$). Also wird y_{NB} benötigt.

$$\begin{aligned} p(Golf = ja) &= \tfrac{9}{14} &= 0,64 \\ p(Golf = nein) &= \tfrac{5}{14} &= 0,36 \\ p(Wind = stark|Golf = ja) &= \tfrac{3}{9} &= 0,33 \\ p(Wind = stark|Golf = nein &= \tfrac{3}{5} &= 0,60 \\ &\vdots \end{aligned}$$

Nach dem gleichen Schema werden noch die Werte für

$$p(Vorhersage = sonnig|Golf = ja), P(Vorhersage = bedeckt|Golf = ja), usw.$$

Vorhersage	Temperatur in C°	Luftfeuchtigkeit in %	Wind	Golf?
sonnig	29,4	85	schwach	nein
sonnig	26,7	90	stark	nein
bedeckt	28,3	78	schwach	ja
regnerisch	21,1	96	schwach	ja
regnerisch	20	80	schwach	ja
regnerisch	18,3	70	stark	nein
bedeckt	17,8	65	stark	ja
sonnig	22,2	95	schwach	nein
sonnig	20,6	70	schwach	ja
regnerisch	23,9	80	schwach	ja
sonnig	23,9	70	stark	ja
bedeckt	22,2	90	stark	ja
bedeckt	27,2	75	schwach	ja
regnerisch	21,7	80	stark	nein

Tabelle 2: *Beispieldaten zur Erklärung des naiven Bayes Algorithmus. In dieser Tabelle sind Trainingsdaten mit numerischen sowie diskreten Werten aufgezeigt, die darstellen, ob man an einem Tag mit den jeweiligen Bedingungen Golf spielen sollte oder nicht.*

berechnet.

Die Berechnung der benötigten numerischen Werte unterscheidet sich von dem oben beschriebenen Vorgehen. Hierfür braucht man die Dichtefunktion

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

der Normalverteilung. Und dafür wiederum, um $f(x)$ auszurechnen, den Mittelwert

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

und die Standardabweichung

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$$

um damit die Dichte des jeweiligen Wertes zu berechnen.

So kommt man auf die notwendigen Werte von $p(Temperatur = 20|ja) = 0,129$, $p(Luftfeuchtigkeit = 90|ja) = 0,045$ und deren entsprechenden Werte für den Zustand $Golf = nein$.

Um nun die neue Instanz einer Klasse zuordnen zu können, muss noch die Wahrscheinlichkeit für die gesetzten Zustände in den einzelnen Klassen berechnet werden:

$$\begin{aligned} p\,(ja)\,P(sonnig|ja)p(20|ja)p(90|ja)p(stark|ja) &= 0,0012 \\ p\,(nein)\,p(sonnig|nein)p(20|nein)p(90|nein)p(stark|nein) &= 0,0016 \end{aligned}$$

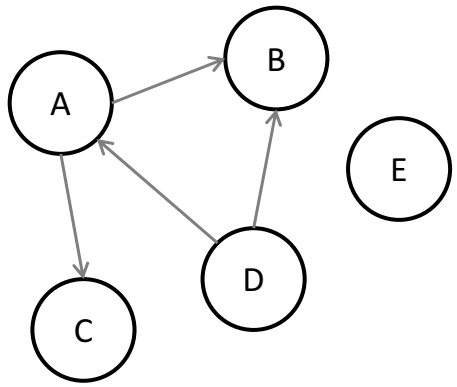

Abbildung 5: *Grafisches Modell. Dargestellt ist ein grafisches Modell mit gegebenen Abhängigkeiten. Knoten A ist abhängig von D, Knoten B von Knoten A und D. Der Knoten C ist abhängig von Knoten A. Da Knoten E keine Kante besitzt, ist er von keinem anderen Knoten abhängig.*

Ein Vergleich der beiden Wahrscheinlichkeiten zeigt, dass der Algorithmus die neue Instanz in die Gruppe $Golf = nein$ zuordnet, da $0,0016 > 0,0012$. Wenn es nun eine neue Instanz gibt, bei der ein Attributwert nicht in jedem Klassenwert vorkommt, dann ist es notwendig, einen kleinen Wert zu den Wahrscheinlichkeiten zu addieren, da man sonst vor dem Problem stände, dass es eine Wahrscheinlichkeit gleich null gibt (additive Glättung, *engl.: additive smoothing*).

Naive Bayes gehört zu der Gruppe der grafischen Modelle, wie auch, das in der Einleitung 1 erwähnte, MRF. Grafische Modelle bringen Graphentheorie und Wahrscheinlichkeitstheorie zusammen und repräsentieren eine Menge bedingter Unabhängigkeiten zwischen einzelnen Zufallsvariablen. Wie auch ein normaler Graph verfügen sie über Knoten V und Kanten E und lassen sich durch $G = (V, E)$ mit $V = \{1, 2, 3, \ldots, m\}$ und $E \subset V \times V$ darstellen. Jede Kante E besteht aus einem Paar von Knoten $(s, t \in V)$. Existiert die Kante (s, t) so bedeutet dies, dass s und t von einander abhängig sind. Gibt es keine Kante, besteht also auch keine Abhängigkeit, vorausgesetzt die Werte der benachbarten Knoten sind bekannt [21]. Ein Beispiel ist in Abbildung 5 gegeben, dort ist $p(A, B, C, D, E) = p(C|A)p(B|A, D)p(A|D)p(D)p(E)$.

2.2.4 Lineare Regression

Die lineare Regression ist ein Spezialfall der Regressionsanalyse, die, ganz allgemein formuliert, Beziehungen zwischen numerischen Variablen darzustellen versucht. Der Begriff Regression stammt aus dem 19. Jh. und wurde von Sir Francis Galton (1822 - 1911) geprägt[3]. Der Wissenschaftler beschäftigte sich mit der Vererbung des Menschen und versuchte die Frage zu beantworten, inwieweit gewisse Merkmale und Eigenschaften der

[3] Die Idee geht auf Johann Carl Friedrich Gauß zurück.

Eltern an die Kinder weitergegeben werden. Um die Frage zu beantworten, sammelte Galton Daten bezüglich der Körpergröße von Eltern und Kindern und untersuchte den Zusammenhang der Größen. Dafür benötigte er die Durchschnittsgröße der Eltern, wo die Daten zunächst normalisiert werden mussten, indem er die Körpergröße der Frauen mit einem bestimmten Faktor multipliziert hat. Er stellte fest, dass die Größe der Kinder fast einer Normalverteilung gleich kam und es eine konstante Varianz zwischen den Familien gab. Er nahm die Durchschnittsgröße der Kinder und erhielt eine Gerade mit der, visuell bestimmten, Steigung von 2/3. Des Weiteren fand er heraus, dass bei überdurchschnittlich großen oder kleinen Eltern die Kinder im Verhältnis entsprechend kleiner oder größer waren, weshalb er die „Regression zum Mittelwert“ [22] bestimmte. Zusätzlich stellte er eine Regressionsgerade, bestimmt durch die Durchschnittsgröße der Kinder, auf. Nach heutigem Verständnis versuchte Galton mit den Regressoren x (Größe der Eltern) die abhängige Variable y (Größe der Kinder) zu bestimmen. Dieses ist das einfachste Regressionsmodell: $y = \beta_0 + \beta_1 x + \epsilon$, bei dem der Einfluss der Regressionsparameter (β) linear ist. Die Parameter werden mithilfe der Methode der kleinsten Quadrate (*least squares*) geschätzt. Angenommen, man hat einen Datensatz $D = \left\{ (y,x)^{(1)}, (y,x)^{(2)}, \ldots, (y,x)^{(n)} \right\}$ mit

$$\begin{aligned} y^{(1)} &= \beta_0 + \beta_1 x^{(1)} \\ y^{(2)} &= \beta_0 + \beta_1 x^{(2)} \\ &\vdots \\ y^{(n)} &= \beta_0 + \beta_1 x^{(n)}. \end{aligned}$$

Um an die Parameterkoeffizienten (β) zu gelangen, muss man nun

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{n} \left(y^{(i)} - \left(\beta_0 + \beta_1 x_1^{(1)} \right) \right)^2$$

ausrechnen. Zu erwähnen ist noch, dass die Regressoren x_i nicht genauso verwendet werden müssen, wie sie im Datensatz vorkommen. Das heißt, weder in der gleichen Reihenfolge, noch in dem gleichen „Zustand“. Wenn man z. B. eine Zeile aus einer Datenbank bekommt, bei der man nicht alle Daten benötigt, sondern nur bestimmte x, dann kann man eine Funktion $\phi(x)$ nutzen, um die Daten in einen anderen Zustandsraum zu transferieren, um möglicherweise einen besseren Schätzer für y zu erhalten. Ohne das Ergebnis der linearen Regression dadurch zu verändern.

In den nächsten Abschnitten werden die beiden unüberwachten Lernverfahren k-means und DBSCAN erläutert.

2.2.5 k-means

Ein Verfahren des unüberwachten Lernens ist k-means, das zur Clusteranalyse genutzt wird, indem es sowohl Cluster als auch Clusterzentren finden kann. Es gehört zu den

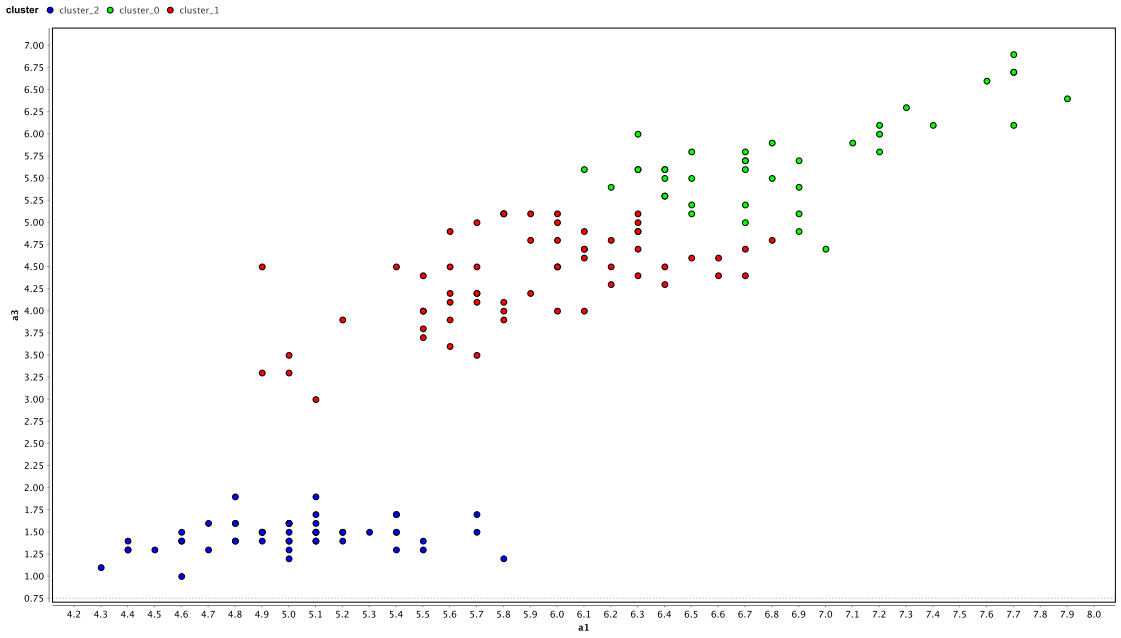

Abbildung 6: *Die Abbildung zeigt den bekannten Iris Datensatz, nachdem er durch k-means geclustert wurde. Die verschiedenen Farben stellen die drei verschiedenen Cluster dar.*

partitioning Algorithms [23], bei denen der Datensatz D in ein Set von Clustern partitioniert wird.

Definition 5. [24] Euklidischer Abstand. Gegeben sind zwei Punkte $\boldsymbol{a} \in \mathbb{R}^n$ und $\boldsymbol{b} \in \mathbb{R}^n$. Der *Euklidische Abstand* dieser Punkte ist definiert als:

$$dist_{Euklid}(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{n}(\boldsymbol{a_i} - \boldsymbol{b_i})^2} = \| \boldsymbol{a} - \boldsymbol{b} \|_2$$

Entspricht also dem direkten Weg zwischen den beiden Punkten.

Für gewöhnlich wird der Algorithmus angewendet, wenn die Variablen alle abzählbar sind und als Abstandsmaß die quadrierte Euklidische Distanz genutzt wird. Man muss vorher wissen wieviele Cluster gefunden werden sollen, da dies ein Eingabeparameter des Algorithmus ist. Innerhalb der Experimente (s. Kapitel 5) werden als Abstandsmaße sowohl der Euklidische Abstand als auch die Manhattan-Distanz genutzt.

Der wohl bekannteste Algorithmus zur Lösung von k-means basiert auf einem Algorithmus von Stuart Lloyd und hat drei Schritte. Der erste Schritt besteht darin, k zufällige Clusterzentren aus dem Datensatz zu bestimmen. Diese initialen Zentren sind für gewöhnlich willkürlich aus den Trainingsdaten gewählt. Im zweiten Schritt werden die einzelnen Objekte dem Clusterzentrum zugeordnet, welchem sie anhand des Euklidischen Abstands am nächsten sind. In Schritt drei werden aus dem Ergebnis der Zuordnung neue Clusterzentren berechnet, indem der Mittelwert der bisher sortierten Punkte gebildet wird. Die Schritte zwei und drei werden so oft wiederholt, bis das Ergebnis des

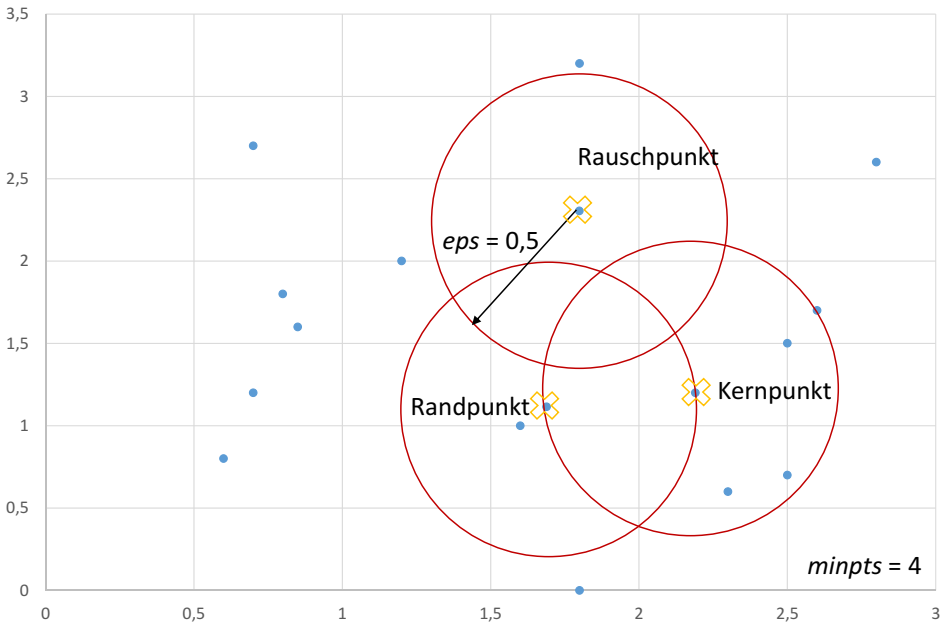

Abbildung 7: *DBSCAN. Darstellung der verschiedenen Arten von Punkten, die von DBSCAN definiert und genutzt werden.*

Algorithmus konvergiert [3, 25]. In Abbildung 6 sieht man das Ergebnis des Verfahrens. Der Datensatz hat vier verschiedene Parameter, in denen $k = 3$ verschiedene Cluster gefunden wurden.

2.2.6 DBSCAN

DBSCAN (*density-based spatial clustering of applications with noise*) ist ein dichtebasiertes Verfahren zur Clusteranalyse. Noise wird hierbei durch die Punkte des Datensatzes, die keinem der vorhandenen Cluster $C_1, ..., C_k$ zugeordnet werden können, definiert und als separates Cluster ausgegeben. Der Algorithmus teilt die Datenpunkte in Regionen mit einer hohen Dichte und einer weniger hohen Dichte ein. Der Vorteil von DBSCAN gegenüber k-means besteht darin, dass es nicht notwendig ist, die Anzahl der Cluster vorzugeben. Diese wird vom Algorithmus selbstständig erkannt. Ein weiterer Vorteil ist, dass man eventuell vorhandenes Problemwissen, also spezielles Wissen über Nähe zwischen Datenpunkten, miteinfließen lassen kann. Auch etwaige Rauschpunkte (Noise) können erkannt, als solche identifiziert und zurückgegeben werden.

Der Algorithmus benötigt den Wert von ϵ und eine Anzahl von *minpts* sowie einen beliebigen Startpunkt q, um die Cluster innerhalb der gegebenen Daten zu berechnen. Es gibt drei verschiedene Arten von Punkten: *Kernpunkte*, *Randpunkte* und *Rauschpunkte*. Die Anzahl der *minpts* ist notwenig, um zu definieren, wann ein Punkt ein Kernpunkt ist. Nämlich wenn dieser mindestens die Anzahl der *minpts* durch ϵ erreichbare Nachbarn hat. Ein Randpunkt liegt innerhalb des Radius ϵ eines Kernpunktes, hat

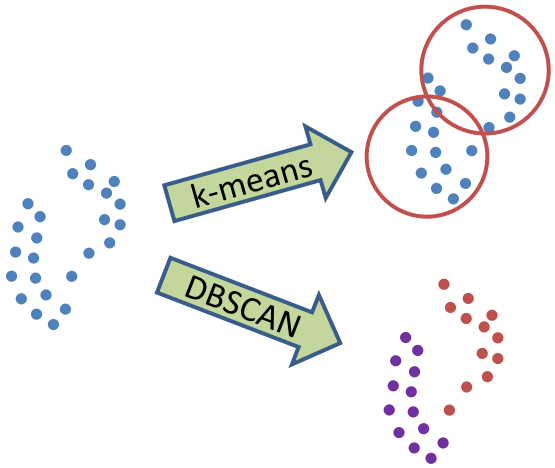

Abbildung 8: k-*means vs. DBSCAN. Links sind die gegebenen Datenpunkte. Während* k-*means, mit* $k = 2$ *(oben) zwei elliptische Cluster erkennt, bestimmt DBSCAN (unten) zwei nicht symmetrische Formen.*

aber nicht genügend Punkte innerhalb des eigenen Radius, um ein Kernpunkt zu sein. Rauschpunkte liegen außerhalb der Umgebung eines Kernpunktes und gehören keinem Cluster an. Abbildung 7 zeigt die verschiedenen Arten der Punkte, die durch den Abstand $\epsilon = 0,5$ und die Anzahl der $minpts = 4$ bestimmt werden. Der Algorithmus beginnt mit einem beliebigen Punkt q und sucht nun alle Punkte, die den Abstand $< \epsilon$ haben. Danach können die Rauschpunkte gelöscht und die Kernpunkte mit einer Kante verbunden werden. Als letzter Schritt werden die Randpunkte dem nächstliegenden Cluster zugeordnet [23]. Der Abstand zwischen zwei Punkten innerhalb eines Clusters ist durch ϵ definiert. Ein Cluster kann auch aus einer Kette von Punkten bestehen, bei der der erste und der letzte Punkt einen beliebigen Abstand haben können. Diese Punkte sind dann dichteverbunden.

Innerhalb der Arbeit werden zwei verschiedene Abstandmaße verwendet. Das ist zum einen das Euklidische Abstandsmaß (s. Def. 5), vorzustellen als Luftlinie, und zum anderen die Manhattan-Distanz, auch bekannt als Taxi-Metrik. Taxi-Metrik deshalb, weil man sich vorstellen kann, dass man sich nur über ein gegebenes Straßennetz zwischen zwei Punkten bewegen darf.

Definition 6. [24] Mahattan-Distanz. Der Name der *Manhattan-Distanz* stammt von der Vorstellung, dass man sich nur innerhalb eines Straßengitters bewegen kann (z. B. eines wie in Manhattan), bei dem die Größe der einzelnen Blöcke identisch ist. Die Entfernung zwischen zwei gegebenen Punkten $\boldsymbol{a} \in \mathbb{R}^n$ und $\boldsymbol{b} \in \mathbb{R}^n$ ist durch

$$dist_{Manhattan}(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} |\boldsymbol{a}_i - \boldsymbol{b}_i| = \|\boldsymbol{a} - \boldsymbol{b}\|_1 ,$$

gegeben. Zwischen den beiden Punkten kann es auch mehrere kürzeste Wege innerhalb dieser Metrik geben (s. Abb. 9).

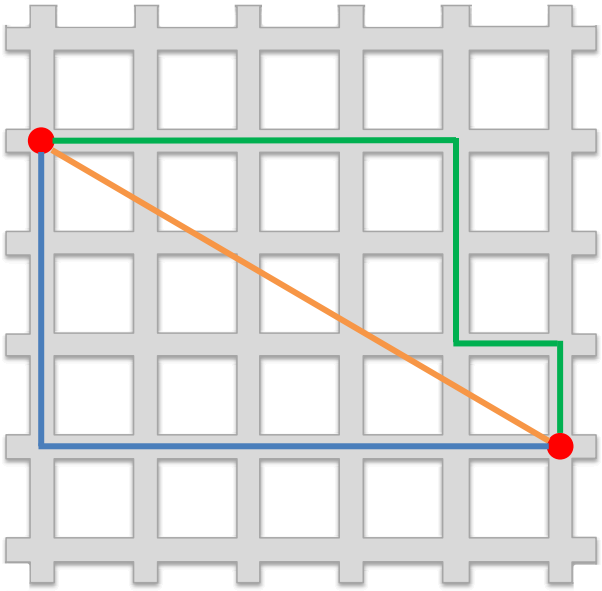

Abbildung 9: *Abstandsmaße. Die direkte Verbindung der beiden Punkte (orangene Linie) ist der Euklidische Abstand. Die anderen beiden farbigen Linien stellen den Abstand zwischen den beiden roten Punkten anhand der Manhattan-Distanz dar. Die Wege der Manhattan-Distanz haben alle die gleiche Länge.*

Zusammenfassend lässt sich sagen, dass DBSCAN in der Lage ist, auch verschiedene Formen zu erkennen, während k-means nur elliptische Cluster erkennen kann (s. Abb. 8).

2.3 Verwandte Arbeiten

In Abschnitt 2.1 wurde kurz das MDL-Prinzip vorgestellt. Dieses kann generell als Hintergrundidee dieser Arbeit betrachtet werden. Natürlich beschäftigen sich auch andere Arbeiten mit MDL und somit gibt es unterschiedliche Ansätze die Idee der Komprimierung bzw. der Kompression von Daten umzusetzen. Jedoch gibt es bislang keinen Ansatz mit der gleichen Herangehensweise, wie sie hier vorgestellt wird. Nachfolgend werden einige dieser Arbeiten angesprochen.

Der größte Unterschied zwischen den hier vorgestellten und den bislang bekannten Ansätzen besteht darin, dass sich die vorliegende Arbeit mit der Kompression des Modells und nicht mit der Kompression der Daten auseinandersetzt.

Aus [26] geht hervor, dass je größer das Modell ist, die Daten umso besser komprimiert werden. Das heißt die Modellgröße wird nicht optimiert, sondern sollte eher möglichst groß sein, was ganz im Gegensatz zu den Zielen und Ergebnissen der vorliegenden Arbeit steht. Hier ibesteht das Ziel darin, ein möglichst kleines Modell zu erzeugen.

Die Idee, Datenpunkte zu clustern, bevor daraus ein Modell gelernt wird, ist ein weit verbreiteter Ansatz. Es gibt viele verschiedene Arbeiten, die sich damit befassen. Als

Beispiel sei die Idee aus [27] genannt. Hier wird zuerst die Anzahl der Support Vektoren durch clustern derselben verringert und dann das Modell gelernt. Dies ist zwar ähnlich mit den Vorgehensweisen dieser Arbeit, aber dennoch unterschiedlich. Das liegt zum einen daran, dass es keine vorgegebene Anzahl von Support Vektoren gibt und auch die Größe der Daten nicht beschränkt wird, und zum anderen wird auch hier nicht das Modell, sondern die Daten geclustert. Dadurch wird zwar der gegebene Datensatz kleiner, aber das daraus gelernte Modell kann trotzdem eine hohe Komplexität haben, da es genauso viele Parameter besitzen kann wie auf dem Originaldatensatz. Das steht aber mit den Zielen dieser Arbeit im Widerspruch.

Ein weiterer Ansatz ist der des *Exceptional Model Mining* [28]. Dabei wird untersucht, wie sich ein Modell verändert, wenn man nur Kleinigkeiten an den gegebenen Daten ändert. Zum Beispiel wie sich ein von der SVM gelerntes Modell verändert, wenn man einfach einen Teil der Trainingsdaten entfernt. Das Ergebnis ist dann ein sogenanntes *exceptional model*, mithilfe dessen man versucht, auffällige Eigenschaften innerhalb des Datensatzes zu finden.

Bei der Recherche nach ähnlichen Ansätzen, bei denen ebenfalls Parameter und geclustert werden, stößt man häufig auf *Mixture Model Clustering*. Diese erweist sich jedoch als eigenes Clusterverfahren und betrachtet somit ebenfalls keine Modelle in der gleichen Form, wie in dieser Arbeit dargelegt.

Zudem gibt es eine Klasse von regularisierungsbasierten Ansätzen. Auf Regularisierung wird in Abschnitt 3.4 weiter eingegangen. Ein interessanter Ansatz für raum-zeitliche Daten ist in [1] zu finden. Die zugrunde liegende Idee sieht vor, dass ein Modell, welches über ein gewisses Zeitintervall betrachtet wird, aus mehreren kleineren Modellen besteht. Ziel ist es nun, den Unterschied der benachbarten Modelle möglichst gering zu halten, sodass sich am Ende ein möglichst kleines Gesamtmodell ergibt.

Nicht zum Clustering gehörend, aber trotzdem erwähnt werden soll die sogenannte Lasso (*Least Absolute Shrinkage and Selection Operator*) Methode [29]. Diese ist ebenfalls ein Beispiel für einen regularisierungsbasierten Ansatz. Hier liegt der Fokus zwar auf einer sogenannten *feature selection,* bei der das Ziel ist herauszufinden, welche der gegebenen Feature wichtig sind, aber trotzdem findet sich hier gleichzeitig die Kompressionseigenschaft bezüglich des gelernten Modells.

Zusammenfassend kann man sagen, dass es viele verschiedene Ansätze gibt, die Daten zu clustern, jedoch noch keine allgemeine Methode, um dies auf Modellen durchzuführen. Häufig werden die Daten geclustert, um mit einem kleineren Datensatz umgehen zu müssen und um eventuelle Messfehler im Vorhinein zu beiseitigen. Der in dieser Arbeit vorgestellte Ansatz ist mehr im Sinne des MDL-Prinzip, da das Ergebnis ein möglichst einfach zu beschreibendes Modell sein soll.

Nachdem nun ein Überblick der Grundlagen und einzelnen Lernverfahren gegeben wurde, befasst sich das kommende Kapitel mit der grundlegenden Idee dieser Arbeit.

3 Latente Strukturen zur Kompression

Aus Abschnitt 2.2 ist bekannt, dass alle Modelle innerhalb des überwachten Lernens parametrisiert sind und einen Parametervektor $\boldsymbol{\theta}$ haben. Dieser Vektor besteht meist aus reellen Zahlen, z. B. $\mathbb{R}^d$ und kann, je nach Modell und Datensatz, sehr groß werden. Als Beispiel kann man sich vorstellen, dass in biologischen Daten, die vielleicht Gene des Menschen enthalten, d eine Größenordnung von 400.000 haben kann. Aber auch bei kleineren und einfacheren Problemstellungen kann es passieren, dass der Parametervektor lang wird, weil es z. B. bei diskreten Wahrscheinlichkeitsverteilungen viele verschiedene Kombinationsmöglichkeiten gibt. Allerdings sind alle Datensätze, die man untersuchen kann und mit denen sich Modelle schätzen lassen, nur eine Teilmenge aus der Realität und können niemals alle unendlich vielen Daten beinhalten. Auch das populäre Schlagwort *Big Data* vermag nicht genügend Daten zu beinhalten, um alle Unsicherheiten auszumerzen. Als Big Data werden sehr große Datensätze bezeichnet, bei denen es schwierig ist sie, aufgrund der Masse an Informationen oder weil sich die Informationen zu schnell ändern, zu analysieren. Laut IDC (*International Data Corporation*) wurden im Jahr 2012 jeden Tag 2,8 Exabyte (1 Exabyte entspricht 10^9 Gigabyte) Daten erzeugt, im Jahr 2015 ist diese Zahl schon auf 8,5 Exabyte gestiegen und wird voraussichtlich im Jahr 2020 40 Exabyte betragen [30]. Aber auch diese Anzahl von Daten muss nicht unbedingt ausreichen, um hochdimensionale Modelle darstellen zu können, da auch diese Menge endlich ist. Zusammenfassend kann man sagen, dass es bei sehr großen Datensätzen auch sehr große Modelle als Ergebnis geben kann.

Angenommen, die Parameter der Vektoren sind bekannt und hinreichend interpretierbar. In der Bayesschen Statistik geht man davon aus, dass die geschätzten Parameter Zufallsvariablen sind und aus dem Zufall der Daten hervorgehen. Im Mittelpunkt steht hier der Satz von Bayes (s. Def. 4), mit dessen Hilfe man die unbekannten Parameter einer Gleichung schätzen kann. Gegeben sei eine Gleichung $Grösse = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \cdot Alter$. Dann kann man annehmen, dass nicht jeder Wert von $\boldsymbol{\beta}_i$ angenommen werden kann und sich die Werte der Parameter nur in einem bestimmten Bereich bewegen. Wenn wie hier, die Größe eines Menschen ein Parameter in dem Modell ist, so kann man sicher sagen, dass der Parameter nur Werte <2,80 m (der größte Mensch der Geschichte war 2,72 m groß) annehmen kann und im Normalfall natürlich noch kleiner ist. Mithilfe dieser a-priori-Wahrscheinlichkeitsverteilung kann es zu den theoretischen Häufungen aus Abbildung 10 kommen.

Nimmt man also an, dass es diese Verteilung der Parameter gibt, dann kann man weiterhin annehmen, dass die Werte noch dichter zusammenliegen oder eventuell sogar alle Parameter den gleichen Wert haben und man das nur aufgrund einer zu geringen Anzahl an Daten nicht festgestellt hat. Die Tatsache, dass die Parameter so nah beieinander liegen und eben noch nicht den gleichen Wert haben, bezeichnet man auch als Rauschen in den Daten. Dieses Rauschen bewirkt, dass ein falsches Modell gelernt wird, mit der Folge, dass es zu Overfitting (s. Abschnitt 2.2) führen kann. Mit dieser Annahme, dass die Werte der einzelnen Parameter sehr ähnlich und eventuell sogar identisch sind, ist die Gleichheit der Werte die latente Struktur. Ziel ist es nun, mithilfe verschiedener Lernverfahren, diese latente Struktur in den Daten zu finden, um eine

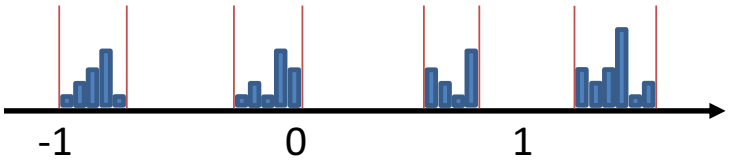

Abbildung 10: *Die Abbildung stellt die eindimensionale Sicht auf einen Parametervektor dar. Die blauen Balken sind die einzelnen Werte des Vektors, die sich, in diesem Fall, in vier verschiedenen Clustern befinden. Je höher der Balken ist, desto mehr Werte befinden sich an der Stelle. Die roten Striche stellen die Begrenzung der einzelnen Cluster dar.*

geringere Komplexität als die des Ursprungmodells zu erhalten.

Der nächste Abschnitt beschreibt das generelle Vorgehen, das in dieser Arbeit genutzt wird und die zentrale Idee. In Abschnitt 3.2 wird erläutert, wie die, schon erwähnte Komplexität, für die einzelnen Lernverfahren berechnet werden kann.

3.1 Vorgehen

Zur Veranschaulichung des prinzipiellen Vorgehens, wie man die latente Struktur finden kann, soll Abbildung 11 dienen. Das überwachte Lernverfahren bekommt einen Datensatz D und schätzt daraus ein Modell mit dem Parametervektor $\boldsymbol{\theta}$. Diesem Parametervektor kann man noch nicht ansehen, welche der Parameter eventuell ähnlich sind und somit in ein Cluster gehören. Man sieht nur die Zahlenwerte und sucht nun eine latente Variable die festlegt, welcher Parameter zu welchem Cluster gehört. Angenommen für jeden Parameter $\boldsymbol{\theta}_i$ gibt es eine zugehörige latente Variable z_i. Sind bei zwei Parametern $\boldsymbol{\theta}_i$ und $\boldsymbol{\theta}_j$ die latenten Variablen die gleichen, also ist $z_i = z_j$, dann bedeutet dies, dass die beiden Parameter zusammen in ein Cluster gehören. Die Aufgabe besteht also darin, alle z_i zu finden, was wiederum identisch zum Vorgehen des Clusterns ist, da i von Cluster C_i der latenten Variable z_i entspricht.

Aufgrund dessen kann man nun auf das Modell aus dem überwachten Lernverfahren ein unüberwachtes Lernverfahren (Clusterverfahren) anwenden. Hier können verschiedene Methoden verwendet werden, z. B. k-means (s. Abschnitt 2.2.5) oder DBSCAN (s. Abschnitt 2.2.6), aber auch andere Verfahren zur Clusteranalyse. Wie z. B. *Expectation-Maximization* [31], hier aber nur am Rande erwähnt. Diese sollen die latenten Strukturen innerhalb des Datensatzes D und die zusammengehörigen Parameter auffinden. Wichtig ist hier zu unterstreichen, dass es sich nur um ein eindimensionales Clustering der Parameter handelt.

Hat man nun die latente Struktur z_i für die einzelnen Parameter herausgefunden, stellt sich die Frage, wie man diese Erkenntnis verwenden kann. Naheliegend ist es, die Werte innerhalb der Cluster zusammenzufassen. Hierfür gibt es bekannte und einfache Aggregationsmöglichkeiten, wie z. B. das Berechnen des Durchschnitts oder das Minimum bzw. das Maximum zu bestimmen. Diese Möglichkeiten werden auch in den Experi-

Abbildung 11: *Ein Überblick der Vorgehensweise dieser Arbeit. Ein überwachtes Lernverfahren lernt aus einem gegebenen Datensatz ein Modell mit dem Parametervektor $\boldsymbol{\theta}$. Auf diesem Modell lernt ein unüberwachtes Lernverfahren ein neues Modell, mit einem neuen Vektor $\hat{\boldsymbol{\theta}}$. Zum Schluss wird die Komplexität der Modelle berechnet und verglichen.*

menten (s. Kapitel 5) verwendet. Auch andere Möglichkeiten sind denkbar, z. B. die Bestimmung des Median. Die Idee ist hierbei, dass ähnliche Werte zu einem Wert zusammengefasst werden, sodass man sich nur noch den Wert selbst und die Information, welcher Parameter diesen Wert besitzt, abspeichern muss. Eventuell stellt sich hierbei heraus, dass alle Parameter gleich sind und das nur aufgrund von zu wenigen Messdaten nicht festgestellt werden konnte. So erhält man einen Schätzer für den tatsächlichen Parameter, wenn man mehr Daten gesehen hätte. Zu beobachten ist hier, dass man bei Verfahren wie k-means, wo man die Anzahl der zu findenden latenten Strukturen im Vorhinein einstellen kann, nur zwischen einer Zahl zwischen 1 und d wählen kann. Denn es kann nicht mehr latente Strukturen im Modell geben, als es Parameter im Ursprungsmodell gab. Selbst wenn das $k > d$ ist, wird es maximal d Cluster geben können, wo jedes Cluster dann nur ein Element hat.

Werden nun alle Werte durch die gewählte Aggregation ersetzt, erhält man einen neuen Parametervektor $\hat{\boldsymbol{\theta}}$, der die latente Struktur hat. Dieser neue Parametervektor ist abhängig von der Anzahl der Aggregationen, also davon, wieviele latente Strukturen gefunden worden sind. Davon ausgehend hat der neue Vektor in den meisten Fällen eine andere Komplexität (s. Abschnitt 3.2) als der Ursprungsvektor θ[4]. In der Regel ist auch die Güte von $\hat{\boldsymbol{\theta}}$ eine andere als bei $\boldsymbol{\theta}$. Deshalb wird hier, wie in Abschnitt 2.2 schon erklärt, sowohl eine Kreuzvalidierung für die Güte, als auch für die Komplexität des neuen Modells durchgeführt, um diese mit Güte und Komplexität des ursprünglichen Modells zu vergleichen.

Die Kreuzvalidierung der Komplexität ist notwendig, da bei einem anderen Datensatz auch andere Modellparameter $\boldsymbol{\theta}'$ geschätzt werden. Aufgrunddesssen ändern sich die Wahrscheinlichkeiten der einzelnen Werte und dadurch auch die Komplexität. Wie genau sich die Komplexität bestimmen lässt, wird im nächsten Abschnitt gezeigt. Durch die verschiedenen Modellparameter unterschiedlicher Datensätze und den damit einhergehenden Änderungen wird die Komplexität mit einer Kreuzvalidierung bestimmt.

[4] Die Komplexität muss nicht in jedem Fall anders sein. Angenommen, die Anzahl der Aggregate g ist gleich der Anzahl der Parameter d, dann wäre die Komplexität des neuen Vektors gleich der Komplexität des alten Vektors.

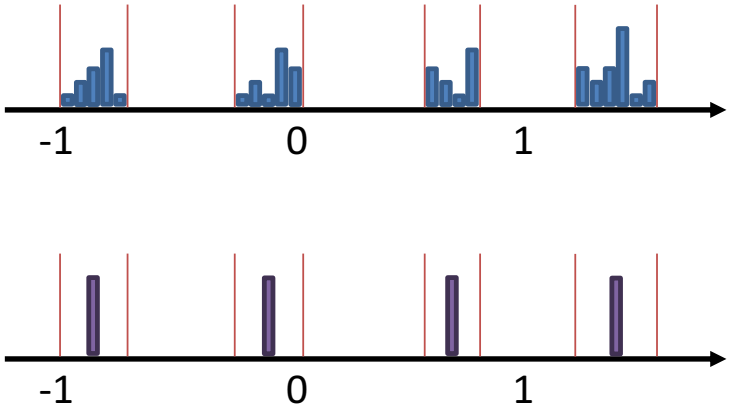

Abbildung 12: *Ähnlich Abbildung 10 ist hier eine eindimensionale Sicht auf einen Parametervektor gegeben. Die blauen Balken im oberen Teil sind die einzelnen Werte innerhalb der Cluster und je höher der Balken, desto mehr Parameter haben den gleichen Wert. Der untere Teil im Bild zeigt die gleichen Cluster, bei denen die Werte jeweils durch den Clustermittelpunkt ersetzt wurden.*

3.2 Berechnung der Komplexität

Die Komplexität eines Modells sagt aus, wie groß dieses ist. Dieses Wissen ist notwendig, um beurteilen zu können, wie sich die Größe des Modells nach der Anwendung des Clusterverfahrens im Gegensatz zum Ursprungsmodell verändert. Im Sinne der Informationstheorie benötigt man ein Alphabet mit Symbolen. Im Folgenden sind die unterschiedlichen Gewichte w_i als Symbole zu interpretieren. Eine zentrale Beobachtung ist, dass, sobald der Parametervektor feststeht, also nach der Kompression, jeder seiner Werte aus einer Menge $W = \{w_1, w_2, \ldots, w_l\}$ ist. Der Parametervektor $\boldsymbol{\theta}$ besteht dann also aus höchstens d verschiedenen Zahlen, wobei die Anzahl der unterschiedlichen Zahlen höchstens l ist und l nur Werte zwischen 1 und d annehmen kann. Ist $l = 1$, so haben alle Parameter den gleichen Wert, bei $l = d$ ist jeder Wert unterschiedlich und es gibt keinen Unterschied zum Ursprungsmodell vor der Kompression. Jeder der Einträge innerhalb des Parametervektors ist eine Zufallsvariable. In der Kodierungstheorie [6] entspricht eine gute Kodierung der kompaktesten Möglichkeit etwas zu verschlüsseln (s. Abschnitt 2.1). Hat man die Anzahl der jeweiligen Symbole innerhalb des Vektors herausgefunden, gibt man dem Symbol, welches am häufigsten vorkommt, den kürzesten Code, dem zweithäufigsten den nächst längeren Code und so weiter. Dadurch garantiert man eine möglichst geringe Länge des kodierten Vektors. Vorausgesetzt wird, dass alle Symbole eindeutig belegt und somit präfixfrei sind, denn dann kann man die Kodierung problemlos wieder rückgängig machen [6].

Es gibt verschiedene Möglichkeiten der Komplexitätsmessung. Die einfachste besteht darin, die Anzahl l der unterschiedlichen Gewichte bzw. Symbole zu zählen. Bei der Regression wäre l also die Anzahl der verschiedenen Parameter. Ähnlich ist es bei Nai-

ve Bayes, wo l der Anzahl der unterschiedlichen Wahrscheinlichkeiten, die vorkommen können, entspricht. Innerhalb eines Entscheidungsbaumes wird die Anzahl der unterschiedlichen Werte an den Verzweigungen gezählt. Bei einer SVM werden die unterschiedlichen Werte der Gewichte, der Stützvektoren α_i und eventuell die der zugehörigen Daten gezählt.

Wie viele verschiedene Parameter es überhaupt gibt, kommt auf das gewählte unüberwachte Lernverfahren an. Bei k-means ist $l \leq k$, da es auch leere Cluster geben und diese nicht mitgezählt werden. Bei DBSCAN kommt es darauf an, ob der Noise-Cluster ebenfalls betrachtet wird oder nicht. Die Experimente in Kapitel 5 werden zeigen, ob es sinnvoll ist, den Noise-Cluster mitzuzählen oder nicht. Damit bekommt man mehr Parameter, also eine größere Modellkomplexität.

Eine andere Möglichkeit die Komplexität zu beurteilen, ist die Entropie (s. Abschnitt 2.1.1, Def. 1). Zur Erinnerung: Die Formel der Entropie lautet:$\mathcal{H}(\boldsymbol{\theta}) - \sum_{w_i} p(w_i) \cdot \log p(w_i)$. Da die genaue Wahrscheinlichkeit $p(w_i)$ nicht bekannt ist, bedient man sich hier eines empirischen Schätzers und zählt wie häufig jeder einzelne Parametervektor vorkommt. Also entspricht die Wahrscheinlichkeit $p(w_i)$ hier $\frac{\#w_i}{d}$, das heißt der Anzahl von w_i geteilt durch die Länge von $\boldsymbol{\theta}$.

Beispiel. In Abbildung 12 soll dargestellt werden, wie die Aggregation eines Parametervektors, auf den schon k-means mit $k = 4$ angewendet wurde, aussieht. Der obere Teil des Bildes zeigt die einzelnen Cluster mit ihren jeweiligen ähnlichen Werten. Jeder blaue Balken entspricht einem Eintrag von $\boldsymbol{\theta}$, entspricht also einem $\boldsymbol{\theta}_i$. Es kann auch vorkommen, dass sich mehrere Blöcke überlagern. Dann sind mehrere $\boldsymbol{\theta}_i$ an der gleichen Stelle. Die Höhe der Balken stellt die unterschiedliche Anzahl der, sich an dieser Position befindenden, $\boldsymbol{\theta}_i$ dar. Im unteren Teil der Grafik wurden alle Werte durch ihren jeweiligen Mittelpunkt ersetzt. Angenommen, es waren ursprünglich 200 Datenpunkte gegeben, so hat man nach der Ersetzung nur noch vier Datenpunkte. Man kann sich vorstellen, dass die Entropie bei Häufigkeiten der einzelnen Werte von $\frac{80}{200}, \frac{50}{200}, \frac{40}{200}, \frac{30}{200}$ geringer ist, als hätte man eine Häufigkeit mit $\frac{1}{200}$, bei der jeder Wert nur einmal vorkommt und man statt vier Punkten wieder die 200 Punkte abspeichern müsste.

Aus Abschnitt 2.1.1 ist bekannt, dass ein Alphabet mit Symbolen benötigt wird, um die Entropie auszurechnen. Wie oben schon erwähnt entsprechen die Gewichte w_i diesen Symbolen und Ziel ist es herauszufinden, wie groß der Parametervektor $\boldsymbol{\theta}$ ist, wenn dieser optimal komprimiert wurde. Also mit einem präfixfreien (dekodierbaren) Code verschlüsselt wurde. Daraus ergibt sich, je kleiner die Entropie eines Modells, desto besser kann $\boldsymbol{\theta}$ komprimiert werden. Zu beachten ist allerdings, dass es auch wichtig ist, die Verlustfunktion zu betrachten, da ein Ergebnis mit einer kleinen Entropie und einer kleinen Genauigkeit (*Accuracy*) schlechter ist, als ein etwas höherer Entropiewert mit einer dafür auch höheren Genauigkeit.

Beide vorgestellten Möglichkeiten sind für alle parametrisierten Modelle gleich anzuwenden. Es gibt eine weitere Möglichkeit, die Komplexität zu messen, indem der Speicherverbrauch des jeweiligen Modells ermittelt wird. Dieser ist für jedes Modell unterschiedlich zu berechnen. Da den einzelnen Verfahren jeweils unterschiedliche Algorithmen

$\boldsymbol{\theta}$	w_i	w_i	w_i	w_k	w_k	w_k	w_i	w_i

$\boldsymbol{\theta}'$	3	w_i	3	w_k	2	w_i

$\boldsymbol{\theta}$	w_i	w_j	w_i	w_j	w_i

$\boldsymbol{\theta}'$	1	w_i	1	w_j	1	w_i	1	w_j	1	w_i

Tabelle 3: *Intervalldarstellung. Die oberen zwei Zeilen der Tabelle zeigen, wie mithilfe der Intervalldarstellung Speicherplatz eingespart werden kann. Anstatt alle Werte einzeln zu speichern, wird erst die Anzahl und dann der entsprechende Wert in die Liste geschrieben. Im unteren Teil ist zu sehen, dass diese Darstellung auch das Risiko von Speicherverbrauch erhöht, wenn es eine ungünstige Anordnung der einzelnen Werte gibt.*

zugrunde liegen, die alle verschiedene Datenstrukturen haben, gibt es keine allgemeine Möglichkeit direkt auszurechnen, wie hoch der Speicherverbrauch bei dem jeweiligen Verfahren ist. Hier muss man für die einzelnen Fälle gesondert überlegen, was abgespeichert werden muss und dann die benötigten Bits zählen.

Ein allgemeiner Ansatz für eine Speicherung von Daten, ist die Intervalldarstellung. Diese lässt sich sinnvoll anwenden, wenn man viele gleiche hintereinander stehende Parameter hat. Tabelle 3 (oben) zeigt beispielhaft, wie sich mithilfe dieser Darstellung Speicherplatz einsparen lässt. Anstatt jedes Zeichen einzeln abzuspeichern, werden gleiche hintereinander stehende Zeichen zusammengefasst, sodass statt der drei Zeichen w_i, w_i, w_i, nur die zwei Zeichen $3, w_i$ abgespeichert werden müssen. Besonders sinnvoll ist dies natürlich, wenn es möglichst lange Reihen gleicher Parameter gibt und wenn nur eine kleine Anzahl unterschiedlicher Parameter vorhanden ist. Statt des Parametervektors $\boldsymbol{\theta}$, speichert man sich den neuen Vektor $\boldsymbol{\theta}'$ ab. Das heißt man hat eine Liste, die zuerst die Anzahl und dann den jeweiligen Parameter angibt: $(\#w_1, w_1, \#w_2, w_2, \ldots, \#w_l, w_l)$. Allerdings kann es hierbei auch dazukommen, dass viel mehr Speicher verbraucht wird als vorher. Wenn z. B. $l = 2$, die beiden unterschiedlichen Parameter jedoch alternieren, hätte man einen doppelt so großen Speicherplatzverbrauch wie zuvor. Die beiden unteren Zeilen in Tabelle 3 zeigen dies. In dem Fall wäre es sinnvoller, die ursprüngliche Darstellung zu wählen und abzuspeichern.

Die Intervalldarstellung wird nun auf alle Verfahren angewendet, muss aber vorher angepasst werden und die jeweiligen Datenstrukturen der einzelnen Verfahren müssen be-

rücksichtigt werden. Diese Darstellung dient dazu den Speicherverbrauch der einzelnen Modelle zu messen. Im Zuge dessen werden die Parameter umsortiert. Allerdings ist es wichtig zu wissen, dass man diese Umsortierung auch wieder rückgängig machen muss und deshalb nicht einfach alles beliebig umstellen kann. Sonst bräuchte man, um den ursprünglichen Zustand wieder herstellen zu können, eine weitere Liste, in der abgespeichert wird, wie die alte Reihenfolge der Parameter war. Und das wäre keine Einsparung des Speicherverbrauchs. Deshalb gilt es, für alle Verfahren Überlegungen anzustellen, um eine möglichst gute Komprimierung durchzuführen, aus der ohne Probleme das Originalmodell rekonstruiert werden kann.

Lineare Regression. Die oben vorgestellte Intervalldarstellung kann immer angewendet werden, wenn die Voraussetzungen erfüllt sind. Zusätzlich ist innerhalb der Regression eine beliebige Parametersortierung möglich, da es hier keine Rolle spielt in welcher Reihenfolge sie stehen. So könnte man erzwingen, dass die gleichen Werte nebeneinander stehen und mithilfe der Intervalldarstellung das Modell mit einem Speicherverbrauch von $2 \cdot l \cdot 64$ Bit speichern, wenn man annimmt, dass d vom Typ *double* ist und deshalb 64 Bit im RAM belegt. Der obere Teil von Tabelle 3 zeigt eine Sortierung, die nicht optimal ist, da am Anfang und am Ende jeweils der Wert w_i vorkommt. Bei der Regression kann man davon ausgehen, dass man die Parameter so sortieren kann, dass alle w_i hintereinander stehen. Somit hätte man ein neues $\boldsymbol{\theta}'$ mit $\{5, w_i, 3, w_k\}$ und müsste nur vier Zeichen, statt der ursprünglichen acht abspeichern. Es wäre also ähnlich dem allgemeinen Ansatz der Intervalldarstellung, mit dem Unterschied, dass hier die kürzestmöglichste Liste abgespeichert wird, da alle gleichen Werte nebeneinanderstehen und sich deshalb keine Werte doppeln. Die abgespeicherte Liste kann also nicht wie folgt aussehen: $\{5, w_i, 3, w_k, 4, w_i\}$, sondern es stünde zu Beginn $\{9, w_i\}$ in der Liste. Hat man $y = \langle \boldsymbol{\theta}, \boldsymbol{x} \rangle$ gegeben, so entspricht dies $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots \boldsymbol{\theta}_d)^T \cdot \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_d \end{pmatrix}$, wenn man nun innerhalb der $\boldsymbol{\theta}_i$ Werte verschiebt, so muss diese Verschiebung auch innerhalb der $\boldsymbol{x}_i$ durchgeführt werden. Das heißt es ergibt sich z. B.: $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_d, \boldsymbol{\theta}_{d-5} \dots \boldsymbol{\theta}_3)^T \cdot \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_d \\ \boldsymbol{x}_{d-5} \\ \vdots \\ \boldsymbol{x}_3 \end{pmatrix}$.

Diese Art der Sortierung funktioniert jedoch nicht bei allen Verfahren, da es häufig gewisse Abhängigkeiten zwischen den einzelnen Parametern gibt.

Probabilistische Modelle. Bei probabilistischen Modellen, wie z. B. Naive Bayes, stellen die Parameter bedingte Wahrscheinlichkeiten dar, die sich auf mehrere Zustände derselben Zufallsvariable beziehen. Da diese zusammenbleiben müssen und die Informationen nicht getrennt werden dürfen, kann man hier die Parameter nicht willkürlich

sortieren. Man kann sich die Darstellung in Form von untereinander geschriebenen Blöcken vorstellen (s. Abb. 13 links). Jeder dieser Blöcke entspricht einem Wert von $\mathcal{X}$ bedingt auf $\mathcal{Y}$, mit $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ und $\mathcal{Y} = \{y_1, \ldots, y_n\}$. Zusätzlich gibt es einen Block nur für $\mathcal{Y}$, welcher die *a-priori-Wahrscheinlichkeiten* der Klassen enthält. Eine a-priori-Wahrscheinlichkeit ist z. B. das vorherige Wissen über die Wahrscheinlichkeitsverteilung, dass Golf gespielt wird (s. Beispiel 2.2.3). Nun kann man annehmen, dass es beliebig ist, in welcher Reihenfolge die einzelnen Werte von $\mathcal{Y}$ stehen, ähnlich der Überlegung bei der linearen Regression, dann kann man die Werte umsortieren und in der Intervalldarstellung abspeichern. Bei den restlichen Blöcken lassen sich die Werte von $\mathcal{X}$ nur für jeweils einen Block für den Wert $\boldsymbol{x}^m$ sortieren und müssen für alle weiteren Blöcke übernommen werden, wie Abbildung 13 (rote und grüne Markierungen) verdeutlichen soll. Die Blöcke, die die Werte von X enthalten, müssen entsprechend der Sortierung innerhalb des $\mathcal{Y}$-Blocks angepasst werden. Wird in $\mathcal{Y}$ beispielsweise w_{y_4} mit w_{y_1} vertauscht, so müssen die Blöcke unter den Bedingungen y_4 und y_1 auch getauscht werden. Die Reihenfolge innerhalb des Blocks von $\mathcal{Y}$ gibt also die Reihenfolge der Blöcke, die $\mathcal{X}$ enthalten, an (s. blaue Markierung in Abb. 13). Zusätzlich können die Blöcke von $\mathcal{X}$ untereinander getauscht werden, da es nicht relevant ist, welcher Block $\boldsymbol{x}^1$ oder $\boldsymbol{x}^4$ ist.

Daraus folgt, dass man im schlimmsten Fall nur die Sortierung für einen Block optimal wählt und alle anderen Blöcke beliebig schlecht sind. Bei dieser Möglichkeit kann man mindestens $m+1$ Blöcke plus eventuell weitere zufällig auftretende Intervalle innerhalb der nicht sortierten Blöcke an Speicherbedarf einsparen.

Support Vector Machine. Ähnlich ist es auch bei der SVM (s. Abb. 14 (links)). Zunächst aber eine kurze Erläuterung, weshalb man auch bei der SVM diese Ansicht wählen kann. Auf den ersten Blick mag es so aussehen, dass wichtige Informationen verloren gehen, wenn man die Gewichte der Stützvektoren α_i und die jeweiligen Daten x_i^n untereinander schreibt, da es keinen Zusammenhang zwischen den einzelnen Datenpunkten zu geben scheint. Allerdings sind die einzelnen Datenpunkte auch nicht von Bedeutung, da die einzelnen numerischen Werte des Modells zusammengefasst werden sollen und nicht die ursprünglichen Datenpunkte. Deshalb entstehen die Blöcke wie folgt: der erste Block enthält alle Gewichte α_i, der zweite Block besteht aus den zugehörigen Daten von α_1, der nächste Block aus den Daten von α_2 und so weiter bis zu den Daten von α_n. Nun kann man sich wieder überlegen, eine beliebige, optimale Reihenfolge innerhalb des ersten Blocks festzulegen, da es keine Rolle spielt, an welcher Stelle der $i-$te Stützvektor steht. Zu beachten ist aber, dass entsprechend auch die Blöcke der Daten vertauscht werden müssen (s. Abb. 14). Wird also innerhalb des ersten Blocks (zur Vereinfachung mit 0 beziffert) α_1 mit α_3 vertauscht, so müssen auch die Blöcke 1 und 3 vertauscht werden. Diese Ordnung ist auch eine gewisse Art einer latenten Struktur, allerdings nicht in dem Sinne wie der Titel der Arbeit es vorsieht, da man die Struktur durch die Sortierung von außen vorgibt. Wichtig zu erwähnen ist noch, dass man theoretisch nicht nur den Block mit den α_i als Ausgangsblock nehmen darf, sondern jeden beliebigen Block, der die beste Sortierung zur Komprimierung bietet.

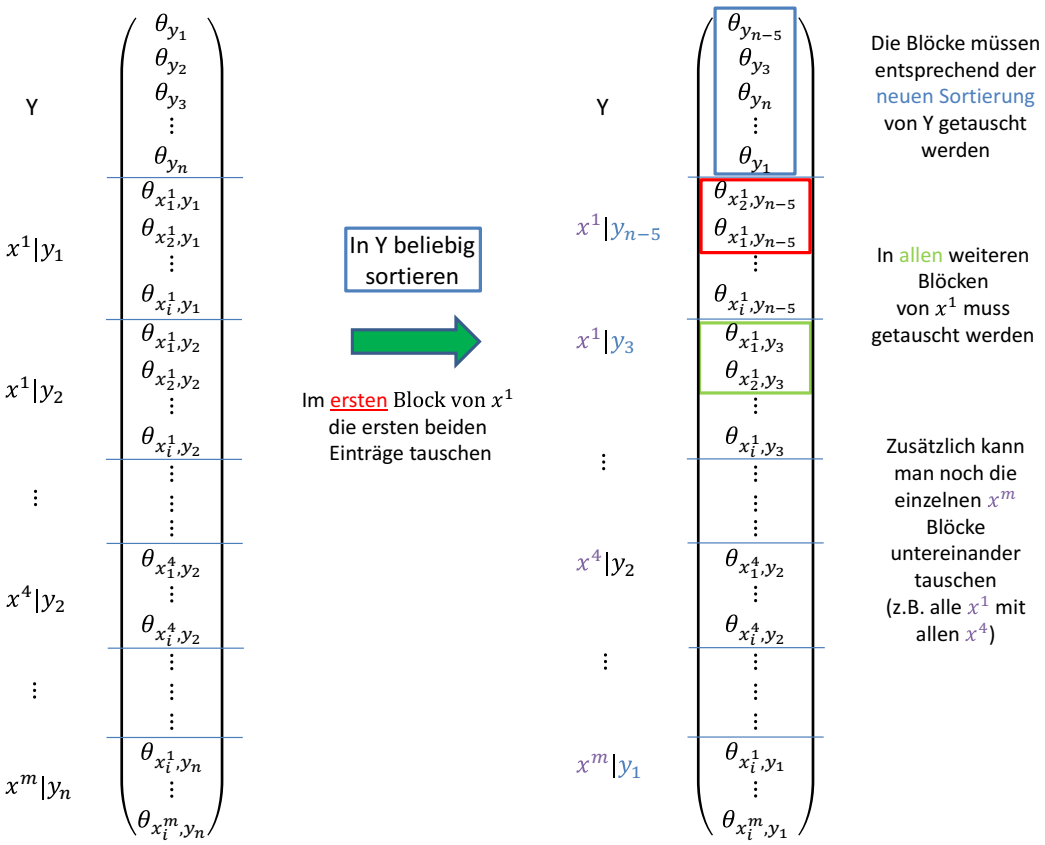

Abbildung 13: *Speicherbedarf für Naive Bayes. Zu sehen sind zwei Parametervektoren. Auf der linken Seite ist der ursprüngliche Vektor, auf der rechten Seite der gleiche Vektor, nun aber mit einer möglichst optimalen Sortierung, um so viele Werte wie möglich zusammenfassen zu können. Hier gibt es drei verschiedene Punkte, die sortiert werden können. Zum einen der Block mit den a-priori-Wahrscheinlichkeiten $\mathcal{Y}$ (blaue Markierung), dann innerhalb des ersten Blocks von $\boldsymbol{x}^m$ (rote und grüne Markierung) und zusätzlich noch die kompletten Blöcke (lila Schrift).*

Entscheidungsbäume. Für einen Entscheidungsbaum bedarf es einiger Erklärungen, was hier genau geclustert werden kann bzw. wie der Parametervektor $\boldsymbol{\theta}$ bei einem Entscheidungsbaum aussieht. Innerhalb eines Entscheidungsbaumes gibt es Verzweigungen, an denen, anhand der zugehörigen Gewichte, entschieden wird, welcher Pfad als nächstes gewählt wird. Diese Gewichte verstehen sich hier als die Parameter und $\boldsymbol{\theta}$ ist dann entsprechend der Vektor, der alle Gewichte $\boldsymbol{\theta}_i$ enthält, die sich innerhalb des Baumes befinden. Die Reihenfolge der Gewichte entspricht der Reihenfolge, in der die Gewichte innerhalb einer Tiefensuche gesehen werden.

Bei einem einzelnen Entscheidungsbaum gibt es keine großen Möglichkeiten etwas geschickt zu sortieren. Deshalb kann man hier nur auf den Zufall hoffen und den Speicherverbrauch in Abhängigkeit der unterschiedlichen Werte innerhalb des Baumes auf $\leq d$ beschränken. Eine geschickte Umsortierung ist nicht möglich, da man zusätzlich

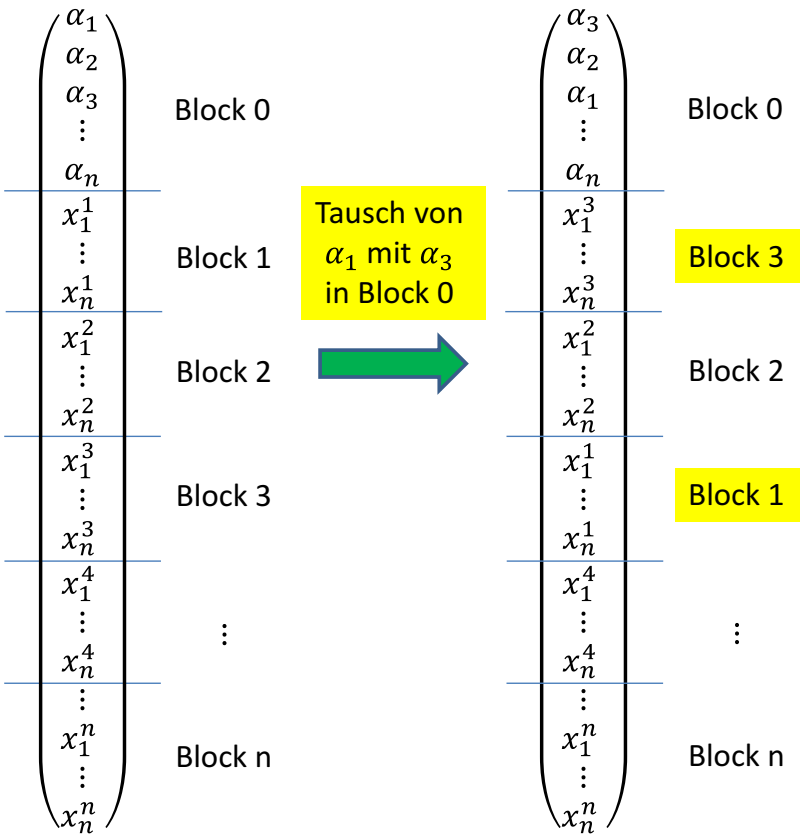

Abbildung 14: *Sortierung zur Berechnung des Speicherverbrauchs einer SVM. Die Abbildung soll darstellen, wie man sich eine mögliche Sortierung bei der SVM vorstellen kann. Der Parametervektor wird in Blöcke eingeteilt, wobei Block 0 die Gewichte der Stützvektoren α_i darstellt. Werden nun zwei Gewichte vertauscht, so müssen auch die jeweiligen Datenblöcke vertauscht werden.*

abspeichern müsste, wie die Sortierung vorher war. Das spart keinen Speicherverbrauch.

Hat man im Gegenzug dazu mehr als einen Entscheidungsbaum, z. B. bei Random Forest, kann man sich nun überlegen, dass es verschiedene Möglichkeiten gibt, um die Parameter zu sortieren. Man kann sich vorstellen, dass die Bäume in einer Matrix dargestellt sind (s. Abb. 15). Die Spalten entsprechen jeweils einem Baum und die Zeilen den einzelnen Verzweigungen. In der ersten Zeile stehen alle Verzweigungen, die auch in den Bäumen an erster Stelle sind, in der zweiten Zeile die der zweiten Stellen usw.

Jetzt kann man sich überlegen, dass man Speicher sparen kann, indem man die Zeilen optimal sortiert und ähnliche Werte zusammenfasst. Dann hätte man allerdings das gleiche Problem, wie bei einem einzelnen Baum, da man sich zusätzlich die ursprüngliche Reihenfolge abspeichern muss, damit eine Rekonstruktion möglich ist. Eine andere Möglichkeit wäre, dass man die Spalten, also die Position der einzelnen Bäume vertauscht. Hier ist die Überlegung die gleiche wie bei Regression oder SVM: es ist

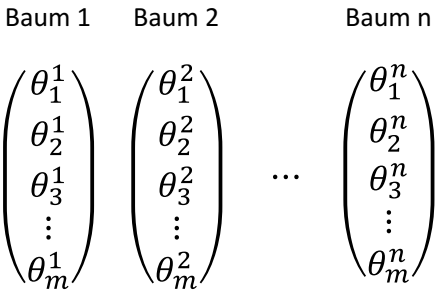

Abbildung 15: *Parametervektoren bei Random Forest. Die einzelnen Parametervektoren stellen die jeweiligen Bäume dar. An der Stelle $\boldsymbol{\theta}_1^i$ steht jeweils die erste Verzweigung der Bäume, an Stelle $\boldsymbol{\theta}_2^i$ die zweite und so weiter. Die einzelnen Bäume können vertauscht werden und müssen die aktuelle Position nicht beibehalten.*

nicht relevant, an welcher Stelle der i-te Baum steht, da dadurch keine Informationen verloren gehen. Hat man eine optimale Sortierung gefunden, könnte man mithilfe der Intervalldarstellung die einzelnen Zeilen abspeichern. Innerhalb der obersten Zeilen sollte jeder Baum einen Wert aufweisen. Allerdings kann es aufgrund der unterschiedlichen Baumgröße passieren, dass man auf einen Parametervektor $\boldsymbol{\theta}_j$ trifft, der kürzer ist als Vektor $\boldsymbol{\theta}_k$. In diesem Fall wird die betroffene Stelle innerhalb von $\boldsymbol{\theta}_j$ mit einer 0 aufgefüllt. Eine weitere Möglichkeit wäre, die Bäume innerhalb eines Vektors untereinander zu schreiben. Nun könnte man sich wieder jeden Baum als Block vorstellen und die einzelnen Blöcke vertauschen, sodass man eventuell an den jeweiligen Schnittstellen Zeichen einsparen kann.

Insgesamt lässt sich feststellen, dass es bei allen beschriebenen Verfahren Möglichkeiten gibt, eine mehr oder weniger effektive und geschickte Sortierung durchzuführen, sodass man den Speicherverbrauch einsparen kann. Allerdings muss man erwähnen, dass der Speicherverbrauch niemals größer als d sein wird. Sollte dies geschehen, kann man direkt das Ursprungsmodell abspeichern.

3.3 Latente Strukturen in überwachten Lernverfahren

Wie in der Einleitung dieses Kapitels beschrieben, ist es Ziel dieser Arbeit, latente Strukturen innerhalb der Parameterräume von maschinellen Lernverfahren zu finden. Erwähnt wurde außerdem auch, dass verschiedene Verfahren verwendet werden, um diesem Ziel näher zu kommen. Für die einzelnen Verfahren gibt es unterschiedliche Anwendungsmöglichkeiten und Interpretationen woraus die latente Struktur besteht.

Entscheidungsbaum. Innerhalb eines Entscheidungsbaums kann es numerische und nominale Parameter geben. In dieser Arbeit werden jedoch nur die numerischen Parameter betrachtet, da eine Anwendung auf die nominalen Parameter keine Änderung ergeben würde und sie somit gar nicht beachtet werden müssen. Das Problem bei den

nominalen Parametern liegt darin, dass man z. B. zwischen „rot“ und „blau“ keine Distanz berechnen kann und aufgrund dessen auch keine Entscheidung treffen kann, wie weit die Punkte voneinander entfernt sind.

Betrachtet man einen einzelnen Entscheidungsbaum, kann es passieren, dass es eventuell nicht genügend numerische Parameter gibt, die man zusammenfassen kann. Um diesem Problem entgegenzuwirken, kann man sogenannten Ensemblemethoden nutzen. Ein häufig genutztes Verfahren ist Random Forest (s. Abschnitt 2.2.1).

Random Forest besteht aus mehreren Entscheidungsbäumen und bietet sich hier an, da man eine große Menge von Bäumen repräsentieren will. Da man nun eine Menge von Bäumen repräsentiert wird, gibt es genügend numerische Parameter, die geclustert werden können. Hier werden die Splitpunkte, also die einzelnen Verzweigungen, aller Bäume innerhalb des Clusterings betrachtet, um die latenten Strukturen zu finden. Damit erzeugt man deutlich größere Modelle, auf denen man das Clustering anwenden kann. Denn aus einer großen Anzahl von Bäumen geht eine große Anzahl von Parametern hervor. Die Bäume werden aus einer zufälligen Teilmenge der Trainingsmenge gelernt und es kann vorkommen, dass es sehr viele ähnliche Splitpunkte gibt, die man gut zusammenfassen kann und somit eine niedrigere Komplexität erzeugt. Allerdings besteht hier die Gefahr, dass gerade die kleinen Unterschiede in den Splitpunkten wichtig sind und dass man einen höheren Fehler produziert, wenn man alle zusammenfasst und die Daten zu sehr manipuliert. Ob sich dieser Punkt bewahrheitet, wird innerhalb der Experimente beantwortet (s. Abschnitt 5.3.3).

Support Vector Machine. Bei der Support Vector Machine gibt es den Unterschied zwischen primaler und dualer Darstellung. In dieser Arbeit wird die Dualdarstellung verwendet, da die beliebtesten Kernel, z. B. RBF, gar keine explizite Primaldarstellung hat. Die duale Darstellung wird repräsentiert durch eine Menge von Gewichten der Stützvektoren α_i und zugehörigen Daten x_i. In den Experimenten, die in Kapitel 5 ausgeführt sind, wird die Frage beantwortet, warum es sinnvoll ist, das Clustering auf den Gewichten der Stützvektoren und zusätzlich auf den Daten anzuwenden. Anders als in [27] werden hier nicht zuerst die Daten geclustert und dann die SVM angewendet, um die Anzahl der Gewichte der Stützvektoren zu verringern, sondern es wird erst die SVM verwendet und dann wird der neue Datensatz geclustert. Zudem ist es wichtig zu erwähnen, dass bei der Methode dieser Arbeit nicht der ganze Vektor geclustert wird, sondern die einzelnen Werte in den Parametervektoren. Zudem besteht die Möglichkeit die Daten sowie die Gewichte der Stützvektoren gleichzeitig zu clustern. Aber auch hier ist es notwendig zu sagen, dass es ein ähnliches Risiko wie bei den Bäumen gibt, dass kleine Unterschiede in den Daten der Stützvektoren wichtige Informationen für die SVM enthalten und man Gefahr läuft das Modell damit zu sehr zu manipulieren. Dieses wird in den Experimenten evaluiert.

Probabilistische Modelle Bei den probabilistischen Modellen ist es plausibel, dass die Chance viele latente Strukturen zu finden sehr hoch ist, da es sich bei den Parametern in der Regel um Wahrscheinlichkeiten oder bedingte Wahrscheinlichkeiten handelt.

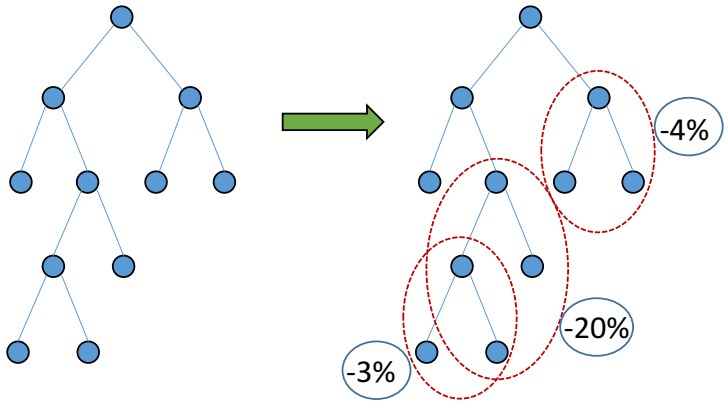

Abbildung 16: *Latente Struktur innerhalb des Prunings. Die Abbildung zeigt auf der linken Seite einen Baum ohne angewandtes Pruning. Auf der rechten Seite stellen die roten Kreise mit den Prozentangaben einzelne Teilbäume mit der jeweiligen Verschlechterung der Güte dar.*

Diese werden, wie in Abschnitt 3.2 beschrieben, empirisch aus den Häufigkeiten der Daten geschätzt. Dieser Schätzer ergibt sich aus $\frac{n}{N}$, wobei $n \in \{0, \ldots, N\}$ die Häufigkeit des Vorkommens ist und N die Gesamtanzahl der möglichen Wahrscheinlichkeiten. Nun kann man sich überlegen, warum die Behauptung, dass es plausibel ist, viele ähnliche Parameter bzw. Wahrscheinlichkeiten zu finden, stimmt. Angenommen, N ist sehr klein, dann sind entsprechend viele der Werte ähnlich, da es nicht viele verschiedene Werte gibt. Ist N hingegen sehr groß, sind die Wahrscheinlichkeiten $\frac{n}{N}$ und $\frac{n-1}{N}$ sich sehr ähnlich und können auch zusammengefasst werden.

3.4 Andere Ansätze zur Komprimierung

Das Verringern der Komplexität eines Modells ist keine neue Idee. So gibt es für Entscheidungsbäume (s. Abschnitt 2.2.1) die Möglichkeit Pruning oder Pre-Pruning zu nutzen (s. Abschnitt 2.2.1), um das Modell zu verkleinern und Overfitting (s. Abschnitt 2.2.1) zu vermeiden. Allerdings wird innerhalb des Prunings nicht explizit nach latenten Strukturen gesucht. Stattdessen werden solange Teilbäume entfernt, solange die Güte des Entscheidungsbaums noch ähnlich der ursprünglichen Güte ist. Dies kann jedoch als implizite Suche nach latenten Strukturen interpretiert werden. Dafür betrachtet man den Baum auf den kein Pruning angewendet wurde und berechnet für jeden Teilbaum die Güte, die verloren ginge, wenn man diesen Teilbaum entfernen würde. Nun werden die einzelnen Teilbäume anhand dieser neuen Werte geclustert und die Teilbäume aus dem Cluster, das die geringsten Werte aggregiert, entfernt (s. Abbildung 16). Dieses Vorgehen ähnelt im Ergebnis dem des Pruning, ist allerdings nicht genau das gleiche und wird auch hier nicht weiter verfolgt. Denn hierbei handelt es sich nicht um die Untersuchung latenter Strukturen in einem Parameterraum, da hier keine Parameter betrachtet werden.

Bei der Support Vektor Machine (s. Abschnitt 2.2.2) gibt es ebenfalls schon einen Ansatz, um möglichst kleine Modelle zu finden. Dieses ist in dem Optimierungsproblem schon berücksichtigt. Der Punkt, der bei der SVM dafür sorgt, dass es eine eindeutige Lösung gibt, kann ebenfalls als Verringerung der Komplexität aufgefasst werden. Parametervektoren $\boldsymbol{\beta}$ werden mit einem Strafterm belegt, der proportional zu ihrer Länge ist. Je größer die Länge des Vektors, desto größer ist auch die Strafe. Die Länge eines Vektors ist definiert durch den Betrag der Einträge in dem Vektor $\boldsymbol{\beta}$. Da die Strafe sich mit größerer Länge ebenfalls erhöht, führt das dazu, dass man nur möglichst kleine Zahlen als Ergebnis bekommt. Dadurch werden Parametervektoren mit großer Länge direkt ausgeschlossen, weshalb man hier von einer Komplexitätsverringerung sprechen kann. Allgemein bezeichnet man das Minimieren der Norm eines Parametervektors als Regularisierung.

Definition 7. Regularisierung. Gegeben sei ein überwachtes Lernproblem mit einer Verlustfunktion L. Nun soll über alle Trainingsbeispiele der Parametervektor $\boldsymbol{\beta}$ gefunden werden, der den Verlust für alle Beispiele minimiert. Das regularisierte Lernproblem besteht nun darin, dass zusätzlich zu L eine gewichtetete P-Norm des Modellvektors minimiert wird: $\min_\beta \sum_{i=1}^{n} L\left(\boldsymbol{\beta}, \boldsymbol{x}_i, y_i\right) + \lambda \parallel \boldsymbol{\beta} \parallel_P$, mit $\lambda > 0$. Wie stark die Norm minimiert wird, kann über das λ eingestellt werden.

Die P-Norm $\parallel \boldsymbol{\beta} \parallel_P := \left(\sum_{i=1}^{n} \mid \boldsymbol{\beta}_i \mid^P\right)^{\frac{1}{P}}$ misst die Länge eines Vektors $\boldsymbol{\beta}$. Über das P lässt sich der Grad der Norm und somit die Art der Bestrafung einstellen[5]. Entsprechend gibt es verschiedene Arten der P-Norm. Die meist genutzten sind die 1-Norm, die 2-Norm und die ∞-Norm.

Bei der 1-Norm bekommt man möglichst viele Nullen, was man auch als das Auffinden einer latenten Struktur betrachten kann, bei der das z_i angibt, ob der Parameter gleich null ist oder nicht. Aus der Optimierung der Norm ([32], Abschnitt 6) folgt, dass Einträge von $\boldsymbol{\beta}$, die kleiner sind als λ, auf 0 gesetzt werden. Vektoren mit möglichst vielen Nullen haben eine kleine Norm. Aufgrund der Ableitung der 2-Norm (wie bei der SVM verwendet), werden die Werte immer kleiner, aber niemals explizit auf 0 gesetzt, womit man möglichst kleine Zahlen bekommt. Die ∞-Norm bestimmt das Maximum der Länge des Vektors. Das λ bestimmt, wie wichtig der Strafterm sein soll. Bei einem großen λ spielt die Verlustfunktion nur eine kleine Rolle.

Innerhalb dieser Arbeit wird jedoch das Vorgehen, welches in den vorherigen Abschnitten (3.1 bis 3.3) erläutert wurde, genutzt. Allerdings kann man die 1-Norm als eine Art Spezialfall des Vorgehens dieser Arbeit ansehen. Denn hierbei werden viele Nullen erzeugt, die man nicht abspeichern muss und das ist ähnlich zu der Methode viele gleiche oder ähnliche Werte zu haben, bei denen auch Speicherplatz gespart werden kann. Setzt man die Werte, die mit der 1-Norm berechnet werden, in die Modelle der Lernverfahren ein, so kann es vorkommen, dass bestimmte Teile der Formeln nicht berechnet werden müssen, wenn vorher bekannt ist, was das Ergebnis sein wird. Ähnlich ist es mit den gleichen Werten des vorgestellten Verfahrens, da auch hier die Formeln vereinfacht werden können, wenn die Werte vorher bekannt sind. Der Unterschied der beiden An-

[5] In der Literatur wird ein kleines p für die Norm verwendet. In dieser Arbeit bezeichnet das kleine p allerdings eine Wahrscheinlichkeit, weshalb die Norm hier die Bezeichnung P hat.

sätze besteht darin, dass die Regularisierung auf die Schätzung der Modellparameter, also während des Trainings, angewendet wird, wohingegen das Clustern der Werte erst danach geschieht. Der Vorteil hierbei ist, dass das Verfahren auf jedes bestehende Lernverfahren angewendet werden kann und man nicht in die bestehende Implementierung eingreifen muss.

Das nächste Kapitel erklärt welche Voraussetzungen geschaffen werden mussten, um die Experimente (s. Kapitel 5) durchführen zu können.

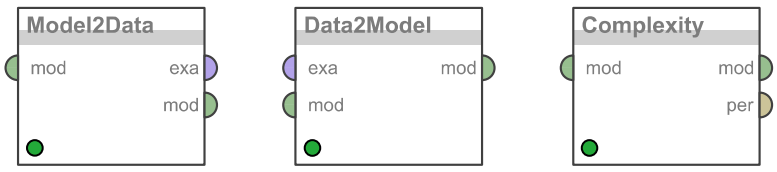

Abbildung 17: *Rapidminer Operatoren. Die drei implementierten Operatoren in Rapidminer, die zur Durchführung der Experimente benötigt werden: Model2Data, Data2Model und Complexity.*

4 Implementierung

Die in Kapitel 5 beschriebenen Experimente werden mit dem Programm Rapidminer[6] durchgeführt. Doch damit eine Durchführung überhaupt realisierbar ist, werden zusätzliche Operatoren benötigt, die das Vorgehen möglich machen, wie es in Abschnitt 3.2(speziell in Abbildung 11) beschrieben wurde.

Rapidminer ist eine Software zur Wissensentdeckung und Visualisierung in und von Daten im Bereich des maschinellen Lernens. Es gibt eine große Anzahl sogenannter Operatoren, die verschiedene Aufgaben erfüllen. Beispielsweise gibt es Operatoren, um Daten einzulesen und auszugeben. Für das Feld des Data-Mining, für maschinelle Lernverfahren und ähnliche Aufgaben. Alle in Kapitel 2 beschriebenen Grundlagen sind in Rapidminer vorhanden und können genutzt werden. Um das im vorherigen Kapitel erklärte Vorgehen durchzuführen, ist es vonnöten auf Operatoren zuzugreifen, die es so noch nicht innerhalb der Werkzeuge von Rapidminer zu finden gab. Das Ziel ist es, mithilfe von unüberwachten Lernverfahren, die auf den Parametervektoren von überwachten Lernverfahren lernen, ein möglichst kleines, aber trotzdem gleich gutes Modell wie das Ausgangsmodell zu erhalten. Deshalb wurden im Laufe der Arbeit drei Operatoren erstellt (s. Abb. 17), die folgende Aufgaben erfüllen sollen:

1. Das vom überwachten Lernverfahren gelernte Modell ist vom Typ `model` und kann aufgrund dessen nicht von einem unüberwachten Lernverfahren bearbeitet werden. Dieses benötigt einen Datensatz (vom Typ `ExampleSet`) auf dem es lernen kann.

2. Die Ausgabe des unüberwachten Lernverfahrens muss wieder in ein Modell umgewandelt werden.

3. Die Modellkomplexität soll, wie in Abschnitt 3.2 beschrieben, berechnet werden.

In Rapidminer gibt es zwar eine Klasse, die Modelle repräsentiert, namens `model`, allerdings ist es nicht festgelegt, wie ein Modell intern von den einzelnen Lernern abgespeichert wird. Die Lernverfahren nutzen unterschiedliche Datenstrukturen und Darstellungen für das `model`, weshalb es keine allgemeine Implementierung der Operatoren geben

[6] https://rapidminer.com/ (Stand: 30.07.2015), Version 5.3.015

kann. Aufgrund dessen benötigen die Operatoren explizites Wissen über die interne Repräsentation der jeweiligen Modelle und müssen zudem unterscheiden können um welches Modell es sich handelt. Dieses Wissen ist ebenfalls notwendig, um die einzelnen benötigten Parameter aus der Darstellung auslesen zu können. Folgende verschiedene Modelltypen werden hier berücksichtigt:

- `LinearRegressionModel`
- `SimpleDistributionModel`
- `TreeModel`
- `RandomForestModel`
- `KernelModel`

Für die aufgelisteten Modelltypen kann der erste Operator Modelle in ein `ExampleSet` umwandeln. Für das Zurückschreiben in ein Objekt der Klasse `model` gibt es jedoch eine Einschränkung (s. Abschnitt 4.2). Es wurde versucht, ein möglichst breites Spektrum an Methoden abzudecken. Es gibt aber noch eine große Anzahl weiterer Modelltypen, für die müsste die bestehende Implementierung jedoch erweitert werden. Hierfür benötigt man den Sourcecode der vorhandenen Lernverfahren und muss herausfinden, mit welcher Datenstruktur das Modell abgespeichert wird und wie man daraus ein einheitliches `ExampleSet` erstellen kann, damit ein Clusterverfahren darauf angewendet werden kann.

4.1 Model2Data

Der erste neue Operator heißt *Model2Data* und erfüllt die Aufgabe die Modelle, welche mit den überwachten Lernverfahren gelernt werden, in einen Datensatz (`ExampleSet`) umzuwandeln, der für das weitere Vorgehen benötigt wird. Für das `ExampleSet` werden die Werte der einzelnen Parameter aus dem Modell benötigt. Damit der Operator *Data2Model* (s. Abschnitt 4.2) den Datensatz wieder in ein Modell zurückführen kann, ist es wichtig zu speichern, wo welcher Parameter stand. Dazu werden zwei neue Hilfsattribute erstellt, zum einen `parameter` und zum anderen `parameterId`. Aus jedem Modell werden die Parameter einheitlich ausgelesen und abgespeichert. Je nach Modell kann es aber auch noch zusätzliche Attribute geben (s. Abb. 18). Ist das angeschlossene Modell dem Operator nicht bekannt, so wird ein leeres `ExampleSet` herausgegeben.

Lineare Regression. Der einfachste Fall, um die Parameter auszulesen liegt bei der linearen Regression. In diesem Fall befinden sich die Parameter (hier Koeffizienten) in einem Array und können daraus der Reihe nach ausgelesen und in die neue Spalte `parameter` eingefügt werden. Die `parameterId` ist hier die Stelle des gelesenen Parameters im Array.

Naive Bayes

Row No.	parameterId	parameter
1	0	-1.099
2	1	-1.099
3	2	-1.099
4	3	5.006
5	4	0.352

Support Vector Machine

Row No.	parameterId	svId	svData	parameter
1	0	0	0	0.244
2	1	0	0	4.900
3	2	0	0	3
4	3	0	0	1.400
5	4	0	0	0.200
6	5	1	0	1

Lineare Regression

Row No.	parameterId	parameter
1	0	0.655
2	1	0.711
3	2	-0.563
4	3	1.845

Random Forest

Row No.	parameterId	treeId	parameter
1	0	1	2.450
2	1	1	1.650
3	2	1	1.650
4	3	1	4.950
5	4	1	6.150
6	5	1	6.150

Abbildung 18: *Ausgabe von Model2Data für das jeweilige Modell. Die inneren Spalten sind* special attributes *die nicht mitgeclustert werden. SVM und Random Forest haben zusätzliche Spalten zur späteren Identifikation.*

Naive Bayes. Bei Naive Bayes sind die ersten k Zahlen die a-priori-Wahrscheinlichkeiten der vorhandenen k Klassen innerhalb der Daten. Abbildung 18 wurde anhand des Iris-Datensatzes erstellt. In diesem gibt es drei verschiedene Klassen, die alle die gleiche Wahrscheinlichkeit $(0,33)$ haben. In der Abbildung sieht man den Wert $-1,099$, was daran liegt, dass die Werte innerhalb von Rapidminer logarithmiert abgespeichert und ausgegeben werden. Für die Interpretation der darauf folgenden Daten muss vorher noch eine Unterscheidung getroffen werden, ob der gegebene Datensatz numerische oder diskrete Werte liefert. Sind die Werte diskret, werden die jeweiligen Wahrscheinlichkeiten ausgegeben. Im Falle numerischer Werte werden Mittelwert und Standardabweichung ausgegeben (s. Abschnitt 2.2.3). Innerhalb der Implementierung in Rapidminer hat der naive Bayes Klassifikator eine spezifische Eigenart bei der Behandlung von numerischen Werten, die nicht aus der formalen Definition von Naive Bayes hervorgeht: durch die Implementierung wird ein dritter Wert gespeichert, welcher die Dichtefunktion (s. Formel (1)) der Normalverteilung darstellt. Werden nun die Wahrscheinlichkeiten benötigt, kann der Wert der Dichtefunktion direkt aus dem Speicher gelesen werden und muss nicht bei jeder Wahrscheinlichkeit erneut ausgerechnet werden, was sehr teuer wäre. Da der Wert für das weitere Vorgehen nicht von Belang ist bzw. sogar das Ergebnis des Clusterings verfälschen könnte, muss der Operator Model2Data den Wert entsprechend behandeln und nicht in das `ExampleSet` einfügen.

Entscheidungsbaum. Wurde das anliegende Modell von einem einzelnen Entscheidungsbaum (s. Abschnitt 2.2.1) gelernt, so ist es vom Typ `TreeModel`. Liegt es einem Ensemble von Entscheidungsbäumen zugrunde, in diesem Fall dem Random Forest Ver-

fahren, so hat es den Typ `RandomForestModel`. In beiden Fällen wird eine zusätzliche Spalte `TreeId` eingefügt, die sicherstellt, dass man für die spätere Zurückwandlung des `ExampleSets` in ein `model` weiß, welcher Parameter zu welchem Baum gehört. In der Spalte `parameter` befinden sich die Werte der einzelnen Verzweigungen innerhalb des Baums geordnet anhand einer Tiefensuche. In `parameterId` steht wieder die ID des jeweiligen Wertes, wie auch bei der Regression und Naive Bayes. Die Spalte ist auch bei der SVM gleich zu interpretieren.

Support Vector Machine. Das Modell, welches von einer Support Vector Machine (s. Abschnitt 2.2.2) gelernt wurde, ist vom Typ ein `KernelModel`. Dieses liegt nicht nur der SVM zugrunde, sondern auch anderen Lernverfahren, wie z. B. Gaußsche Prozesse. Das heißt, dass der vorgestellte Operator in der Lage ist auch Modelle von anderen Lernverfahren die ein Modell vom Typ `KernelModel` ausgeben, in ein `ExampleSet` umzuwandeln. Innerhalb der Versuche werden die sogenannten *MySVM* und *LibSVM* genutzt, die beide vom Typ `Kernelmodel` erben, spezieller aber vom Typ `AbstractMySVMModel` bzw. `LibSVMModel` sind. Diese Unterscheidung spielt hier noch keine Rolle, wird aber für den nächsten Operator wichtig. Für die SVM gib es zusätzlich eine oder auch zwei Spalten. Dies kommt auf die Auswahl innerhalb der Optionen (s. unten) an. Die Spalten heißen `svId` und für den Fall, dass die Daten mitgeclustert werden sollen, `svData`. In der Spalte `parameter` befinden sich die Dualparameter, im Fall der SVM die Werte von α_i und für den besonderen Fall, dass die Daten inkludiert sind, auch die Werte von x_i. Die Spalte `svData` wird nur eingefügt, wenn die Stützvektordaten mitbeachtet werden. Die Werte in `svData` sind immer 0. Hier ist es aber wichtig zu erwähnen, dass keine realen Werte eingefügt, sondern nur Nullpointer gesetzt werden. Somit wird mit dieser Spalte auch kein zusätzlicher Speicher belegt. Mit der Existenz dieser Spalte kann der Operator *Data2Model* unterscheiden, ob sich in der Spalte `parameter` nur die Werte von α_i oder auch die von x_i befinden. Im Gegensatz zu der Darstellung in Abbildung 14 ist die Reihenfolge der Werte etwas anders. Hier werden zunächst immer erst α_i und dann die zugehörigen x_i geschrieben, sodass hierbei nicht erst alle Gewichte und dann die Daten kommen.

Alle Operatoren in Rapidminer haben Eingang- und Ausgangsports, mit denen sie untereinander verbunden werden können. *Model2Data* besitzt einen Inputport, der das Modell einliest und zwei Outputports. Einen durch den das Modell weitergeleitet wird und einen, aus dem die umgewandelten Daten ausgegeben werden. Die Daten liegen dann in einem `ExampleSet` vor und können als Eingabe eines jeden Operators, der einen solchen Typ benötigt, genutzt werden.

Des Weiteren kann man den einzelnen Operatoren verschiedene Optionen zuweisen, die die Einstellmöglichkeiten für den Benutzer darstellen. In diesem Fall hat der Nutzer die Möglichkeit zu definieren, ob bei Nutzung einer SVM die Gewichte der Stützvektoren (α_i) und die zugehörigen Daten (x_i) von dem unüberwachten Lernverfahren (hier: k-means (2.2.5), DBSCAN (2.2.6)) geclustert werden sollen, oder nur die Stützvektoren allein. Diese Option findet man unter *include support vector data.* In den Experimenten hat sich herausgestellt, dass sich das Clustern der Stützvektoren und der zugehörigen Daten mehr auszahlt, als wenn die Daten nicht mit einbezogen werden. Aufgrund dessen

ist diese Option standardmäßig ausgewählt.

Nachdem nun ein `ExampleSet` erzeugt wurde, kann darauf jeder beliebige Operator, der diesen Datentyp als Eingabe benötigt, angewendet werden. Im Falle dieser Arbeit handelt es sich um ein Clusterverfahren in Form von k-means oder DBSCAN, welches die neuen Daten in Cluster einteilt.

4.2 Data2Model

Der zweite neue Operator heißt *Data2Model* und dient dazu, das `ExampleSet`, das von dem Clusterverfahren neu erzeugt wurde, in ein Objekt vom Typ `model` zurückzuwandeln. Dies ist notwendig, damit das neue Modell mit dem Ursprungsmodell verglichen werden kann. Für das neue Modell werden die Werte aus dem `ExampleSet` ausgelesen und können mithilfe der eingefügten Spalten von *Model2Data* richtig zugeordnet werden. Es ist z. B. bekannt welcher Wert zu welchem Baum oder zu welchem Stützvektor gehört.

Auch dieser Operator kann zwischen den verschiedenen Modelltypen unterscheiden. Hier können folgende Modelle verarbeitet werden:

- `LinearRegressionModel`
- `SimpleDistributionModel`
- `TreeModel`, `RandomForestModel`
- `AbstractMySVMModel`
- `LibSVMModel`

Es fällt auf, dass in dieser Aufzählung das `KernelModel` nicht mehr auftaucht und stattdessen zwei unterschiedliche Typen von SVM Modellen genutzt werden. Die Erklärung hierfür folgt weiter unten.

Data2Model hat zwei Inputports: einen an den das `ExampleSet` aus *Model2Data* angelegt wird und einen weiteren, der ein Modell als Eingabe erwartet. Das ist notwendig, um zu wissen um welchen Modelltyp es sich bei der Ausgabe handeln soll. Diese Ausgabe wird in Form eines Modells am Outputport ausgegeben. Das Modell welches am Inputport anliegt, stammt aus dem Operator *Model2Data*.

Der Operator besitzt ebenfalls Optionen zum Modifizieren der Ausgabe. Hier kann ausgewählt werden, ob bei DBSCAN der Noise-Cluster zusätzlich ersetzt und welche Ersetzungsstrategie für das Austauschen der Werte genutzt werden soll. Folgende Strategien stehen zur Auswahl, um die Werte mit den neuen geclusterten Werten zu ersetzen:

- der Mittelwert
- das Minimum
- das Maximum

Das bedeutet, dass sich Werte in dem neuen Modell aus den berechneten Werten aus dem Clusterverfahren ergeben. Nutzt man z. B. k-means und eins der k Cluster hat als Minimum den Wert $0,256$, so werden alle Werte der Parameter, die sich in dem Cluster befinden durch $0,256$ ersetzt.

Wie schon erwähnt wird auch hier zwischen den einzelnen Modelltypen unterschieden und die Daten auf unterschiedliche Weise in das neue Modell geschrieben.

Lineare Regression. Handelt es sich bei dem erwarteten Modell um ein `LinearRegressionModel`, so ist das der einfachste Fall. Wie auch *Model2Data* liest der Operator die Werte ohne weitere Berechnungen aus dem `ExampleSet` und schreibt diese zurück in das benötigte Array an die Stelle, die in der `parameterId` gespeichert wurde. Die ursprünglichen Koeffizienten, die durch die lineare Regression gelernt wurden, werden also direkt durch die neuen, aus dem Clusterverfahren stammenden Werte, ersetzt.

Naive Bayes. Wurde das überwachte Lernverfahren Naive Bayes angewandt, so handelt es sich bei dem erwarteten Modelltyp um ein `SimpleDistributionModel`. Hier muss die Unterscheidung zwischen diskreten und numerischen Werten getroffen werden. Handelt es sich um diskrete Werte, so müssen die Daten aus dem Clusterverfahren vor dem Einfügen in das Modell erst noch normalisiert werden, da man sonst nicht die richtigen Wahrscheinlichkeiten geliefert bekommt. Dann könnte es passieren, dass die Wahrscheinlichkeiten ungleich 1 sein könnten. Zudem muss hier die im vorherigen Abschnitt erwähnte Dichtefunktion zurückgeschrieben werden, damit das Modell dem Ursprungsmodell von Naive Bayes entspricht und das Ergebnis kein falsches Modell liefert. Dieses muss wiederum nur bei den numerischen Werten geschehen, da es nur hier notwendig ist.

Entscheidungsbaum. Die bestehende Implementierung von `TreeModel` und `RandomForestModel` in Rapidminer machte es notwendig sich dem Konzept der Reflexion zu bedienen. Hierbei wird ausgenutzt, dass Java zur Laufzeit noch alle Strukturen des Programms kennt und auf diese zugreifen kann. Mithilfe der Reflektion kann man von einem Objekt zur Laufzeit ermitteln, wie Konstruktor, Methoden und Variablen aussehen, auch wenn Variablen als *private* deklariert sind. Genau dies war das Problem, welches während der Implementierung auftrat: hier war eine Zugriffsbeschränkung vorhanden, sodass man nicht direkt auf die benötigten Daten zugreifen konnte. Damit *Data2Model* die Aufgabe, ein Modell zu erstellen, erfüllen kann, werden für das Modell das einen Baum repräsentiert, die Bedingungen benötigt, anhand derer der Baum erstellt wird. Auch hier werden diese Bedingungen wieder durch die neuen Werte, die durch das unüberwachte Lernverfahren berechnet wurden, ersetzt. Um die dazu benötigte Variable auslesen zu können, wurde der Codeausschnitt aus Algorithmus 3 genutzt.

Des Weiteren muss zur Erstellung des Baummodells die Unterscheidung getroffen werden, ob es sich um eine `GreaterSplitCondition` oder eine

Algorithmus 3 : java reflection

```
1 Field cond = null;
  try
2 |   cond = Edge.class.getDeclaredField("condition");
  |   cond.setAccessible(true);
3 catch (Exception ex)
```

`LessEqualsSplitCondition` handelt. Um diese Werte umschreiben zu können und daraus den neuen Baum zu erstellen, benötigte man wieder die Methode der Reflexion.

Support Vector Machine. Der Operator kann zwei verschiedene Modelltypen für eine SVM behandeln und nicht mehr, wie *Model2Data*, alle Modelle vom Typ `KernelModel`. Das liegt an den unterschiedlichen Datenstrukturen der einzelnen Modelle, die es für die SVM gibt. Es gibt keinen allgemeinen Fall, um ein Modell, welches genau der SVM entspricht, zu erzeugen. Deshalb wurden hier die beiden speziellen Typen `AbstractMySVMModel` und `LibSVMModel` implementiert. Diese erben beide von `KernelModel`, weshalb es für den ersten Operator keine Beschränkung dieser Art gibt, da die Daten nur ausgelesen werden müssen. In beiden Fällen wird auf die Existenz der Spalte `svData` geprüft, um festzustellen, ob es sich bei den Werten nur um die Gewichte α_i oder zusätzlich um die zugehörigen Daten x_i handelt. In dem Fall, dass es um ein `LibSVMModel` geht, musste eine benötigte Variable wieder mithilfe der Reflexion ausgelesen werden, da diese in der bestehenden Implementierung als `private` deklariert ist.

Wurden nun die Daten in ein Modell umgewandelt, so kann man dieses, mithilfe des dritten Operators, bezüglich der Komplexität und der Güte oder Genauigkeit bewerten.

4.3 Complexity

Ein weiterer benötigter Operator, der im Laufe der Arbeit implementiert wurde, heißt *Complexity* und berechnet, wie der Name schon andeutet, die Komplexität (s. Abschnitt 3.2) des neuen Modells. Der Operator verfügt über einen Inputport, über den er das Modell bezieht und zwei Outputports. Einen, um das Modell auszugeben und einen zweiten, um die berechnete Komplexität zu liefern. Mit dieser Komplexität kann ein anderer Operator in Rapidminer messen, wie gut das Ergebnis wirklich ist. Die dazu genutzten Operatoren heißen *Performance.* Um in den Experimenten die erwähnte Kreuzvalidierung (s. Abschnitt 2.2) anwenden zu können, benötigte diese einen Performanzvektor, der durch *Performance* ausgegeben wird. *Complexity* hat zum Teil eine ähnliche Funktionalität wie *Model2Data*, da auch hier das Wissen über die einzelnen Werte notwendig ist.

Der Operator hat verschiedene Funktionalitäten, die vom Nutzer innerhalb der Optionen ausgewählt werden können. Diese sind:

- *include support vector data*
- *complexity count*
- *complexity entropy*
- *complexity memory*

Die erste Funktion ist die gleiche wie auch bei *Model2Data*. Hier kann man bestimmen, ob, falls eine SVM genutzt wurde, die Daten der Stützvektoren vom Clustering miteinbezogen werden sollen oder nicht. Die restlichen drei Funktionen bestimmen die jeweilige Grundlage, wie die Komplexität berechnet und ausgegeben werden soll. Es ist aber auch möglich alle Optionen zu kombinieren.

Auch dieser Operator kann die vier verschiedenen Modelle (`TreeModel`, `RandomForestModel`, `KernelModel` und `SimpleDistributionModel`) verarbeiten und für alle Modelle, die vom erwarteten Typ sind, die Komplexität berechnen.

Wie in Abschnitt 3.2 erklärt, gibt es verschiedene Methoden, um die Komplexität zu messen. Die drei beschriebenen Möglichkeiten (einfaches Abzählen, Berechnen der Entropie, Berechnung des Speicherbedarfs) wurden hierfür implementiert.

complexity count (einfaches Abzählen) Die erste Möglichkeit ist *complexity count*, bei der die Anzahl l der unterschiedlichen Parameter bzw. Symbole gezählt wird.

complexity entropy (Berechnung der Entropie) Als zweite Möglichkeit kann man sich die Entropie (s. Abschnitt 2.1.1) des Modells ausgeben lassen. Da die genaue Wahrscheinlichkeitsverteilung $p(w_i)$ nicht bekannt ist, wird für die Berechnung hier ein empirischer Schätzer genutzt (s. Abschnitt 3.2). Zusätzlich zur Entropie des Modells werden noch die Wahrscheinlichkeiten der einzelnen Gewichte ausgegeben, auf denen die Entropie berechnet wurde.

complexity memory (Berechnung des Speicherbedarfs) Die aufwendigste Möglichkeit, um die Komplexität zu messen, ist die Berechnung des Speicherverbrauchs des neuen Modells. Dieser ist für jedes Modell anders zu ermitteln. Die Aussage über den Speicherverbrauch bezieht sich auf eine optimale Sortierung der Parameter. Diese Sortierung wird zwar berechnet, aber nicht innerhalb des Modells geändert, da es in Rapidminer nicht entsprechend implementiert ist.

Die aufwendigste Heuristik für die Berechnung liegt bei Naive Bayes. Aufwendig ist es deshalb, da es hier die meisten Möglichkeiten gibt, um die Parameterwerte zu sortieren (s. Abb. 13). Betrachtet man die Abbildung, liegt die Vermutung nahe, dass man für eine optimale Sortierung jede mögliche Kombination der Sortierung der einzelnen Blöcken betrachten muss. Dieses ist aber sehr teuer, da man bei n Parametern $n!$ Möglichkeiten der Anordnung hat, was einer Laufzeit von $O(n^n)$ entspricht, also mit einem großen n beliebig groß wird. Auch wenn der Wertebereich der a-priori-Wahrscheinlichkeiten

häufig nicht sehr groß ist, so ist dies für die Blöcke von $\mathcal{X}$ nicht gegeben, weshalb eine Betrachtung aller möglichen Sortierungen nicht realistisch ist. Deshalb wird eine Heuristik genutzt. Wie schon in der Abbildung beschrieben, kann man den obersten Block mit den a-priori-Wahrscheinlichkeiten sortieren, womit sichsich auch die jeweiligen Blöcke von $\mathcal{X}$ bedingt durch $\mathcal{Y}$ umsortieren. Das heißt, man hat die besten Intervalle innerhalb von $\mathcal{Y}$ gefunden und betrachtet nicht die weiteren Kombinationen der Blöcke für $\mathcal{X}$. Dieses gilt auch für die Sortierung der einzelnen Blöcke für $\boldsymbol{x}_i$. Der erste vorkommende Block wird sortiert und alle weiteren entsprechend übernommen. Zusätzlich ist es auch noch möglich die Blöcke von $\mathcal{X}$ als kompletten Block zu vertauschen, da es egal ist, an welcher Stelle diese stehen. So kann z. B. der Block $\boldsymbol{x}^5$ mit dem Block $\boldsymbol{x}^{m-8}$ vertauscht werden.

Die Heuristik der SVM beschreibt, dass zunächst alle α_n optimal sortiert werden und danach, anhand dieser Reihenfolge, die zugehörigen $\boldsymbol{x}_n$ (s. Abb. 14). Für die einzelnen Blöcke von $\boldsymbol{x}_n$ verfährt man genau wie auch bei Naive Bayes. Hier wird ebenfalls der erste Block vertauscht und alle anderen dementsprechend angepasst.

Bei den Entscheidungsbäumen stellt man sich die Bäume wie in Abbildung 15 dargestellt vor. Anhand dieser Vorstellung wird dann die erste Zeile der Matrix sortiert. Also werden die einzelnen Bäume anhand der optimalsten Sortierung der ersten Verzweigungen in den Bäumen vertauscht. Da nicht alle Bäume die gleiche Tiefe haben, kann es in den unteren Zeilen viele Nullen geben, die keine Aussage über eine optimale Sortierung liefern. Deshalb wurde hier festgelegt, die erste Zeile zu nutzen.

Für die ganzen Sortierungen ist es wichtig, eine neue Struktur anzulegen, damit nachher noch bekannt ist, an welcher Stelle welcher Wert zu finden ist. Dafür wird eine Struktur von Paaren benötigt, wo jeweils die erste Spalte sortiert wird und die zweite, in der die Positionen gespeichert sind, sich automatisch mitsortiert.

4.4 LogPerformance

Um die Ergebnisse komfortabel miteinander vergleichen zu können, benötigt man einen Operator, der die Ergebnisse in eine Logdatei schreibt. Einen solchen Operator gibt es in Rapidminer schon, mit dem Namen *log.* Allerdings erfüllt dieser nicht alle Anforderungen, die hier benötigt werden. Das Problem des gegebenen Operators ist es, dass man nur für eine Art der berechneten Performanz die Standardabweichung ausgeben kann. Gesucht war aber eine Möglichkeit, von allen errechneten Parametern die Standardabweichung auszugeben. Dafür wurde der vorhandene Operator erweitert und ein neuer Operator *LogPerformance* implementiert. Dieser hat alle Eigenschaften und Funktionen des ursprünglichen Operators. Neben dem ursprünglichen Inputport vom Typ *through,* an dem jedes anliegende Objekt ohne Änderung zum entsprechenden Outputport des gleichen Typs geliefert wird, verfügt der Operator zusätzlich noch über einen weiteren Inputport vom Typ *performance.* Werden nun, z. B. mit dem Operator *Complexity,* mehrere Werte ausgegeben, so fügt der Operator jeden dieser Werte und die entsprechende Standardabweichung hinzu. Benutzt man die schon vorhandene Option *persistent*, werden die Ergebnisse während des laufenden Prozesses ausgegeben und können innerhalb des Ergebnisbildschirms von Rapidminer beobachtet werden.

Im folgenden Kapitel wird die Verwendung der hier vorgestellten Operatoren innerhalb der Experimente vorgestellt.

5 Experimente

Nachdem alle Voraussetzungen für die Experimente geschaffen wurden, wird in diesem Kapitel das in Kapitel 3 erläuterte Vorgehen getestet. Die Experimente in Rapidminer sollen nun zeigen, ob die Hypothese, dass durch die gewählte Maßnahme ähnlich gute, aber kleinere Modelle erzeugt werden können, verworfen werden soll. Das Ziel ist es, mithilfe von unüberwachten Lernverfahren, die auf überwachten Lernverfahren lernen, ein möglichst kleines, aber trotzdem gleich gutes Modell, ähnlich des Ausgangsmodells, zu erhalten. Es werden die Genauigkeit sowie die Modellkomplexität der Modelle vor und nach der Anwendung bestimmt. Dafür werden die einzelnen Vorgehen kreuzvalidiert. Für die Durchführung werden die oben beschriebenen, neu implementierten Operatoren angewandt.

Alle unüberwachten Lernverfahren sollten ursprünglich mit zwei verschiedenen Abstandsmaßen durchgeführt werden, um zu evaluieren, welche Auswirkungen die gewählte Metrik (s. Definitionen 5 und 6) hat. Nachdem sich in den Experimenten aber herausgestellt hat, dass der Unterschied nicht von großer Bedeutung ist, wurde dieses Experiment nur mit Naive Bayes und auf dem Datensatz Sonar durchgeführt (s. Abschnitt 5.3.6). Damit man eine möglichst breite und allgemeingültige Aussage treffen kann, werden Experimente sowohl mit möglichst vielen verschiedenen Methoden als auch Datensätzen durchgeführt.

Zusätzlich soll die Frage beantwortet werden, ob die Daten der Stützvektoren bei der SVM wichtig sind und inwieweit sich ein eventuelles Einbeziehen auf den Speicherverbrauch sowie die Qualität des Modells auswirkt.

5.1 Verwendete Methoden

Zusätzlich zu den in Kapitel 4 beschriebenen Operatoren, werden für die Experimente eine Vielzahl anderer Operatoren aus Rapidminer benötigt. Eine Übersicht findet sich im Anhang in Tabelle 5.

Um eine ausreichende Basis für die Untersuchung der aufgestellten Hypothesen zu bekommen, werden die Experimente auf verschiedenen Datensätzen durchgeführt, die im nächsten Abschnitt weiter erläutert werden. Die überwachten Lernverfahren werden in unterschiedlichen Formen angewendet. Folgende Kombinationen von Lernverfahren wurden in den Experimenten genutzt:

- SVM (mit RBF-Kernel)
- SVM + k-means
- SVM + DBSCAN
- Naive Bayes
- Naive Bayes + k-means

- Naive Bayes + DBSCAN
- Random Forest
- Random Forest + k-means
- Random Forest + DBSCAN
- Lineare Regression
- Lineare Regression + k-means
- Lineare Regression + DBSCAN
- SVR (mit RBF-Kernel)
- SVR + k-means
- SVR + DBSCAN

Da die Algorithmen viele dieser Methoden mehrere Hyperparameter haben, die man unterschiedlich einstellen kann, wurde hier der Operator *Loop Parameter* genutzt. Hier kann man einstellen mit welchen Kombinationen die Lernaufgabe gelöst werden soll. Nachfolgend sind die verschiedenen Werte der einzelnen Hyperparameter aufgelistet, die für die Experimente genutzt wurden:

- SVM:
  - γ: 0.001, 0.01, 0.1, 1, 10
  - C: 1, 10, 100, 1000
- Random Forest:
  - Anzahl der Bäume: 10, 100, 1000
  - Maximale Tiefe der Bäume: 16, 32, 64

Beim Verfahren DBSCAN ist es nicht einfach, vorab ein sinnvolles ϵ anzugeben, da es auf den Abstand der Parameter des Datensatz ankommt. Deshalb wurden hier die Werte exponentiell von 0.001 bis 10 gewählt. Einzig bei Naive Bayes ist das ϵ unabhängig von den Daten, aber nur für den Fall, dass sie diskret sind, da alle Parameter $\log$-Wahrscheinlichkeiten entsprechen und damit die gleichen Distanzen untereinander haben. Will man hier die Wahrscheinlichkeiten zusammenfassen, enspricht $\epsilon = 0,01$ einem Unterschied von 1 % zwischen zwei Parametern.

- DBSCAN:
  - ϵ: 0.001, 0.01, 0.1, 1, 10
  - Anzahl der $minpts$: 2, 4, 8

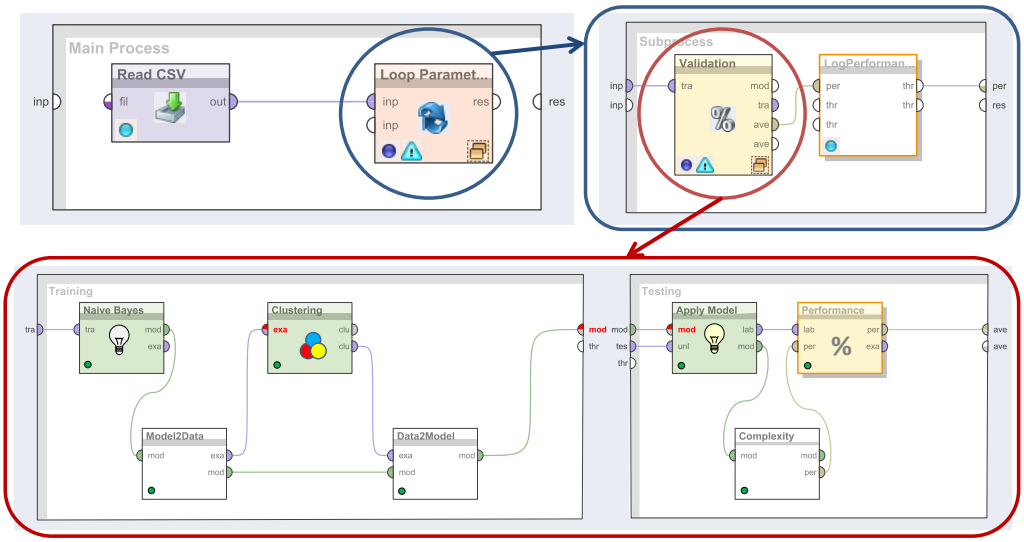

Abbildung 19: *Ein Beispiel für einen in den Experimenten genutzten Prozess. Oben links sieht man die Hauptebene des Prozesses. Hier sind der Datensatz und der Loop Operator zu sehen. Mit einem Doppelklick auf den Loop Operator kommt man in den Subprozess, welcher oben rechts dargestellt ist. Hier befinden sich die Kreuzvalidierung und der Operator, welcher die Perfomance loggt. Durch einen weiteren Doppelklick gelangt man in die Kreuzvalidierung mit zwei Seiten: Testing und Training. Hier befindet sich der eigentliche Prozess. Auf der Trainingsseite befinden sich das überwachte Lernverfahren, das unüberwachte Lernverfahren und die beiden Verbindungsoperatoren Model2Data und Data2Model. Auf der Testingseite befinden sich die Operatoren Apply Model, Complexity und ein Performance Operator.*

Die Werte, die hier für k-means und DBSCAN angegeben sind, werden für alle überwachten Lernverfahren genutzt, um ein möglichst breites Spektrum an Werten abzudecken. Man kann sich aber auch für die einzelnen Verfahren plausible Werte für k überlegen.

- k-means:
  - k: 2, 4, 8, 16, 32, 64, 128

Generell ist k zwischen 1 und d (Anzahl der Parameter in dem Parametervektor), wobei sich d bei den hier verwendeten Methoden wie folgt berechnen lässt.

Bei der SVM ist es zum einen vom gegebenen Lernproblem abhängig und zum anderen davon, wie gut der Kernel geeignet ist. Ist dieser nicht optimal gewählt, kann es passieren, dass die SVM den Datensatz auswendig lernt und alle Beispiele als Stützvektor klassifiziert. Man kann die Dimension d wie folgt angeben: $d = S + S \cdot n$, wobei S die Anzahl der Stützvektoren ist und n die Dimension von $\boldsymbol{x}$. Ist nun ein geeigneter Kernel gewählt, dann ist $S \ll N$. Wenn N die Anzahl der Datenpunkte darstellt, ist der Kernel nicht geeignet und die SVM lernt nur auswendig. Dann entspricht $S \sim N$.

Bei der linearen Regression lässt sich die Dimension einfacher angeben. Hier ist $d = n + 1$, wobei n die Anzahl der Gewichte darstellt, auf die der Bias addiert wird.

Die Dimension des Parametervektors bei Naive Bayes entspricht $d = |\mathcal{Y}| + \sum_{i=1}^{n} |\mathcal{X}_i| \cdot |\mathcal{Y}|$. Dieses gilt allerdings nur für diskrete Werte, da bei numerischen Werten die Standardabweichung und der Mittelwert gespeichert werden und somit $|\mathcal{X}_i| = 2$ ist.

Für ein Ensemble von Entscheidungsbäumen lässt es sich im Vorhinein nicht so einfach sagen, wie die maximale Tiefe l der Bäume aussieht. Hierfür müsste man sich alle Bäume ansehen, da die Bäume unterschiedlich tief sein können. Also lässt sich d hier nur beschränken. Da es sich um Binärbäume handelt, lässt sich folgende obere Schranke angeben: $d \leq T \cdot (2^l - 1)$, mit T gleich der Anzahl der Bäume. Genauer kann man es nur angeben, wenn man weiß, wieviele Variablen numerisch sind, da nur diese betrachtet werden.

Generell ist d für die Modellkomplexität verantwortlich und hat auch Auswirkungen auf DBSCAN, denn je mehr Werte man clustert, desto mehr verschiedene Cluster kann es geben. In den Experimenten wird gezeigt, dass man nicht d Parameter braucht, sondern weniger, was einen geringeren Speicherverbrauch zur Folge hat.

Beispielprozess Die hier verwendeten Prozesse bestehen aus drei Ebenen. Ein typischer Prozess, wie in den Experimenten genutzt, ist in Abbildung 19 zu sehen. Die in der Abbildung weiß dargestellten Operatoren wurden für diese Arbeit implementiert. In der Hauptebene befinden sich der Datensatz und der Operator *Loop Parameter*. Auf dieser Ebene wurde die Option *random seed* auf den Wert -1 ([7]) gesetzt. Die zweite Ebene befindet sich in dem Operator *Loop Parameters*. Hier befindet sich die Kreuzvalidierung (*Validation*) und der Operator, der die errechneten Daten in eine Datei schreibt (*LogPerformance*). Mit Doppelklick auf die Kreuzvalidierung gelangt man auf die dritte Ebene. Innerhalb der Kreuvalidierung findet man nun den eigentlichen Prozess. Hier wird, durch ein überwachtes Lernverfahren, das Modell gelernt, mit Model2Data in ein `ExampleSet` umgewandelt und vom unüberwachten Verfahren geclustert. Dies geschieht auf den Daten, die von der Kreuzvalidierung als Trainingsdaten ausgewählt wurden. Der zweite Teil der Kreuzvalidierung besteht aus dem Testing. Dort wird das gelernte Modell auf einen Testdatensatz angewendet und die Komplexität und Genauigkeit berechnet.

Einstellungen der einzelnen Operatoren und deren Parametern Die Operatoren haben verschiedene Optionen, die vom Benutzer eingestellt werden können. Im Folgenden werden alle geänderten Optionen aufgelistet. Nicht erwähnte Optionen oder Operatoren behalten die Standardeinstellungen.

[7] Das heißt es soll ein zufälliger random seed verwendet werden, damit die Resultate der Experimente sich immer unterscheiden.

SVM

- γ und C werden durch den Loop Operator auf die oben genannten Werte eingestellt
- handelt es sich bei der Lernaufgabe um eine Regression wird als svm type die epsilon-SVR gewählt

Lineare Regression

- feature selection: none
- eliminate colinear features: aus
- ridge: 0.0 (d.h. es wird eine einfache least squares Regression durchgeführt)

Random Forest

- minimal gain: 0.001
- no pre pruning: an
- no pruning: an
- guess subset ratio: aus und den Wert auf 0.01

Complexity

- hier werden alle Haken gesetzt

5.2 Verwendete Datensätze

In Tabelle 4 sind die genutzen Datensätze und die Rahmeninformationen aufgelistet.

Iris Der Iris-Datensatz wurde nur zu Testzwecken während der Implementierung der Operatoren genutzt und nicht zur Untersuchung der Hypothese, soll hier aber nicht verschwiegen werden. Es handelt sich hierbei um einen bekannten und beliebten Datensatz, der zur Klassifikation dient. 1936 von Ronald Aylmer Fisher das erste Mal erwähnt [33], werden die Daten auch heute noch häufig zu Testzwecken genutzt und sind in Rapidminer unter `\\Samples\Data` zu finden. Der Datensatz besteht aus 150 Beobachtungen, von denen jeweils 50 einer von drei verschiedenen Klassen von Schwertlilienarten zugehörig sind. Alle 150 Instanzen haben jeweils vier Attribute, die anhand von Länge und Breite der Blütenblätter erzeugt wurden. Der Datensatz ist vollständig und hat keine fehlenden Werte [34].

Name	Lernaufgabe	# Klassen	# Instanzen	# Attribute
Iris	Klassifikation	drei (jew. 33,3 %)	150	4
Sonar	Klassifikation	zwei (46,6 %, 53.3 %)	208	60
Dorothea	Klassifikation	zwei (9,7 %, 90,2 %)	1.950	100.000
Insight Flow	Regression	zwei	12.528	2.372
Insight Sat	Regression	zwei	12.528	2.372

Tabelle 4: *Verwendete Datensätze. Die Tabelle zeigt die Eigenschaften der Datensätze, die in dieser Arbeit als Grundlage für die Experimente herangezogen wurden. Insight ist hier zweimal aufgeführt, da der Datensatz verschiedene Informationen beinhaltet, die auf zwei Datensätze aufgeteilt wurden.*

Sonar Auch dieser Datensatz ist von Haus aus in Rapidminer zu finden und ist ein Datensatz, der die Klassifikation zur Aufgabe hat. Hier gibt es zwei unterschiedlich verteilte Klassen. Zum einen die Klasse Stein mit 46,6 % und die Klasse Mine mit 53,3 %. Ziel ist es auch hier, die 208 Beispiele mit jeweils 60 Attributen einer der beiden Klassen zuzuordnen. Der Datensatz stammt aus dem Jahr 1987 und hat seinen Namen von der Technik, die genutzt wurde, um einen Stein bzw. eine Mine zu erkennen [35]. Dieser Datensatz ist einer der beiden, die in dieser Arbeit zur Klassifikation genutzt wurden.

Dorothea Dorothea ist der zweite Datensatz, der hier zur Klassifikation verwendet wurde und hat 1.950 Beispiele mit jeweils 100.000 Attributen. Die Daten sind in zwei Klassen aufgeteilt: A und I. Es geht darum, chemische Zusammensetzungen zu finden, die sich aktiv (A) an Thrombin binden oder es nicht tun und somit inaktiv (I) sind. Der Datensatz wurde für den KDD (Knowledge Discovery in Data Mining) Cup 2001 von der Firma *DuPont Pharmaceuticals* zur Verfügung gestellt.

Ursprünglich war geplant, die Experimente auf dem kompletten Datensatz sowie auf kleineren Teildatensätzen durchzuführen. Dafür wurde die Anzahl der Attribute von 100.000 auf 8.000, 32.000 und 64.000 verringert. Doch auch diese Anzahl von Daten konnte in der gegebenen Zeit, mit den zur Verfügung stehenden Ressourcen, nicht komplett verarbeitet werden. So wurde nur der Datensatz mit 8.000 Attributen und 1.950 Beispielen verwendet. Für die Anwendung der SVM wurde eine weitere Verkleinerung durchgeführt, sodass hierfür nur 1.024 Attribute und 384 Beispiele zu bearbeiten waren. Die Anzahl der Beispiele ergibt sich aus der Anzahl der beiden Klassen A und I. Hier wurden jeweils 192 Beispiele beider Klassen verwendet. Zum Ende der Bearbeitungszeit dieser Arbeit stellte sich heraus, dass auch diese Größe des Datensatzes nicht von Rapidminer in der vorgegebenen Zeit bearbeitet werden konnte. Aus diesem Grund liegen keine Daten für Dorothea und die SVM vor.

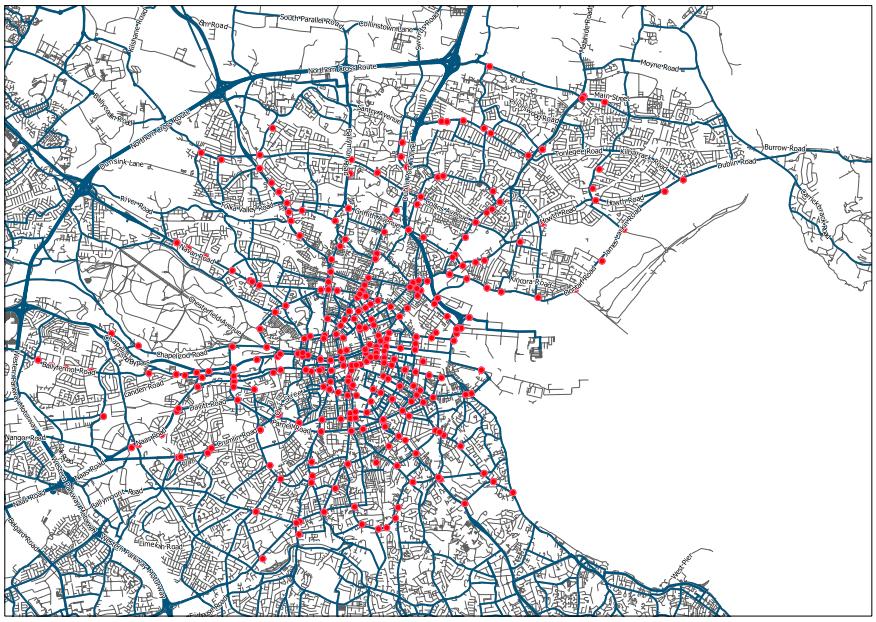

Abbildung 20: *Sensoren in Dublin. Die Abbildung zeigt eine Karte von Dublin, Irland. Die roten Punkte sind installierte Sensoren, die bestimmte Eigenschaften des Straßenverkehrs messen.*

Insight Insight ist ein EU Projekt[8], welches zum generellen Ziel hat, den Umgang mit Notfallsituationen in Städten zu verbessern. Teil dieses Projektes ist die Aufzeichnung des Straßenverkehrs in Dublin. Dafür wurden 2.372 Sensoren in der ganzen Stadt installiert (s. Abb. 20), welche die relative Anzahl der vorbeifahrenden Fahrzeuge und die Detektorauslastung des jeweiligen Straßenabschnitts gemessen haben. Die Daten der einzelnen Sensoren liegen vor.

Der Datensatz hat 12.581 Beispiele und 2.372 Attribute (Sensoren), wobei er aber einige fehlende Werte aufwies. Da nicht alle genutzten Methoden mit diesem Umstand umgehen können, wurden die entsprechenden Zeilen und Spalten mithilfe von Rapidminer herausgefiltert (s. Abb. 21). Zudem hat der Originaldatensatz die Daten minutengenau geliefert, was eine viel zu große Datenmenge war, weshalb die Daten für die Experimente in *slots* von 15 Minuten umgewandelt wurden. Daraus ergab sich ein Datensatz mit 12.528 Beispielen und 2.245 Attributen. Doch wie auch der Dorothea Datensatz, erwies sich auch dieser als zu groß, um mit Rapidminer in angemessener Zeit bearbeitet werden zu können. Aufgrund dessen wurde die Anzahl der Attribute, in diesem Fall die Anzahl der Sensoren, auf 1.024 und die der Beispiele auf 64 reduziert. Die Berechnung des Originaldatensatzes hätte mehrere Wochen gedauert. Die Reduzierung der Sensoren wurde manuell durchgeführt. Eine andere Möglichkeit wäre gewesen, innerhalb von Rapidminer die geeignetsten Sensoren auszuwählen, indem eine *forward feature selection* durchgeführt wird. Hierbei werden viele Regressionen angewendet und jeweils die beste Spalte hinzugefügt, bis sich eine gewünschte Anzahl von Spalten ergibt. Allerdings wäre dieses Verfahren zu langwierig geworden und hätte in der gegebenen Zeit nicht

[8] http://insight-ict.eu/ (Stand 19.09.2015)

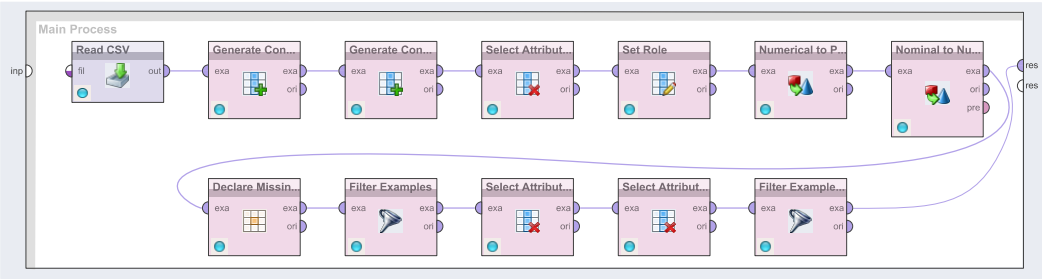

Abbildung 21: *Prozess in Rapidminer. Die Abbildung zeigt eine Aneinanderkettung verschiedener Operatoren in Rapidminer. Ziel hierbei ist es, den Datensatz, welcher im ersten Operator eingelesen wird, mithilfe der Folgeoperatoren in eine Form zu bringen, in der keine fehlenden Daten mehr vorhanden sind.*

durchgeführt werden können. Bei Anwendung der dualen SVM auf diesem reduzierten Datensatz, ist die Auswahl der Spalten ohnehin irrelevant, da es für die gewünschte Untersuchung des Speicherverbrauchs keinen Unterschied macht, welche Sensoren ausgewählt werden. Die Regression hingegen liefert, aufgrund der nicht durchgeführten *forward feature selection,* mit der manuellen Auswahl eventuell nicht den besten Wert. Das Ergebnis lässt sich aber dennoch ausreichend interpretieren.

5.3 Ergebnisse der einzelnen Experimente

Die Ergebnisse werden einzeln für die jeweiligen Kombinationen der überwachten Lernverfahren mit den unüberwachten Lernverfahren dargestellt. Die Bezeichnung *Accuracy*, die sich in den Diagrammen wiederfindet, ist die relative Häufigkeit der korrekt klassifizierten Beispiele. Nachfolgend auch häufig *Genauigkeit* genannt.

5.3.1 Ergebnisse SVM

Da es aufgrund der großen Anzahl von Daten nicht möglich ist, alle Kombinationen abzubilden, werden hier nur jene, die bei dem Experiment ohne Clustering, die beste Accuracy lieferten, betrachtet und verglichen. In dem Fall der SVM ergab sich die höchste Genauigkeit für $\gamma = 0.1$ und $C = 100$. Für jede Methode gibt es für jede Kombination vier verschiedene Plots. Diese beinhalten die unterschiedlichen Möglichkeiten, die Komplexität zu berechnen (s. Kapitel 3.2). In jedem Diagramm wurde zudem die berechnete Standardabweichung eingetragen. Wie schon erwähnt, liegen für dieses Verfahren nur die Ergebnisse für den Datensatz Sonar vor, da die Bearbeitung des Dorothea Datensatzes länger als zwei Wochen gedauert hat und aufgrund dessen abgebrochen werden musste.

Vergleich der Ergebnisse bezüglich der Daten der Stützvektoren Zu Beginn der Arbeit gab es die Überlegung, dass es sich eventuell als sinnvoll erweisen könnte, die Daten der Stützvektoren nicht mitzuclustern und stattdessen nur die α_i zu betrachten.

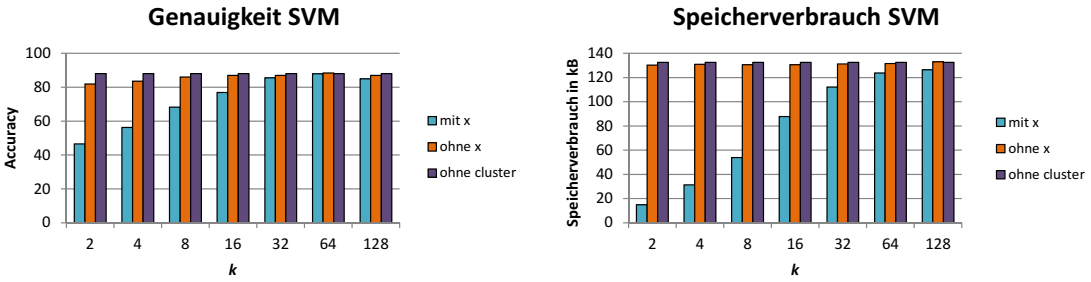

Abbildung 22: *Vergleich SVM. Diese Abbildung zeigt den Vergleich zwischen dem Ursprungsmodell, dem Modell ohne den Daten der Stützvektoren und dem Modell mit den Daten. Als Clusterverfahren wurde hier k-means verwendet, was man auch an der X-Achse sehen kann. Hier sind die Werte der verwendeten k angegeben. Die Genauigkeit auf der linken Seite wird in Prozent gemessen und der Speicherverbrauch, im rechten Teil der Abbildung, in kB.*

Im Laufe der Experimente hat sich jedoch herausgestellt, dass dies keine Alternative darstellt, da ein Clustering ohne die Daten zu beachten, zwar eine höhere Genauigkeit, aber auch ein viel höheren Speicherverbrauch mit sich bringt. In Abbildung 22 sieht man die Ergebnisse unter Verwendung von k-means. Auf der linken Seite ist die Genauigkeit der einzelnen Verfahren abgebildet. Die Daten der Stützvektoren sind hier durch x gekennzeichnet. Entsprechend der Legende stellt der jeweils linke Balken die Genauigkeit unter Berücksichtigung der Daten dar. Der Balken in der Mitte zeigt die Ergebnisse für ein Clustering ohne Beachtung der Daten und der jeweils rechte Balken beschreibt die Werte von der SVM allein, ohne Clustering. Hierfür gibt es eigentlich nur einen Wert, da es kein k gibt. Zur besseren Veranschaulichung wurde dieser einzelne Wert für jeden Wert von k in das Diagramm eingetragen. In dem Diagramm sieht man nun deutlich, dass die Genauigkeit mit steigendem k, bei den Balken des Clusterings, ebenfalls ansteigt. Die Tatsache, dass diese Steigung bei den linken Balken deutlicher ist, liegt daran, dass es viel mehr Daten gibt, die zusammengefasst werden können. Bei einer kleinen Anzahl von Clustern, ist die Genauigkeit nicht so hoch, da viele wichtige Informationen durch das Clustering verloren gehen. Betrachtet man nur dieses Diagramm, kann man zu dem Schluss gelangen, dass es durchaus sinnvoll sein kann, die Daten der Stützvektoren nicht zu beachten, da hier die Genauigkeit von Anfang an höher ist. Aber die Hypothese, auf der die Arbeit beruht, sagt, dass das Ergebnis der vorgestellten Methode, ein Modell mit ähnlich gleicher Güte (Genauigkeit), aber weniger Speicherverbrauch ist. Aufgrund dessen muss man sich also auch den Speicherverbrauch ansehen, welcher auf der rechten Seite der Abbildung zu sehen ist. Hier sind die Balken genauso angeordnet, wie bei dem Diagramm der Genauigkeit und auch hier ist ein Anstieg mit größer werdendem k zu vermerken. Betrachtet man nun z. B. in beiden Diagrammen die Werte für $k = 16$, so kann man sehen, dass bei einem geringen Verlust der Genauigkeit ungefähr ein Drittel an Speicher eingespart werden kann, wenn man die Daten der Stützvektoren ebenfalls clustert. Lässt man sie außen vor, so ist der Speicherverbrauch fast genauso hoch wie vor dem Clustering. Man würde also keinen

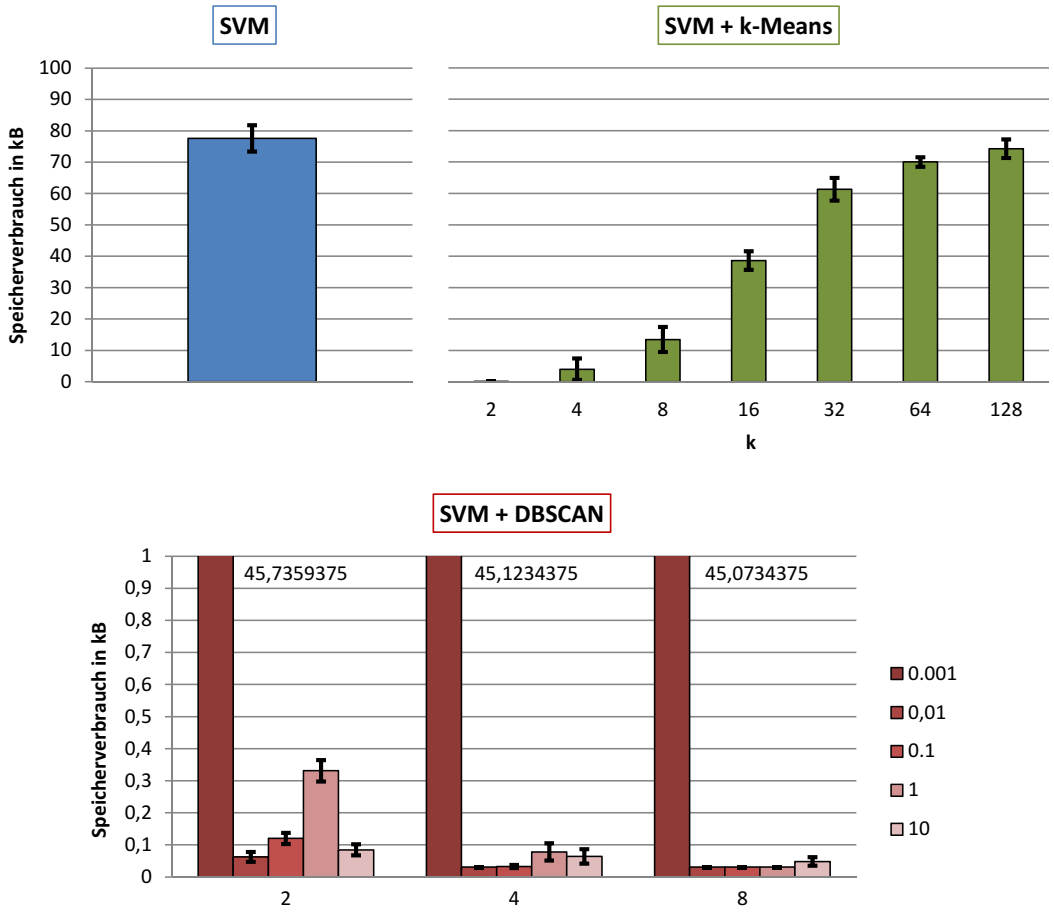

Abbildung 23: *Diagramm für den Speicherverbrauch der SVM. Die Diagramme in der Abbildung zeigen den unterschiedlichen Speicherverbrauch der SVM ohne Clustering, mit k-means und DBSCAN. Zu beachten ist hier, dass bei der Kombination SVM + DBSCAN eine andere Skalierung gewählt werden musste, da sonst die Werte von $\epsilon = 0.01$ bis 10 nicht zu erkennen wären.*

Verbrauch einsparen und könnte genauso gut das Ursprungsmodell nutzen.

Speicherverbrauch SVM In Abbildung 23 ist der Speicherverbrauch in kB der einzelnen Kombinationen zu sehen. Ganz links befindet sich der Balken mit dem Speicherverbrauch, wenn nur die SVM, also ohne Clustering, genutzt wurde. Dieser wird verglichen mit dem Speicherverbauch der SVM zusammen mit k-means und mit DBSCAN. In dem Diagramm, in dem k-means dargestellt ist, bezeichnen die Zahlen auf der X-Achse die verschiedenen k. Hier sieht man schon, dass selbst das Ergebnis mit $k = 128$, also mit 128 verschiedenen Clustern, einen, wenn auch nur geringfügig, kleineren Speicherverbrauch hat, als ohne angewandtes Clustering. Bei dem Diagramm in dem DBSCAN aufgezeigt wird, ist es wichtig zu erwähnen, dass die Skalierung des unteren Diagramms anders gewählt werden musste, da aufgrund der kleinen Werte sonst keine Balken zu

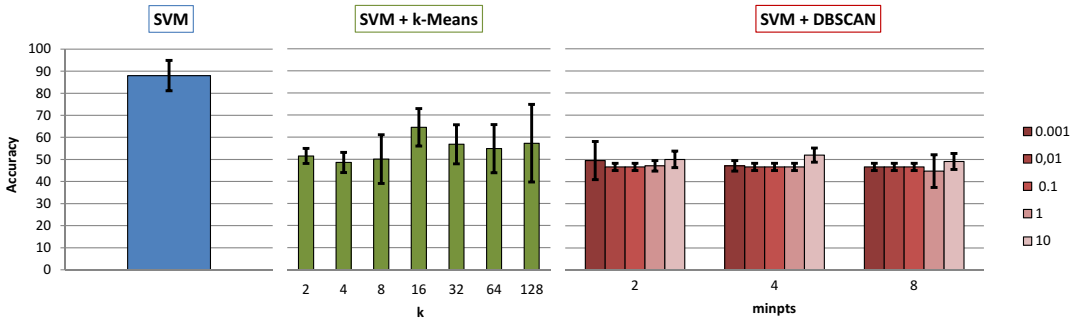

Abbildung 24: *Diagramm für die Genauigkeit der SVM. Hier wird die Genauigkeit der SVM, ohne und mit den verschiedenen Clusterverfahren, dargestellt. Man kann feststellen, dass der blaue Balken fast doppelt so hoch ist, wie die Balken von DBSCAN (rot). Das zeigt, dass, durch Anwendung des Clustering, viele wichtige Informationen verloren gehen, weshalb die Genauigkeit des Modells nicht mehr so gut ist.*

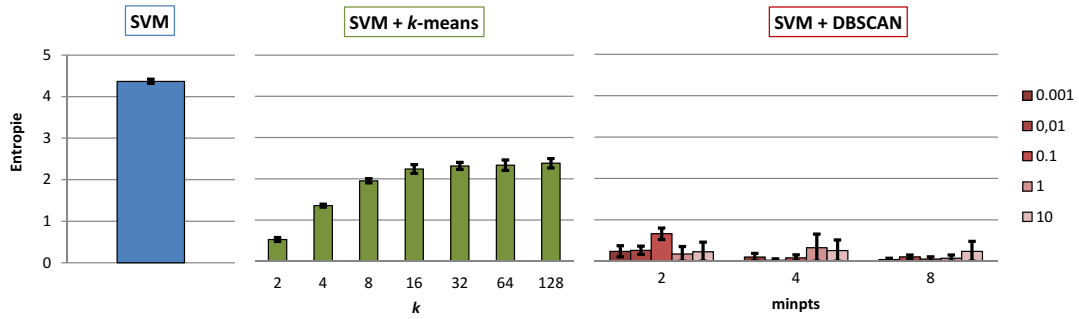

Abbildung 25: *Diagramm für die Entropie der SVM. Das Diagramm zeigt die Entropie der SVM (blau), für die SVM + k-means (grün) und die SVM + DBSCAN. Je kleiner die Entropie, desto besser ist es für die Komplexität des Modells. In diesem Fall ist die Entropie bei den Werten von DBSCAN sehr gering, was wiederum auf einen geringen Speicherverbrauch hinweist.*

sehen wären. Das bedeutet, dass der Speicherverbrauch nach Anwendung von DBSCAN viel kleiner ist als ohne Clustering. Die einzelnen Balken stellen jeweils einen anderen Wert für γ dar. Und zwar die unterschiedliche Anzahl der $minpts$ 2, 4 und 8.

Genauigkeit SVM Die Genauigkeit der SVM ist bei der Verwendung der einzelnen SVM etwas höher, als wenn zusätzlich ein Clusterverfahren angewendet wird. Dieses ist in Abbildung 24 zu sehen. Betrachtet man aber gleichzeitig den Speicherverbrauch und die Entropie, so ist ein geringer Verlust der Genauigkeit akzeptabel, bezüglich der Höhe der Daten, die abzuspeichern sind.

Entropie SVM Bei Betrachtung der Entropie (s. Abb. 25) gilt: je geringer der Wert, desto besser das Ergebnis. Auch hier wird demnach wiedergespiegelt, welch großen Nut-

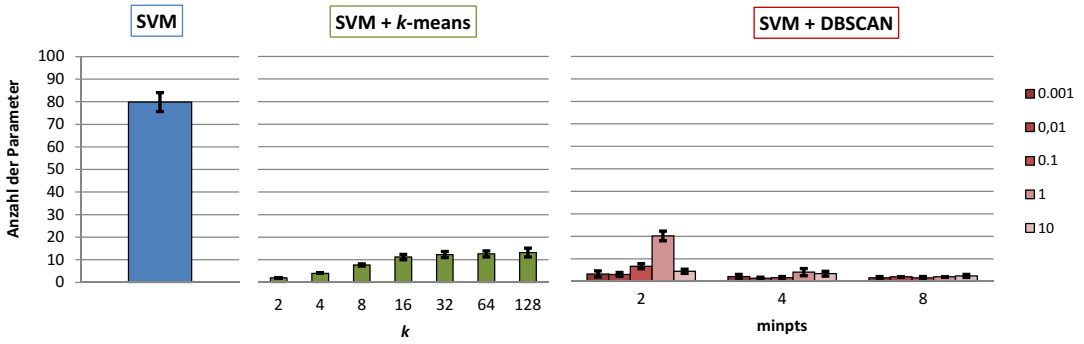

Abbildung 26: *Diagramm zur Anzahl der unterschiedlichen Parameter unter Verwendung der SVM. In diesem Diagramm ist klar zu sehen, dass die Anzahl von unterschiedlichen Parametern im linken Diagramm viel höher ist, als in den anderen beiden Teilen. Die grünen Balken sind sich alle ziemlich ähnlich, was daran liegt, dass es ab einem* $k = 16$ *keine große Änderung bezüglich der Anzahl der unterschiedlichen Werte gibt. Der Ausreißer bei einem Wert von* $\epsilon = 1$ *in dem rechten Diagramm liegt an der Anzahl der* $minpts$*. Bei einem Wert von 2 werden mehr Cluster gefunden, was den Anstieg der unterschiedlichen Werte erklärt.*

zen das Clustern der einzelnen Werte der SVM hat. Betrachtet man die Entropie der einzelnen SVM, so ist diese schon gegenüber der SVM in Verbindung mit k-means deutlich höher und beträgt ungefähr das Vierfache gegenüber der Nutzung von DBSCAN.

Anzahl der unterschiedlichen Parameter Anhand der Anzahl der unterschiedlichen Parameter lässt sich gut erklären, wieso der Speicherverbrauch als auch die Entropie bei der Verwendung zusätzlicher Clusterverfahren deutlich geringer sind, als unter der SVM alleine. In Abbildung 26 sieht man, dass die Anzahl bei der einzelnen SVM ungefähr achtmal höher ist, als bei den Werten der Clusterverfahren. Das bedeutet, dass es viel mehr unterschiedliche Werte gibt, die man abspeichern muss und sich damit viel weniger Möglichkeiten ergeben, diese Werte, z. B. mithilfe der Intervalldarstellung (s. Abschnitt 3.2), zusammenzufassen, um damit den Speicherverbrauch zu verringern.

5.3.2 Ergebnisse Naive Bayes

Das Verfahren Naive Bayes bringt mit sich, dass es keine eigenen Variablen hat, die man unterschiedlich einstellen kann, wie z. B. die SVM mit C und γ. Aufgrund dessen gibt es nicht so viele Kombinationen wie bei der SVM oder auch Random Forest und es können alle Ergebnisse betrachtet werden.

Speicherverbrauch Naive Bayes Die Ergebnisse zwischen den Datensätzen Sonar und Dorothea unterscheiden sich hier nicht besonders, weswegen an dieser Stelle in Abbildung 27 nur das Diagramm für die Daten von Sonar aufgezeigt ist. Das Diagramm für den Dorothea Datensatz findet sich im Anhang. Der blaue Balken zeigt wieder

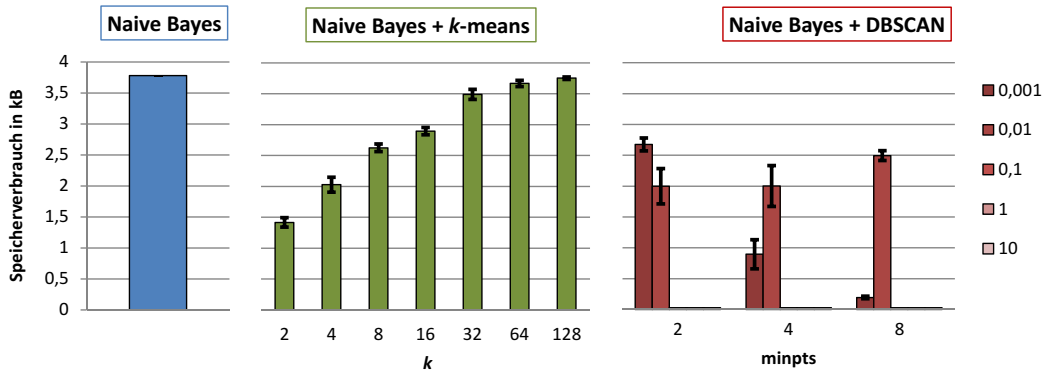

Abbildung 27: *Diagramm für den Speicherverbrauch von Naive Bayes. Dieses Diagramm stellt den Speicherverbrauch des Verfahrens Naive Bayes dar. Eine Besonderheit in dem Diagramm ist, dass die Werte für* $\epsilon = \{0,1; 1; 10\}$ *nicht vorhanden sind. Das liegt liegt daran, dass das* ϵ *hier zu groß war und nur eine geringe Anzahl an Clustern gefunden wurde, denen aber jeweils viele Datenpunkte zugeordnet wurden. Dadurch entstehen viele gleiche Werte und eine hohe Chance, diese zusammenfassen zu können. Allerdings leidet unter diesem Informationsverlust die Genauigkeit (s. Abb. 28 oben).*

den Speicherverbrauch, wenn kein Clustering verwendet wurde. Dem gegenübergestellt wurden die Diagramme mit Anwendung des Clusterings. Hier sind die Unterschiede nicht so hoch wie bei der SVM, was daran liegt, dass nicht so viele einzelne Werte vorhanden sind, die geclustert werden können. Aber auch hier sieht man, dass der Speicherverbrauch nach dem Zusammenfassen der Werte geringer ist. Die Einträge für $\epsilon = \{0,1; 1; 10\}$ sind nur der Vollständigkeit halber in der Legende aufgezeigt. Hier hat ϵ einen zu großen Wert, weswegen wenige Cluster mit vielen Datenpunkte entstehen, die alle den gleichen Wert bekommen. Für diese Werte ist die Genauigkeit um 20 % geringer.

Genauigkeit Naive Bayes Abbildung 28 zeigt die Genauigkeit des gelernten Modells auf den Datensätzen Sonar (oben) und Dorothea (unten). Um darzustellen, was die Größe des Datensatzes für Auswirkungen hat, werden hier beide Datensätze vorgestellt. Schon bei dem kleineren Datensatz sieht man deutlich, dass sich der Wert der Genauigkeit nach Anwendung von k-means gering vergrößert hat. Selbst im schlechtesten Fall, wenn der Wert von z. B. $k = 16$ nach unten abweicht, ist die Genauigkeit noch fast genauso hoch, wobei der Speicherverbrauch niedriger ist (s. Abb. 23). Noch deutlicher wird es bei Betrachtung des größeren Datensatzes. Hierbei ist die Genauigkeit auf 88 % bei k-means und sogar auf 92 % bei DBSCAN gestiegen. Bei einem deutlich gesunkenen Speicherverbrauch von knapp 500 kB ohne Anwendung von DBSCAN und auf 264 kB nach der Anwendung. Anhand dieser Beispiele lässt sich die Aussage der Hypothese der Arbeit mehr als verdeutlichen.

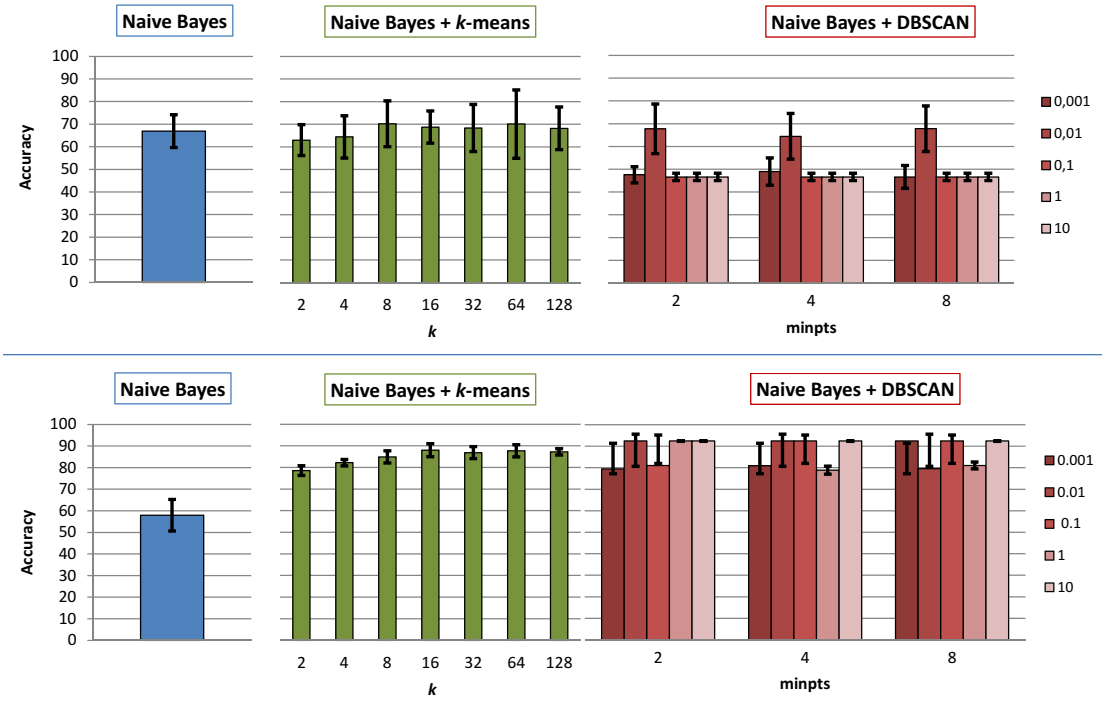

Abbildung 28: *Diagramm zur Genauigkeit von Naive Bayes. Die Abbildung zeigt zwei verschiedene Diagramme. Im oberen Teil ist der Datensatz Sonar aufgezeigt und im unteren Teil der Datensatz Dorothea. Hier soll gezeigt werden, wie viel die Anzahl und auch die Unterschiedlichkeit der Daten zur Genauigkeit des Modells beitragen. Im unteren Teil ist überraschenderweise die Genauigkeit des „normalen" Verfahrens schlechter, als unter Anwendung von einem Clusterverfahren. Das ist durchaus erstaunlich, denn durch das Zusammenfassen von Werten erwartet man eigentlich einen Informationsverlust, der hier allerdings nicht gegeben ist. Dies liegt an der Beschaffenheit und an der größeren Anzahl des Datensatzes.*

Entropie Naive Bayes Bezüglich der Entropie sind die Ergebnisse bei Naive Bayes sehr ähnlich zu denen der SVM. Auch hier sind die Werte nach Anwendung der Clusterverfahren viel kleiner. Das zugehörige Diagramm findet sich im Anhang.

Anzahl der unterschiedlichen Parameter Bei der Anzahl der unterschiedlichen Parameter sieht man, welch großen Unterschied die Anwendung eines Clusterverfahrens liefert. Betrachtet man den Wert ohne Clustering, so liegt dieser bei 242 verschiedenen Parametern. Der höchste Wert unter Verwendung von k-means ist nur halb so groß und auch die Werte von DBSCAN sind viel kleiner. Ebenso wie bei der SVM lässt sich auch hier die Speichereinsparung mit dem hohen Unterschied der verschiedenen Parameter zwischen den verschiedenen Kombinationen der Verfahren erklären.

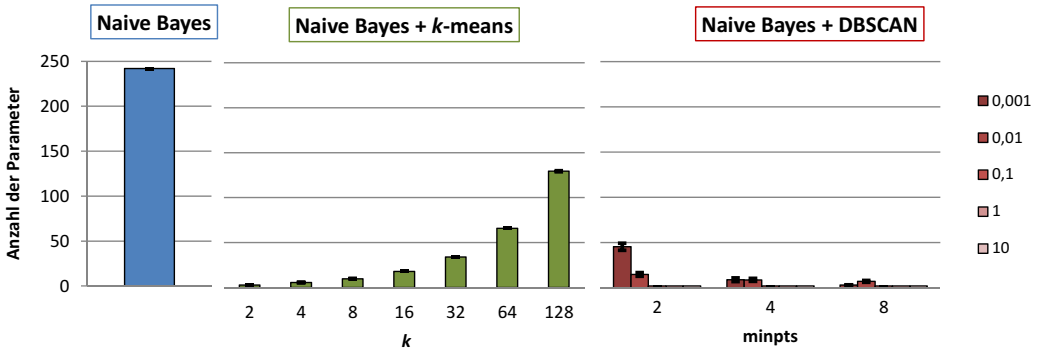

Abbildung 29: *Diagramm für die Anzahl unterschiedlicher Parameter bei Naive Bayes. In dem Diagramm sieht man die deutlichen Auswirkungen des Clusterns der einzelnen Werte des Modells.*

5.3.3 Ergebnisse Random Forest

Wie schon erwähnt, besteht der Datensatz Dorothea aus zwei Klassen: A und I. Da diese aber sehr unregelmäßig verteilt sind (s. Tabelle 4), wurde innerhalb von Rapidminer ein Operator genutzt, der nicht alle Werte des gegebenen Datensatzes nutzt, sondern nur eine bestimmte Anzahl an Samples. In diesem Fall wurden A (192) und I (200) fast gleich aufgeteilt, damit es eine geeignete Grundlage zum Lernen des Modells gab. Bei kleineren Experimenten zum Testen der Datensätze und Prozesse, hat sich herausgestellt, dass auch mit dieser Aufteilung Random Forest niemals A vorhergesagt hat, wenn zusätzlich Pruning oder Pre-Pruning genutzt wurde. Wie schon erläutert, kann Pruning als eine gewisse Art der Komplexitätsreduktion interpretiert werden. Da dieses aber die Methode, die hier zur Reduktion genutzt werden soll, unwirksam macht, wurde das Pruning bei den Experimenten ausgeschaltet. Ein weiterer Vorteil ist, dass man dadurch noch wichtige Kleinigkeiten in den Daten sehen kann, da die Baumstruktur nicht verändert wird. Zu klären ist die Frage, ob das Zusammenfassen der Daten ein adäquater Ersatz zum Pruning ist.

Bei der Darstellung der Ergebnisse für Random Forest, gilt es auf eine Besonderheit hinzuweisen. Nachdem die Daten des Datensatzes Dorothea geclustert wurden, haben alle Verzweigungen den Wert 0,5. Aufgrund dessen ist hierfür die Anzahl der verschiedenen Parameter immer gleich 1, da es nur den einen Wert gibt. Deswegen wird sowohl für die Entropie als auch für die Anzahl der unterschiedlichen Parameter der Sonardatensatz herangezogen, da die Diagramme für Dorothea keine Balken liefern. Für die Genauigkeit und den Speicherverbrauch sind die Daten beider Datensätze aufgezeigt, um darzustellen, welche Auswirkungen die Größe des Datensatzes bei diesem Verfahren hat.

Speicherverbrauch Random Forest In Abbildung 30 sieht man den Speicherverbrauch des Verfahrens in kB für Sonar (oben) und Dorothea (unten). Auffällig ist, dass hier bei beiden Datensätzen der Speicherverbrauch zwischen den einzelnen Methoden jeweils ähnlich ist.

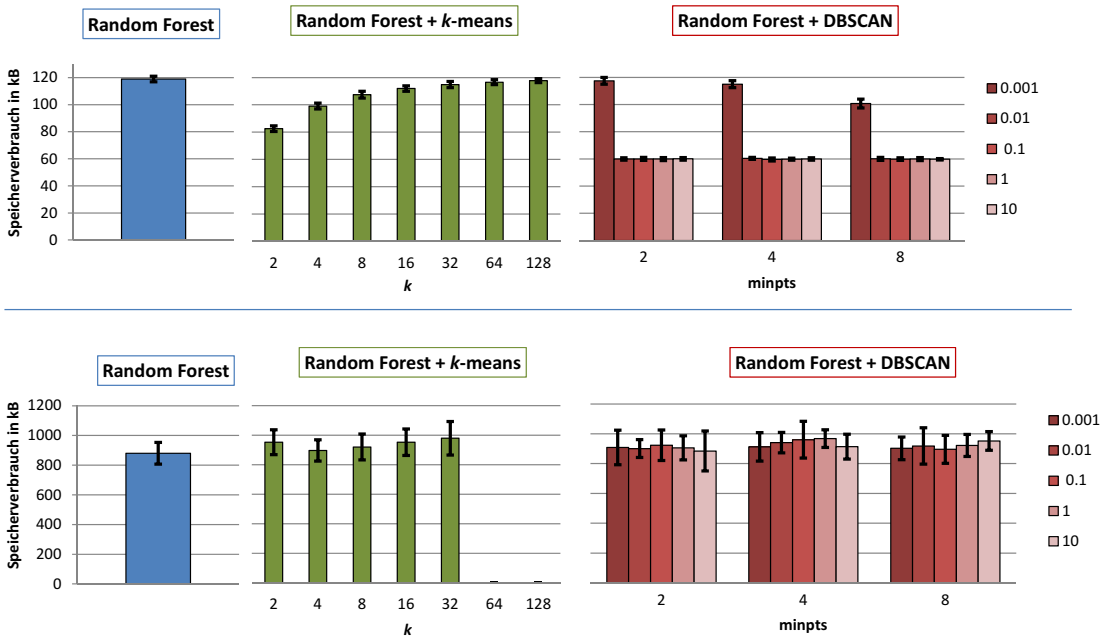

Abbildung 30: *Diagramm zum Speicherverbrauch von Random Forest. Hier sieht man Diagramme für zwei verschiedene Datensätze: Sonar (oben) und Dorothea (unten). Insgesamt sieht man, dass es keine großen Unterschiede zwischen den einzelnen Balken gibt. Mit Ausnahme der Balken für* $\epsilon = \{0,01; 0,1; 1; 10\}$ *bei DBSCAN im oberen Diagramm. Hier ist der Speicherverbrauch nur halb so groß, was daran liegt, dass der Wert für* ϵ *zu groß ist. Unter zusätzlicher Betrachtung von Abbildung 31 sieht man, dass sich auch dort die gleichen Werte von den anderen unterscheiden.*

Lediglich innerhalb des Sonardatensatzes ist der Speicherverbrauch der Werte $\epsilon = \{0,01; 0,1; 1; 10\}$ ungefähr nur halb so hoch, was man an der Ausprägung der Balken erkennen kann. Dies hängt damit zusammen, dass diese Werte für ϵ zu groß gewählt sind und sich damit nur ein großes Cluster finden lässt, in dem alle Parameter den gleichen Wert bekommen. Wie auch bei Naive Bayes sind hier die Werte der Genauigkeit um circa 20 % geringer. In dem unteren Diagramm der Abbildung ist der Speicherverbrauch im Schnitt sogar höher als ohne Anwendung des Clustering. Bei der Berechnung des Speicherverbrauchs muss man auch die Datenstruktur beachten. Diese kostet im Fall von Random Forest zusätzlich enormen Speicherverbrauch, sodass man auch bei wenigen Clustern noch viel Speicher verbraucht. Durch die Einhaltung der Datenstruktur hat das Modell, bei einer Einschränkung der Clusteranzahl, nicht so viel Spielraum, um sich genügend anzupassen. Dieses hat eine starke Reduktion der Modellkomplexität zur Folge, was aber nicht gleichbedeutend mit der Speicherkomplexität ist. Auch wenn das Modell klein ist, muss man sich die vorgegebene Datenstruktur mitabspeichern. Zudem fehlen hier die Werte für $k = \{64, 128\}$, da diese zu groß sind und nicht so viele Cluster gefunden werden konnten.

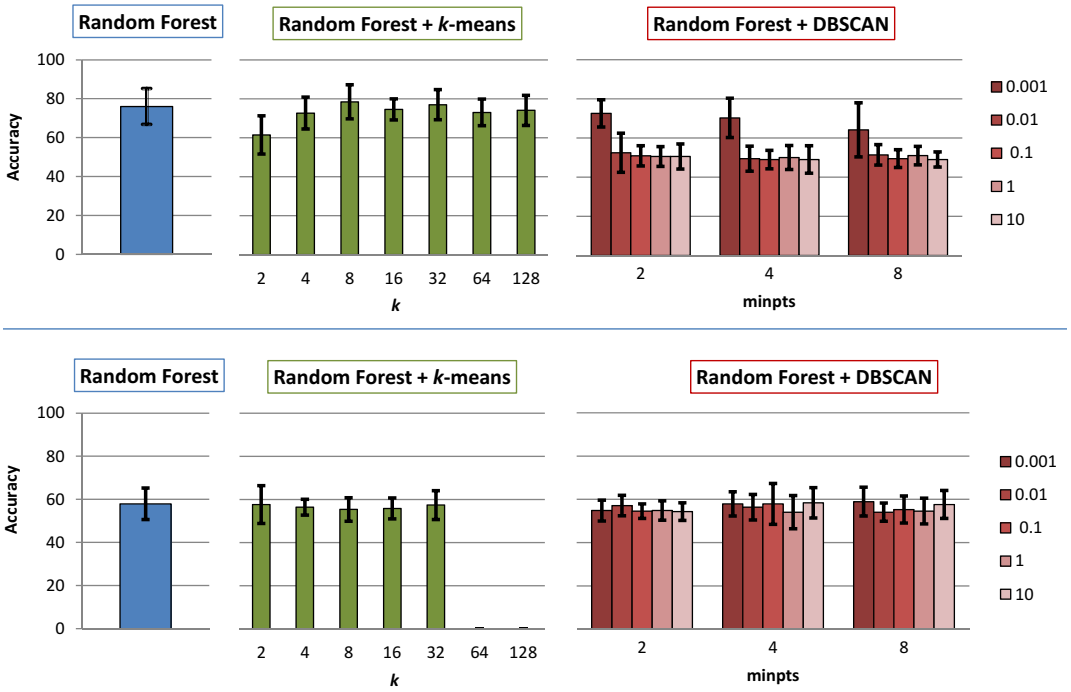

Abbildung 31: *Diagramm zur Genauigkeit bei Random Forest. Auch in dieser Abbildung sind wieder zwei Diagramme zu sehen. Oben das für den Datensatz Sonar und unten für den Datensatz Dorothea. Die Genauigkeit ähnelt sich bei allen Kombinationen für das Verfahren Random Forest. Daraus lässt sich ableiten, dass es hierbei keinen Unterschied macht, ob ein Clusterverfahren zusätzlich angewendet wird oder nicht.*

Genauigkeit Random Forest Die Diagramme bezüglich der Genauigkeit (s. Abb. 31) zeigen ein ähnliches Bild wie die des Speicherverbrauchs. Auch hier sind viele ähnliche Werte vorhanden, sodass es sich auf den ersten Blick zwar lohnen würde das Verfahren mit zusätzlichem Clustering anzuwenden, aber unter Einbeziehung des Speicherverbrauchs bringt es keinen Unterschied. Es ist hingegegen sogar aufwendiger und viel intensiver in der Rechenzeit. Zum Vergleich: das Berechnen der Daten ohne Clustering hat ungefähr 7 Minuten, das Berechnen unter DBSCAN hat 34 Stunden gedauert.

Entropie Random Forest Wie schon erwähnt, gibt es für die Entropie nur die Ergebnisse des Sonardatensatzes, welche in Abbildung 32 dargestellt sind. Wie auch bei dem Diagramm über den Speicherverbrauch, ist hier deutlich zu sehen, dass die Werte für ϵ zu groß sind, weshalb die entsprechenden Balken im Diagramm nicht vorhanden sind. Hier sieht man mit Anstieg von k ebenfalls einen Anstieg der Entropie, was daran liegt, dass mit einer höheren Anzahl der Cluster auch die Anzahl der unterschiedlichen Werte ansteigt. Bei DBSCAN verringert sich die Entropie mit zunehmender Anzahl der $minpts$, da dadurch weniger Cluster berechnet und mehr Werte zusammengefasst werden können. Die Entropie bei dem Datensatz Dorothea ist immer gleich null, wes-

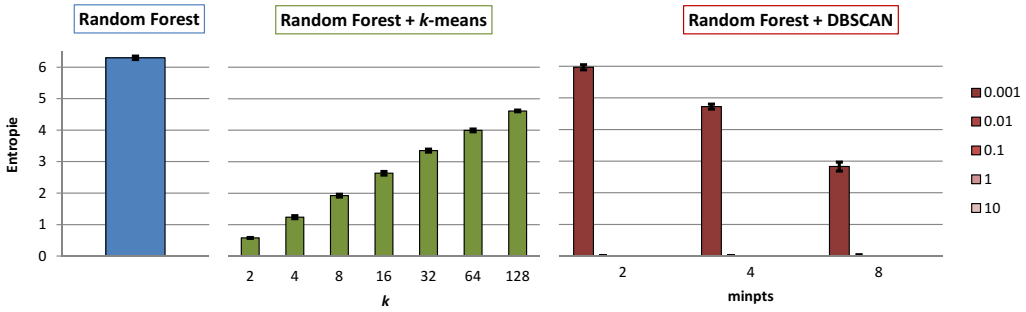

Abbildung 32: *Diagramm für die Entropie von Random Forest. Die Abbildung zeigt, wie sich die gewählte Kombination auf die Entropie des Modells auswirkt. Die höchste Entropie entsteht bei dem normalen Random Forest. Unter zusätzlicher Verwendung von k-means steigt die Entropie mit größer werdendem k ebenfalls an. Bei DBSCAN sind nur die Werte für $\epsilon = 0,001$ zu sehen, da es für die restlichen Werte keine aussagekräftigen Ergebnisse gibt, da ϵ zu groß ist.*

wegen hier kein Diagramm aufgezeigt ist.

Anzahl unterschiedlicher Parameter Das Diagramm für diese Möglichkeit, die Komplexität des Modells zu messen, ist sehr ähnlich zu dem vorherigen der Entropie und befindet sich aufgrund dessen im Anhang. Wie anfangs erwähnt, gibt es für bestimmte Werte von ϵ nur einen Wert nach Anwenden des Clustering. Das bezieht sich aber nur auf den Datensatz Sonar, da bei Dorothea alle Werte gleich 0,5 sind und es deswegen auch nur einen Wert für die Parameter gibt.

5.3.4 Ergebnisse lineare Regression

Sowohl für die lineare Regression als auch für die Regression mit der SVM wurde der Insight Datensatz als Grundlage genutzt. Wie schon in Abschnitt 5.2 erwähnt, veranschaulichen die Daten zum einen die Detektorauslastung des Straßenabschnitts und zum anderen die relative Anzahl vorbeifahrender Fahrzeuge an einem Sensor. Für die Experimente wurden diese Informationen getrennt, sodass sich ein Datensatz mit den Geschwindigkeiten und einer mit der Anzahl der Fahrzeuge ergab. Die usprüngliche Idee hier war, dass der Datensatz so bearbeitet wird, dass es am Ende möglich ist, die zukünftigen Ergebnisse der einzelnen Sensoren vorherzusagen. Dieses gestaltete sich aber als zu aufwendig, da viele Daten, aufgrund von ausgefallenen Sensoren oder Messfehlern, nicht vorhanden waren. Als Alternative wurde ein Sensor ausgesucht, der möglichst wenig Messfehler und fehlende Werte aufzeigte, und als Label deklariert. Damit bekommt man nun ein Vorhersagemodell für diesen einen Sensor für den Fall, dass er ausfallen sollte.

Da sich die Ergebnisse der Datensätze innerhalb der Diagramme sehr ähneln, wird hier nur der Datensatz, der die Geschwindigkeiten der Fahrzeuge darstellt, zur Veranschaulichung genutzt. Die Diagramme des anderen Datensatzes befinden sich zum Teil im

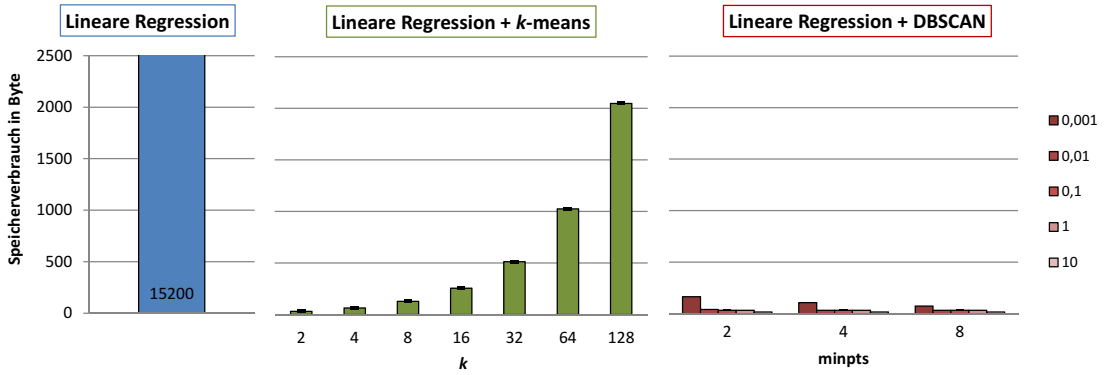

Abbildung 33: *Diagramm Speicherverbrauch der linearen Regression. Hier sieht man den Speicherverbrauch der linearen Regression. Die Auswirkungen des Clusterings sind hier sehr deutlich zu erkennen. Beträgt der Speicherverbrauch ohne Clustering noch 15200 Byte, so sinkt er unter Verwendung von k-means rapide auf einen maximalen Wert von 2000 Byte. Noch deutlicher sieht man es bei DBSCAN (rote Balken), hier beträgt der höchste Verbrauch 164,8 Byte und beträgt somit nur 1,084 % der einfachen linearen Regression.*

Anhang. Zum Teil nur deshalb, da die Experimente für die SVM nicht innerhalb der Bearbeitungszeit fertiggestellt werden konnten.

Speicherverbrauch lineare Regression Der Speicherverbrauch, der in Abbildung 33 aufgezeigt wird, ist in Byte angegeben, da die Werte für k-means, als auch für DBSCAN sonst zu gering gewesen wären. Aufgrund der Höhe des Speicherverbrauchs der linearen Regression, ohne dass ein Clustering durchgeführt wurde, ist der entsprechende Balken nicht komplett in dem Diagramm zu sehen. Der Wert innerhalb des Balkens bezeichnet die Höhe, also 15.200 Byte. Das ist mehr als das Siebenfache im Vergleich zu dem Verbrauch von k-means mit dem größten k. Hier steigt der Speicherverbrauch mit größer werdenden Anzahl von k exponentiell an, da die Parameter, wie in Abschnitt 3.2 gezeigt, beliebig umsortiert werden können. Der größte Wert der unter Verwendung von DBSCAN zu beobachten ist, liegt bei 164,8 Byte. Hier sieht man deutlich, wie viel Speicherverbrauch man einsparen kann, wenn ein Clusterverfahren angewendet wird.

Quadratischer Fehler lineare Regression Aus Abschnitt 2.2.4 ist bekannt, dass die Parameter mithilfe der Methode der kleinsten Quadrate (*least squares*) geschätzt werden. Aufgrund dessen wird hier nicht, wie bei einer Klassifikationsaufgabe, die Genauigkeit angegeben, sondern der quadratische Fehler, was dem Wert für $1 - Accuracy$ entspricht. Abbildung 34 zeigt das Diagramm für den errechneten durchschnittlichen quadratischen Fehler. Auch hier ist die Skalierung so gewählt, dass man möglichst viele Schlüsse aus dem Diagramm ziehen kann. Zwar ist der Wert bei der linearen Regression, als auch bei k-means, nicht exakt zu erkennen, aber das zeigt wiederrum wie klein

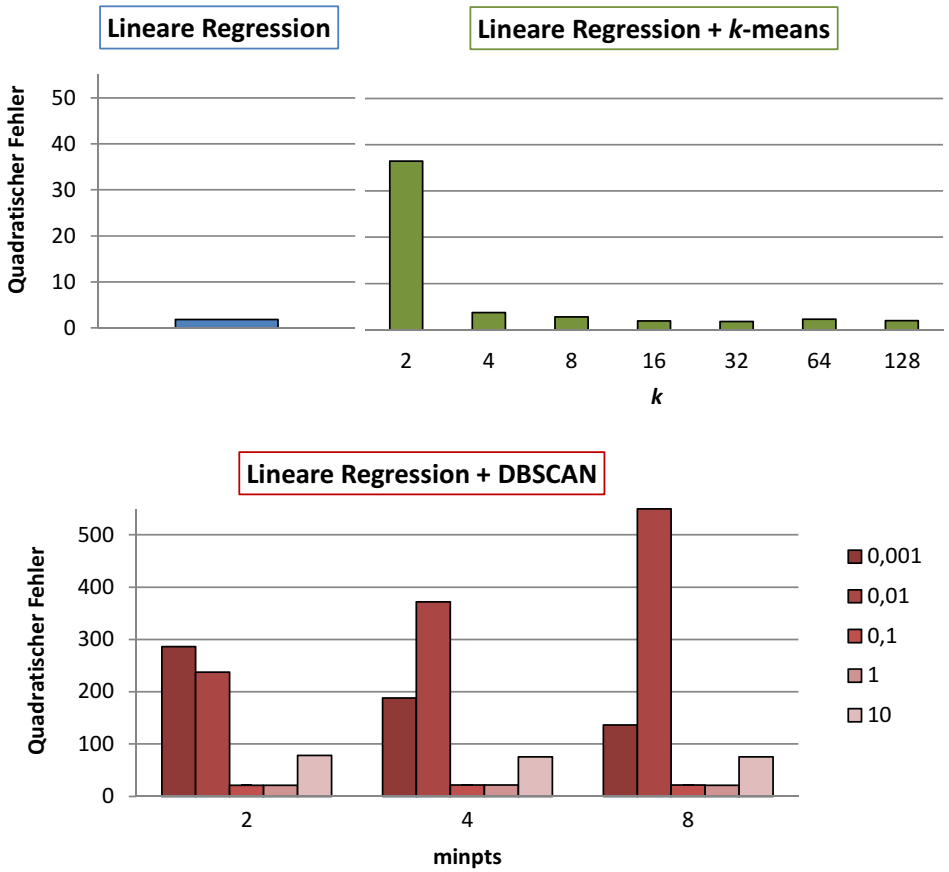

Abbildung 34: *Diagramm für den quadratischen Fehler der linearen Regression. Aufgrund der zu unterschiedlichen Werte musste das Diagramm aufgeteilt werden. Im oberen Teil sieht man die Werte für die normale lineare Regression und die Regression mit k-means. Im unteren Teil ist das Diagramm für die Regression und DBSCAN zu sehen. Hier ist die Skalierung zehnmal so groß wie im oberen Teil. Daran sieht man auch, dass DBSCAN für die lineare Regression nicht geeignet ist, um eine hohe Genauigkeit zu liefern.*

die Fehler im Gegensatz zu DBSCAN sind. Zusätzlich ist zu erwähnen, dass der Graph von DBSCAN multimodal ist. Hier ist es also wichtig, dass man unterschiedliche Werte für ϵ ausprobiert und keinen einfachen Greedy-Algorithmus benutzt, welcher terminiert sobald die Werte für den quadratischen Fehler größer werden. Was in diesem Fall dazu führen würde, dass die kleinsten Werte nicht gefunden werden. In diesem Diagramm wurde die Standardabweichung, aufgrund besserer Lesbarkeit, nicht eingetragen. Diese war fast immer höher als der zugehörige Balken. Das ist eine Eigenschaft des Datensatzes, in dem es offenbar eine sehr hohe Varianz der Daten gibt. Aus dem Diagramm geht hervor, dass der Fehler, wenn die Daten zusätzlich geclustert werden, um einiges höher ist als ohne Clustering. Damit lässt sich sagen, dass es bei der linearen Regression nicht sinnvoll ist, diese Methode anzuwenden, da dadurch viele wichtige Informationen verloren gehen können.

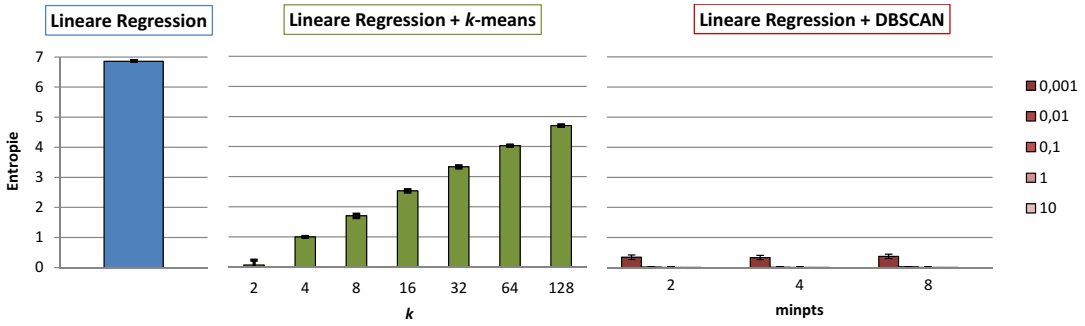

Abbildung 35: *Diagramm für die Entropie der linearen Regression. Das Diagramm zeigt einen deutlichen Unterschied zwischen den Methoden. Besonders zwischen der linearen Regression ohne Clustering und der mit zusätzlichem DBSCAN sind die Werte extrem verschieden.*

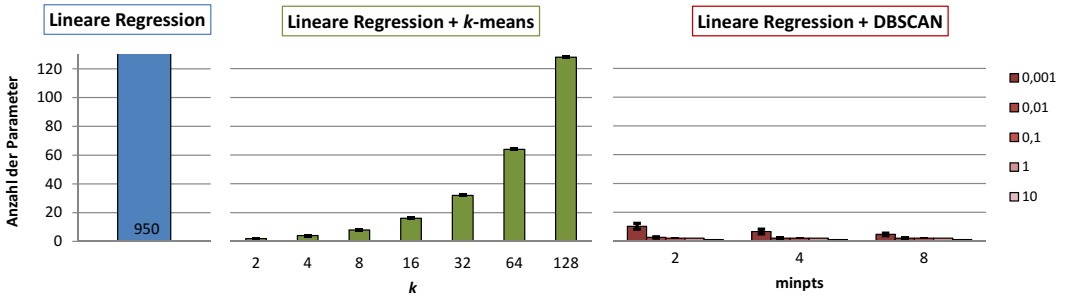

Abbildung 36: *Diagramm für die unterschiedlichen Parameter der linearen Regression. Hier sieht man deutlich den Unterschied, der durch das Clustern von Werten entsteht. Der blaue Balken ist ungefähr achtmal so hoch wie der höchste Balken von k-means. Aufgrund der Skalierung wurde der Wert in das Diagramm geschrieben. Aber auch so kann man sehen, dass sich die Werte sehr voneinander unterscheiden. Die kleinsten Werte liefert DBSCAN. Hier hätte man eventuell noch kleinere Werte für ϵ wählen können, damit eine höhere Anzahl unterschiedlicher Parameter entsteht.*

Entropie lineare Regression Die Darstellung der Entropie in Abbildung 35 ist ähnlich dem Diagramm, welches den Speicherverbrauch angibt, wenn auch nicht mit so drastischen Unterschieden. Auch hier sieht man, dass der Balken für die normale lineare Regression höher ist als unter Anwendung der Clusterverfahren. Die Balken bei k-means steigen mit größer werdendem k an. Dies war auch zu erwarten, da sich die Entropie immer stark dem Speicherverbrauch ähnelt.

Anzahl unterschiedlicher Parameter Die Anzahl der unterschiedlichen Parameter (s. Abb. 36) beträgt ohne Clustering 950. Unter Anwendung von k-means steigen die Werte quadratisch an und entsprechen dem gewählten k. DBSCAN hingegen hat eine sehr geringe Zahl unterschiedlicher Parameter, was eventuell daran liegt, dass der Wert für ϵ zu groß gewählt wurde.

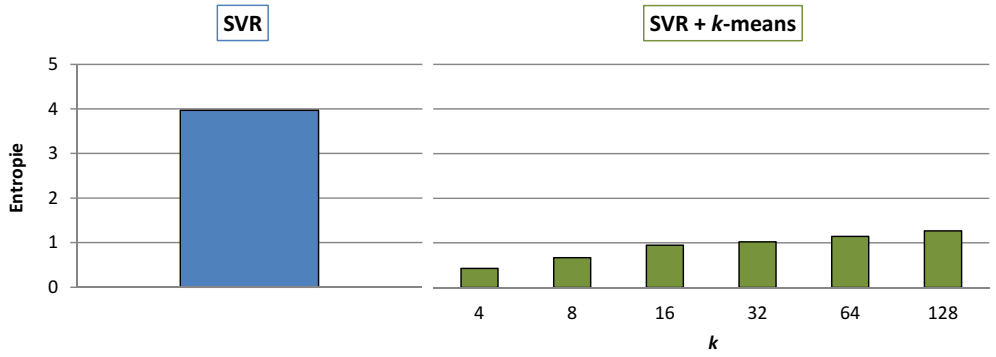

Abbildung 37: *Diagramm für die Entropie der SVR. Misst man die Komplexität mithilfe der Entropie, ergeben sich viel kleinere Werte unter Verwendung eines zusätzlichen Clusterverfahrens.*

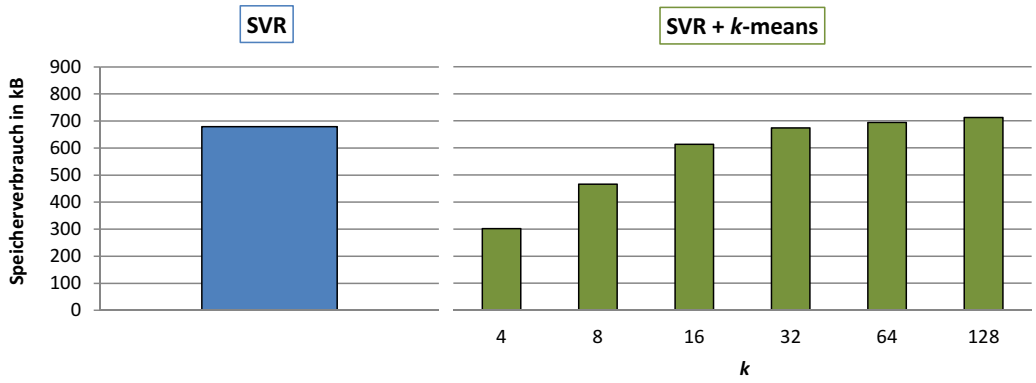

Abbildung 38: *Diagramm für den Speicherverbrauch der SVR. Bei größer werdendem k liefert das Clustern der Parameter ein ähnliches Resultat im Vergleich zu den ungeclusterten Parametern. Der Speicherverbrauch ist sogar noch ein wenig höher.*

5.3.5 Ergebnisse SVR

Die SVR ist eine SVM, die für Lernaufgaben der Regression bestimmt ist. Diese Experimente wurden ebenfalls auf den beiden Datensätzen von Insight gemacht. Allerdings sind hier die Ergebnisse nicht komplett. Nach einer Rechenzeit von zwei Wochen wurde der Prozess für die SVR in Verbindung mit DBSCAN abgebrochen. Die Ergebnisse beider Datensätze sind sich so ähnlich, dass auch hier nur einer aufgezeigt wird. Ähnlich den Diagrammen für die SVM, wird hier nur eine Kombination von γ und C, mit den Werten $\gamma = 0.001$ und $C = 10$, aufgeführt, da diese den geringsten quadratischen Fehler liefert. Der Datensatz mit den Daten für die Detektorenauslastung befindet sich im Anhang.

Speicherverbrauch SVR Das in Abbildung 38 aufgezeigte Diagramm zeigt den unterschiedlichen Speicherverbrauch zwischen der SVR und der SVR in Verbindung

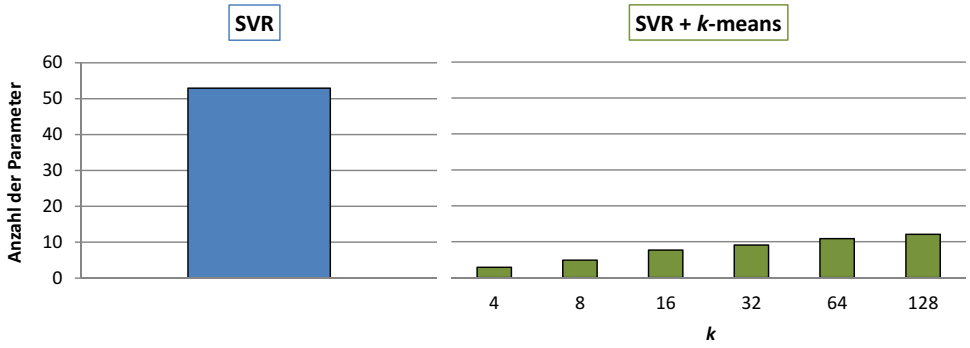

Abbildung 39: *Diagramm für die Anzahl der unterschiedlichen Parameter der SVR. Ähnlich dem Diagramm für die Entropie, ist auch hier der Unterschied zwischen den beiden Methoden deutlich zu erkennen. Nach Anwendung des Clusterverfahrens bleiben viel weniger unterschiedliche Parameter übrig.*

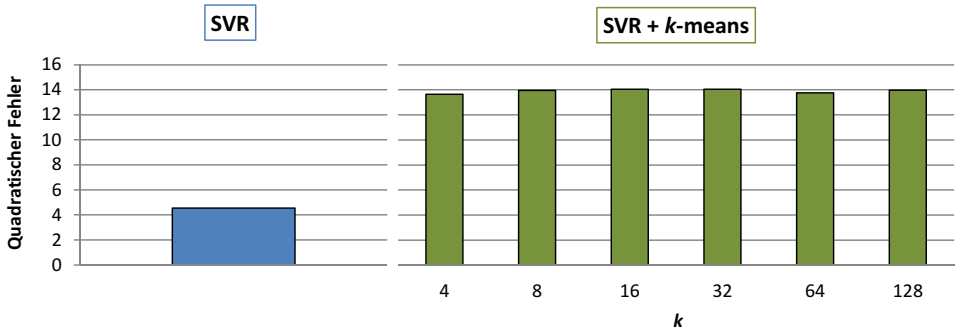

Abbildung 40: *Diagramm für den quadratischen Fehler der SVR. Hier sieht man einen deutlichen Unterschied zwischen den genutzten Verfahren. Ohne Verwendung eines Clusterverfahrens ist der Fehlerwert viel kleiner, als mit dem Clustering.*

mit k-means. Verwendet man zusätzlich ein Clusterverfahren, wird, bei einer kleinen Anzahl von Clustern im Vergleich zu vielen Clustern, weniger als die Hälfte an Speicher verbraucht. Betrachtet man nur die Speicherkomplexität des Modells, scheint es sich von Vorteil zu erweisen, zusätzlich ein Clusterverfahren anzuwenden, wobei gilt dies nur für kleine Werte von k gilt.

Quadratischer Fehler SVR Zieht man zur Betrachtung ein weiteres Diagramm hinzu, das bezüglich des quadratischen Fehlers des Modells (s. Abb. 40), so stellt sich heraus, dass bei nachfolgendem Clustering der Parameter ein viel höherer Fehlerwert entsteht, als würde man sich gegen das Clustering entscheiden. Der niedrigste Fehlerwert bei k-means beträgt 13,65, im Gegensatz dazu steht ein Fehlerwert von 4,55 bei der SVR. Schon an dieser Stelle lässt sich sagen, dass, ähnlich wie bei der linearen Regression, die Methode der SVR sich nicht eignet, um das in dieser Arbeit vorgestellte Verfahren anzuwenden.

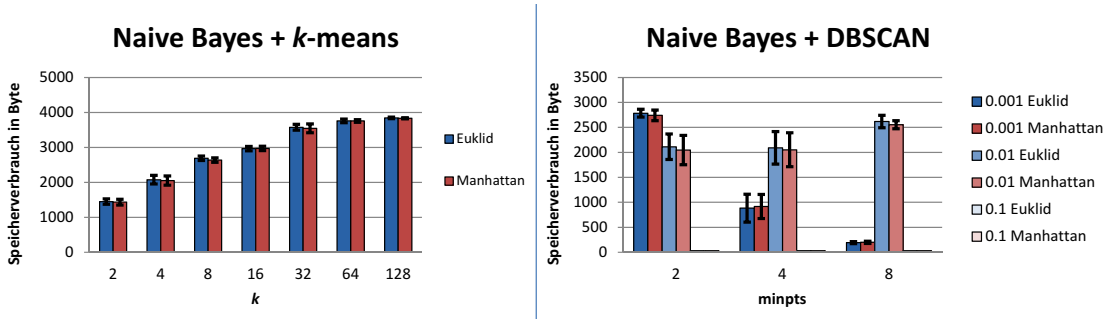

Abbildung 41: *Vergleich zwischen Euklidischem Abstand und Manhattan-Distanz. Das Diagramm verdeutlicht den Unterschied zweier unterschiedlicher Metriken. Der rechte, blaue Balken steht für den Euklidischen Abstand und der rote Balken für die Manhattan-Distanz. Alles in allem sind die Unterschiede der beiden Abstandsmaße nicht sonderlich groß, sodass es im Großen und Ganzen keine Rolle spielt, für welches Abstandsmaß man sich entscheidet.*

Entropie und Anzahl der unterschiedlichen Parameter der SVR Die Diagramme für die Entropie (s. Abb. 37) und die Anzahl unterschiedlicher Parameter (s. Abb. 39) zeigen sehr ähnliche Ergebnisse. Bei beiden ist der Wert ohne Clustering fast viermal so hoch wie bei dem Clustering mit k-means. Offensichtlich hat das Clusterverfahren eine große Auswirkung auf die Berechnung dieser Arten der Komplexität. Aus diesen Ergebnissen lässt sich auch ableiten, weshalb der Speicherverbrauch für kleine Werte von k geringer ist. Bei einer kleinen Anzahl von Clustern gibt es viele gleiche Werte, die zusammengefasst werden können. Aber trotz dieser guten Resultate spricht der berechnete quadratische Fehler dagegen, eine SVM für die Regression mit zusätzlicher Verwendung eines Clusterverfahrens zu nutzen.

5.3.6 Vergleich zwischen Euklidischem Abstand und Manhattan-Distanz

In Abbildung 41 wird der Vergleich zwischen den Abstandsmaßen Euklidischer Abstand und Manhattan-Distanz, anhand des jeweiligen Speicherverbrauchs in Byte, aufgezeigt. Dieser Vergleich wurde nicht für alle Methoden durchgeführt, sondern nur für Naive Bayes, da sich die Ergebnisse alle ähneln und die Berechnung mit diesem Verfahren vergleichsweise schnell ist. In der Abbildung stellen die blauen Balken immer den Euklidischen Abstand und die roten Balken die Manhattan-Distanz dar. Die Unterschiede zwischen den Balken sind minimal und es sind auch kaum Auffälligkeiten zu erkennen. Insgesamt kann man feststellen, dass der Speicherverbrauch unter Verwendung des Euklidischen Abstands immer ein wenig höher ist. Der Unterschied beträgt maximal 65,6 Byte und ist somit sehr klein. Aufgezeigt sind hier auch nur die Werte $\epsilon = \{0,001; 0,01; 0,1\}$, da die anderen Werte zu groß waren, um ein aussagekräftiges Ergebnis zu liefern. Alle anderen Experimente wurden mit dem Euklidischen Abstand durchgeführt.

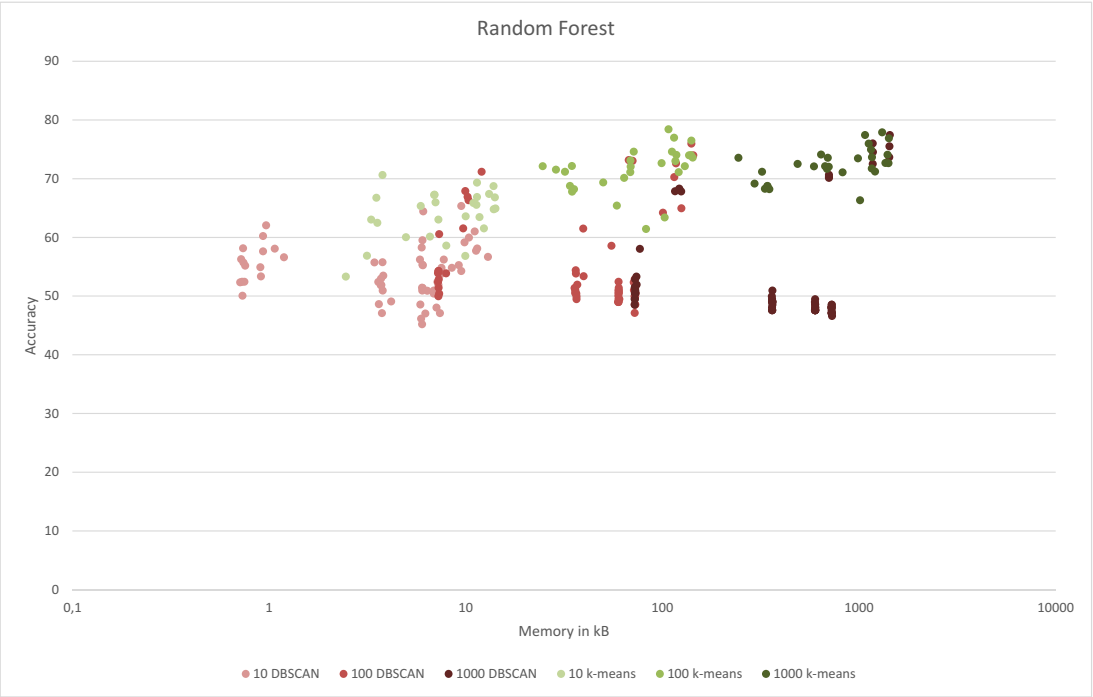

Abbildung 42: *Unterschiedliche Anzahl von Bäumen für k-means und DBSCAN. Das Diagramm zeigt die Ergebnisse der Clusterverfahren k-means und DBSCAN, wobei ersteres durch die grünen und letzteres durch die roten Punkte dargestellt wird. Hier ist für jedes Verfahren eine unterschiedliche Anzahl von verwendeten Bäumen aufgezeigt. Mit steigender Baumanzahl erhöht sich der Speicherverbrauch und für gewisse Kombinationen auch die Accuracy.*

5.3.7 Accuracy versus Speicherbedarf

Ein sicherlich interessanter Punkt besteht in der Betrachtung, wie sich die Genauigkeit gegenüber dem Speicherverbrauch verhält, wenn man gewisse Eingabeparameter der Methoden ändert. In Abbildung 42 wird die Auswirkung der Anzahl der berechneten Bäume bei Random Forest betrachtet. Es ist für k-means und DBSCAN jeweils die verschiedene Anzahl der Bäume eingetragen. Es lässt sich erkennen, dass sich mit steigender Anzahl der Bäume auch der Speicherverbrauch erhöht, aber nicht unbedingt die Genauigkeit besser wird. Hat man bei 100 Bäumen und DBSCAN z. B. eine Genauigkeit von 71 % und einen Speicherverbrauch von 12 kB, so erhöht sich bei 1.000 Bäumen der Speicherverbrauch auf 1.435 kB, wobei die Accuracy nur wenig auf 77 % steigt. Bei k-means sind ähnliche Ergebnisse zu erkennen. Insgesamt unterscheiden sich die Clusterverfahren darin, dass DBSCAN im Durchschnitt eine geringere Accuracy hat als k-means.

Abbildung 43 zeigt die Auswirkungen bei Änderung von C bzw. γ. Auffällig ist, dass die Änderung von γ als auch von C bei DBSCAN offenbar keinen großen Einfluss hat. Hier sind alle Werte, was die Höhe der Genauigkeit angeht, ungefähr gleich verteilt. Lediglich der Speicherverbrauch ist bei einigen Punkten höher als bei anderen, was von

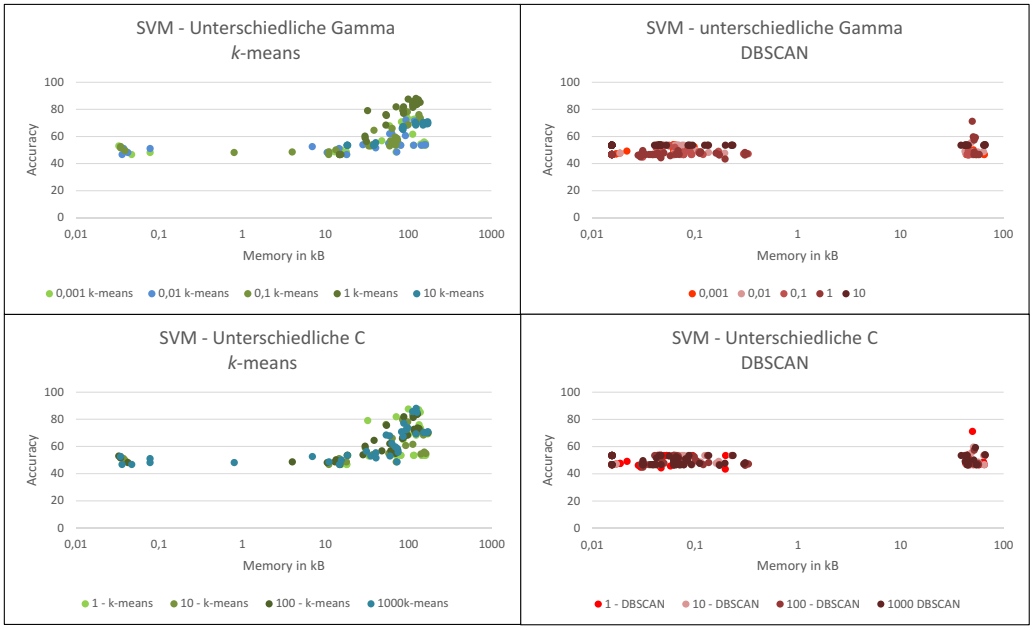

Abbildung 43: *Diagramme für die unterschiedlichen Werte von γ und C von der SVM. Die Diagramme auf der linken Seite sind jeweils für verschiedene Werte von γ und C für das Verfahren k-means. Auf der rechten Seite sind die Diagramme entsprechend für DBSCAN. Wichtig sind die unterschiedlichen Farben innerhalb der einzelnen Diagramme.*

dem gewählten Wert für ϵ abhängt. Auch unter Verwendung von k-means scheint es, als hätte das γ keinen großen Einfluss auf die Werte. Das einzig Auffallende ist, dass sich alle Werte, bis auf C=1, auf der X-Achse sehr breit gestreut befinden.

5.3.8 Übersicht der einzelnen Methoden

Um eine Übersicht zu bekommen, wo sich die Ergebnisse der einzelnen Methoden mit den verschiedenen Clusterverfahren bewegen, wurden die Ergebnisse in Abbildung 44 zusammengefasst. Zur besseren Orientierung wurden Hilfslinien eingefügt, die das Diagramm in Quadranten einteilen. Gewünscht ist eine hohe Genauigkeit sowie ein niedriger Speicherverbrauch. Dieses ist jedoch nur bedingt gegeben. Gute Ergebnisse erzielt das Verfahren Naive Bayes, wo im Durchschnitt wenig Speicher verbraucht wird, das gelernte Modell aber eine akzeptable Accuracy hat. Das Verfahren SVM in Verbindung mit DBSCAN ist weniger geeignet. Diese Werte befinden sich fast alle am unteren Rand des Wertespektrums.

Insgesamt ist dies eine gute visuelle Zusammenfassung der Experimente, da man einen groben Überblick über die Lage der einzelnen Verfahren bekommt.

Abbildung 44: *Diagramm zum Datensatz Sonar. Dieses auf den ersten Blick unübersichtliche Diagramm zeigt die Ergebnisse der Verfahren für die Klassifikation. Die Punkte stellen das Verfahren Naive Bayes dar, die Quadrate Random Forest und die Dreiecke sind die Ergebnisse der SVM. Die Verfahren sind ohne und mit Verbindung der zwei Clusterverfahren k-means und DBSCAN dargestellt.*

6 Zusammenfassung und Ausblick

Dieses letzte Kapitel fasst die Arbeit und die gewonnenen Ergebnisse zusammen und liefert einen Ausblick auf zukünftige Möglichkeiten der hier vorgestellten Idee. Zunächst ein Rückblick auf diese Arbeit.

Über eine kurze Einführung in das Themengebiet des maschinellen Lernens sowie einigen Grundbegriffen wurden die verschiedenen Lernaufgaben und Anwendungsgebiete vorgestellt. Darüber hinaus wurden die unterschiedlichen Arten des Lernens vorgestellt. Hierbei wurde der Unterschied zwischen überwachten und unüberwachten Lernverfahren betrachtet und die jeweils zugehörigen Lernverfahren vorgestellt. Verwendet wurden vier überwachte und zwei unüberwachte Verfahren, auch Clusterverfahren genannt.

In Kapitel 3 wurde zunächst erklärt, dass es innerhalb eines Parameterraums eine latente Struktur geben kann und wie sich diese darstellt. Auf diese Erklärung folgte eine Beschreibung des prinzipiellen Vorgehens, welches Kern dieser Arbeit war. Nachdem nun bekannt war, wie man sich der latenten Struktur innerhalb des Parameterraums nähern kann, wurden verschiedene Arten zur Berechnung der Komplexität vorgestellt. Dieses war notwendig, um eine Möglichkeit zu gewinnen, das Ursprungsmodell mit dem komprimierten Modell vergleichen zu können. Die unterschiedlichen Möglichkeiten, um die Komplexität des Modells feststellen zu können, wurden auf die einzelnen überwachten Lernverfahren angewendet. Aber auch andere, schon vorhandene Ansätze zur Komprimierung von Modellen, wurde betrachtet und kurz vorgestellt.

Im Anschluss wurde dargestellt, dass es mit den vorhandenen Methoden des Programms Rapidminer keine Möglichkeit gab, das Vorhaben umzusetzen. Zunächst folgte eine Beschreibung, welche Funktionen benötigt werden, aber nicht vorhanden sind. Aufgrund dessen wurden eigene Operatoren implementiert. Diese sind in Kapitel 4 beschrieben worden. Im Laufe der Arbeit wurden vier neue Operatoren erstellt, von denen drei einen großen Anteil an der Umsetzung der Experimente hatten und der vierte die Aufgabe der Erstellung einer Logdatei ausführte.

Nachdem nun alle Vorraussetzungen geschaffen wurden, konnten die Experimente durchgeführt werden, um die vorgestellte Hypothese zu untersuchen. Dieses wurde in Kapitel 5 beschrieben. Zu Beginn wurden die Hypothese und einzelne Teile des Vorgehens beschrieben. Darauf folgte eine Beschreibung der verwendeten Methoden und Datensätze, die hier genutzt wurden. Im Anschluss wurden die einzelnen Ergebnisse der Experimente dargestellt, grafisch veranschaulicht und am Schluss des Kapitels zusammengefasst.

6.1 Fazit

Insgesamt lässt sich sagen, dass die in den Kapiteln 1 und 3 formulierten Ziele weitesgehend erreicht und bestätigt wurden. Es wurden mit verschiedenen Lernverfahren unter zusätzlichem Clustering, Modelle mit annähernd gleicher oder sogar besserer Güte erzeugt, die weniger Speicherplatz verbrauchen als das jeweilige Ursprungsmodell.

Unter Verwendung der Support Vector Machine ist der Speicherverbrauch sehr deutlich

gesunken, wenn zusätzlich ein Clusterverfahren genutzt wurde. Allerdings ist hierbei auch die Genauigkeit gesunken, weswegen man abwegen muss, welcher Effekt stärker zu bewerten ist. Kann man sich damit abfinden, dass die Genauigkeit um mehr als 20 % sinkt, dann ist es denkbar dieses Verfahren anzuwenden. Das Verfahren Naive Bayes ist hingegen in beiden Fällen sehr gut geeignet, um die Aussage der Hypothese zu bestätigen. Hier ergaben die Experimente eine Einsparung des Speicherverbrauchs und, was noch wichtiger ist, eine Steigerung der Vorhersagegenauigkeit des Modells. Besonders bei dem Datensatz Dorothea ist eine sehr deutliche Steigerung zu erkennen. Hier ist die Genauigkeit der Clusterverfahren fast um 30 % gestiegen. Aber auch bei dem kleineren Datensatz Sonar ist eine mindestens ähnliche Genauigkeit zu verzeichnen.

Allerdings gibt es auch Ausnahmen. So ist das Verfahren Random Forest nicht gut geeignet, um zusätzlich ein Clustering darauf anzuwenden. Hier ergaben sich kaum Unterschiede sowohl was die Genauigkeit als auch den Speicherverbrauch der einzelnen Kombinationen betraf. Teilweise war der Speicherverbrauch sogar noch höher, nachdem Werte zusammengefasst wurden. Die Vermutung liegt nahe, dass es an der Baumstruktur liegt. Hier kann man nicht viel sortieren oder geschickt abspeichern (s. Abschnitt 3.2) und muss zusätzlich die Datenstruktur beachten, weshalb es zu einem höheren Speicherverbrauch kommt. Ein anderer Grund könnte die spezielle Implementierung des Verfahrens in Rapidminer sein. Eventuell bekommt man andere Ergebnisse, wenn man eine andere Implementierung nutzt.

Bei der Lernaufgabe der Regression liefert das Verfahren der linearen Regression in Verbindung mit k-means einen guten Beitrag zur Verifizierung der zugrunde liegenden Idee. In Verbindung mit DBSCAN scheint es jedoch nicht so gut geeignet. Zwar ist der Speicherverbrauch bei beiden Clusterverfahren niedrig, bei DBSCAN noch viel niedriger als bei k-means, aber der berechnete Fehler bei DBSCAN ist sehr viel höher. Somit ist eine Kombination der linearen Regression mit DBSCAN nicht empfehlenswert. Das Verfahren der SVR liefert keine guten Resultate für die vorgestellte Idee. Hier werden zwar Modelle mit einer kleinen Komplexität erzeugt, jedoch ist die Güte des Modells viel schlechter als die des Ursprungsmodells.

Die zu Beginn des Kapitels gestellte Frage, inwieweit sich das Einbeziehen der Stützvektordaten bei der SVM auswirkt, konnte auch beantwortet werden. Hier lässt sich sagen, dass das Beachten der Daten sehr wichtig für den Speicherverbrauch ist. Dieser ist in diesem Fall deutlich geringer, da es viel mehr Werte gibt, die zusammengefasst werden können. Auch wenn die Genauigkeit bei kleinen k nicht so hoch ist, so steigt sie bei einer größeren Clusteranzahl schnell an.

Insgesamt hätte man eventuell bei einigen Experimenten ein kleineres ϵ für DBSCAN wählen können, aber wie schon erwähnt, ist es schwierig diesen Wert vorher zu bestimmen, da man nicht weiß, wie weit die einzelnen Datenpunkte voneinander entfernt liegen. Dies ist je nach Datensatz verschieden.

6.2 Ausblick

Unter Betrachtung der Ergebnisse, welche diese Arbeit hervorgebracht hat, bleiben einige Fragen offen. Diese könnten in weiterführenden Untersuchungen betrachtet werden. Auffallend war das sehr gute Ergebnis von Naive Bayes, das sich im Vorhinein so nicht angedeutet hat. Hier wäre es möglich, eine theoretische Untersuchung zu führen, welche Gründe hier anzuführen sind. Eventuell hängt es mit der Skalierung der einzelnen Parameter zusammen.

Des Weiteren könnte es interessant sein, die Werte für ϵ aus den Daten besser zu schätzen und eventuell kleinschrittiger auszuwählen. Die Ergebnisse sind sehr stark von dem Wert von ϵ abhängig, weshalb sich vielleicht etwas andere Ergebnisse ergeben, wenn sich die Werte von ϵ nicht so stark voneinander unterscheiden bzw. besser an die Daten angepasst sind.

Wie schon erwähnt, sind die Ergebnisse für Random Forest natürlich von der gewählten Implementierung des Verfahrens in Rapidminer abhängig. Eventuell liefern hier andere Operatoren, die das Verfahren realisieren, bessere Ergebnisse. Ein weiterer Punkt, der dieses Verfahren betrifft, ist der mögliche Vergleich zwischen der hier vorgestellten Methode und dem Pruning eines Baums. Zu untersuchen wäre, ob es zu einer ähnlichen Komplexität des Modells führt oder ob es große Unterschiede gibt.

Da hier nur zwei Datensätze für jedes Verfahren getestet wurden, wäre es möglicherweise interessant, welche Ergebnisse größere und andere Datensätze liefern.

Literatur

[1] Nico Piatkowski, Sangkyun Lee, and Katharina Morik. Spatio-temporal random fields: compressible representation and distributed estimation. *Machine Learning*, 93(1):115–139, 2013.

[2] Günther Görz. *Handbuch der künstlichen Intelligenz*. Oldenbourg Verlag, 2003.

[3] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2003.

[4] Ethem Alpaydin and Simone Linke. *Maschinelles Lernen*. Oldenbourg Verlag, 2008.

[5] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[6] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

[7] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 1991.

[8] Christian Schawel and Fabian Billing. *Die Top 100 Management Tools*. Springer, 2004.

[9] Tom M Mitchell. *Machine learning. 1997*. Mcgraw-Hill Education.

[10] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 1–15, 2000.

[11] John Ross Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] Leonard A Breslow and David W Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(01):1–40, 1997.

[13] Vladimir N Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[16] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.

[17] Carl Edward Rasmussen. *Gaussian processes for machine learning*. Citeseer, 2006.

[18] Arthur Pap. *Analytische Erkenntnistheorie: kritische Übersicht über die neueste Entwicklung in USA und England*. Springer-Verlag, 1955.

[19] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[20] David J. Hand and Keming Yu. Idiot's bayes: Not so stupid after all? *International Statistical Review / Revue Internationale de Statistique*, 69(3):pp. 385–398, 2001.

[21] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

[22] Ludwig Fahrmeir, Thomas Kneib, and Stefan Lang. Lineare Regressionsmodelle. pages 59–188. Springer, 2007.

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[24] Christian Royer. *Simultane Optimierung von Produktionsstandorten, Produktionsmengen und Distributionsgebieten*. Herbert Utz Verlag, 2001.

[25] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):pp. 100–108, 1979.

[26] Sibylle Hess. Untersuchung von Code-Tabellen zur Kompression von binären Datenbanken, 2015.

[27] Quang-Anh Tran, Qian-Li Zhang, and Xing Li. Reduce the number of support vectors by using clustering techniques. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 2, pages 1245–1248, 2003.

[28] Wouter Duivesteijn. *Exceptional Model Mining*. PhD thesis, Universiteit Leiden, 2013.

[29] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[30] IDC. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. 2012.

[31] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[32] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):123–231, 2013.

[33] Ronald A Fisher. The use of multiple measurements in taxonomic problems. 1936.

[34] M. Lichman. UCI machine learning repository, 2013.

[35] R Paul Gorman and Terrence J Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural networks*, 1(1):75–89, 1988.

A Verwendete Operatoren in Rapidminer

Name	Funktion	Wo zu finden
Datensatz	Datensatz	Samples - data
read CSV	externen Datensatz importieren	Import - Data
Linear Regression	überwachtes Lernverfahren	Modeling - Classification and Regression - Function Fitting
Naive Bayes	überwachtes Lernverfahren	Modeling - Classification and Regression - Bayesian Modelling
LibSVM	überwachtes Lernverfahren	Modeling - Classification and Regression - Support Vector Modelling
Suport Vector Machine	überwachtes Lernverfahren	Modeling - Classification and Regression - Support Vector Modelling
Random Forest	überwachtes Lernverfahren	Modeling - Classification and Regression - Tree Induction
k-Means	unüberwachtes Lernverfahren	Modeling - Clustering and Sementation
DBSCAN	unüberwachtes Lernverfahren	Modeling - Clustering and Sementation
X-Validation	Kreuzvalidierung	Evaluation - Validation
Apply Model	Gelerntes Modell verwenden	Modeling - Model Application
Performance	Misst die Performanz	Evaluation - Perfomance Measurement - Classification and Regression
Model2Data	Wandelt ein Modell in ein ExampleSet um	Parameter Space Tools
Data2Model	Wandelt ein ExampleSet in ein Modell um	Parameter Space Tools
Complexity	Misst Komplexität eines Modells	Parameter Space Tools
LogPerformance	Loggt die Performance	Parameter Space Tools
Loop Parameters	Iteriert über die gegebenen Parameter	Process Control - Loop
Sample	kann spezielle Daten auswählen	Data Transformation - Filtering - Sampling - Sample

Tabelle 5: *Verwendete Operatoren. Die Tabelle zeigt alle in den Experimenten verwendeten Operatoren von Rapidminer und wo sie sich in der Struktur von Rapidminer befinden.*

B Zusätzliche Ergebnisse der Auswertung

Hier befinden sich die Diagramme, die aus Gründen der Übersichtlichkeit nicht in den jeweiligen Abschnitten aufgeführt sind.

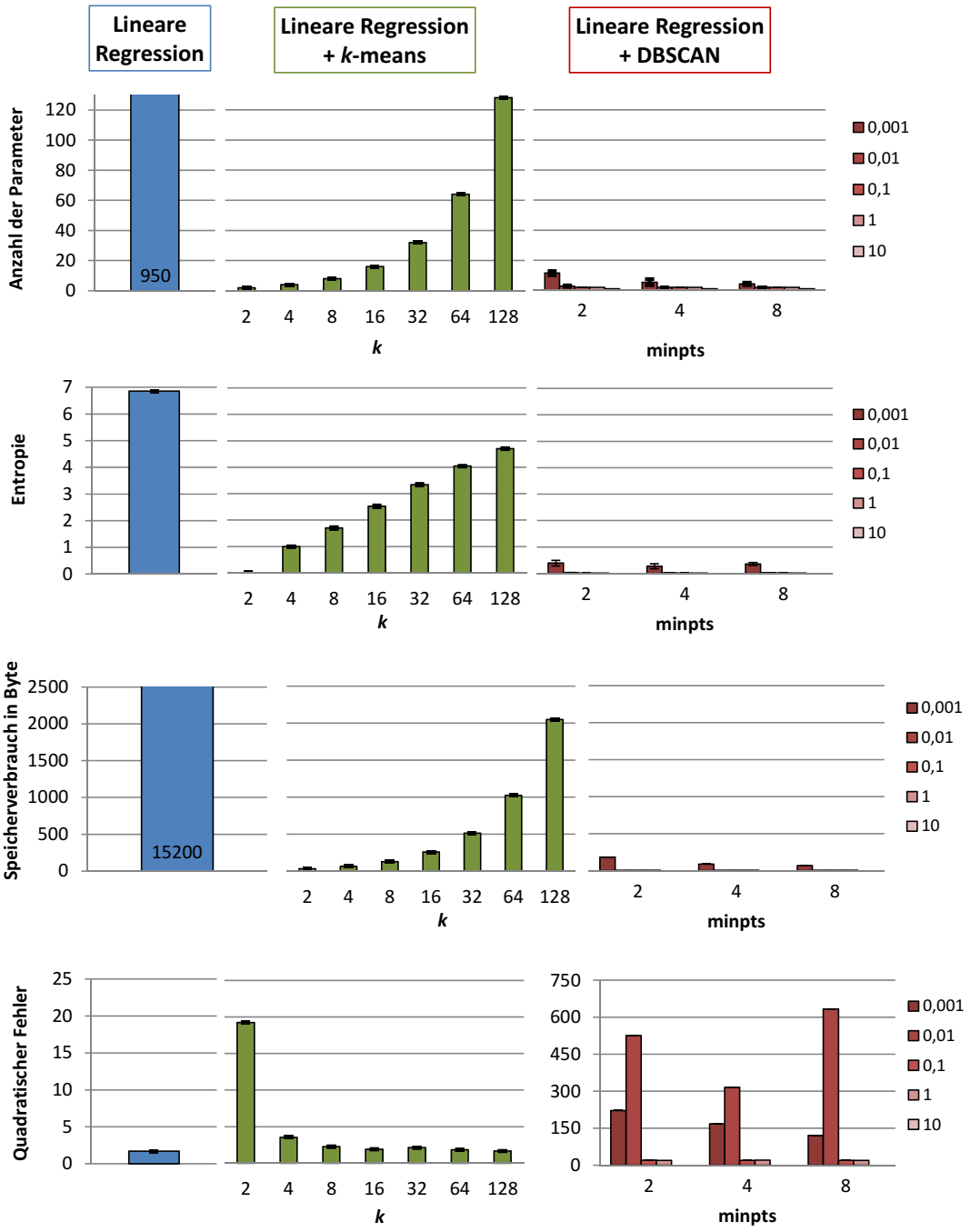

Abbildung 45: *Diagramme der linearen Regression für die restlichen Insight Daten.*

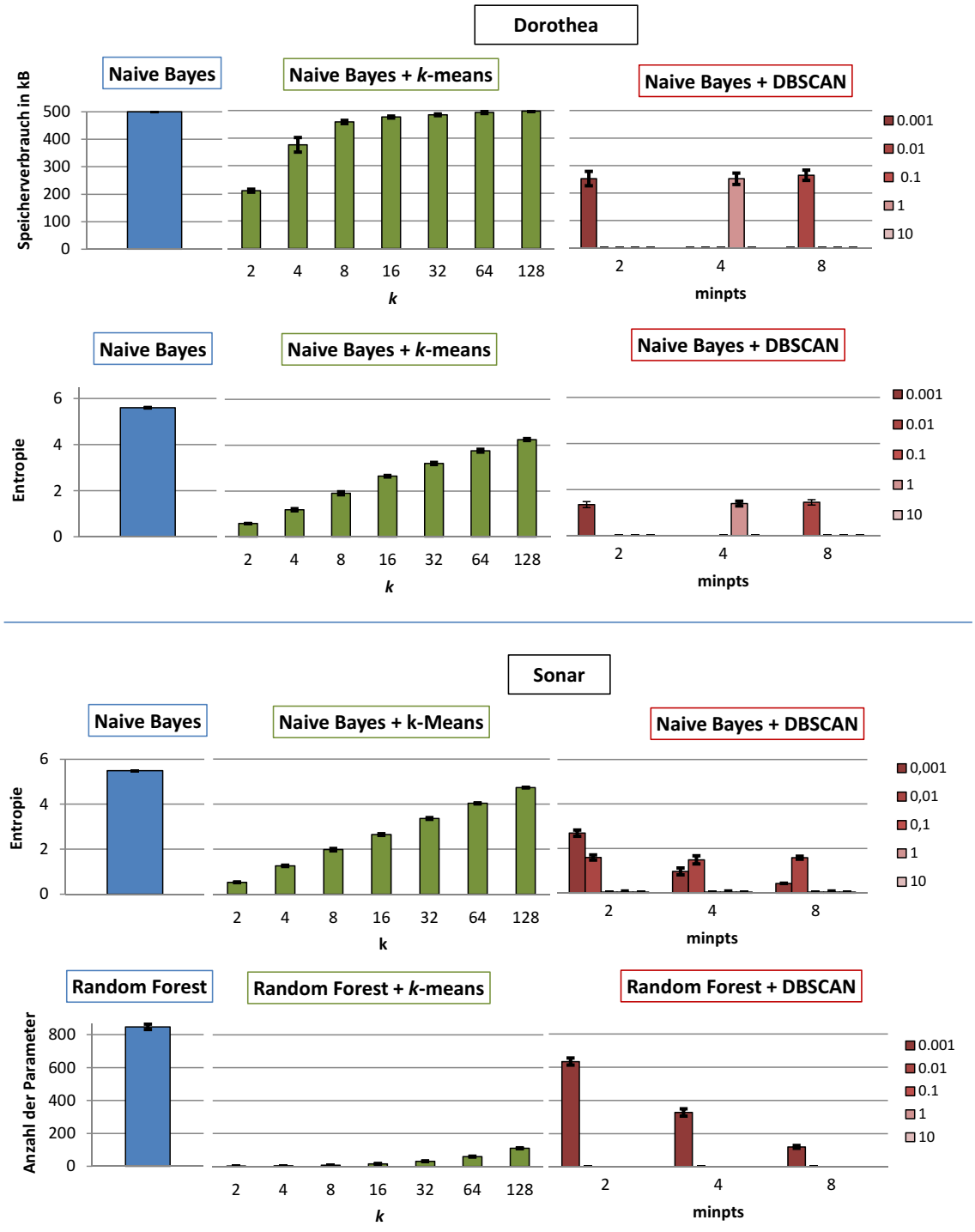

Abbildung 46: *Zusätzliche Diagramme für die Datensätze Dorothea und Sonar*

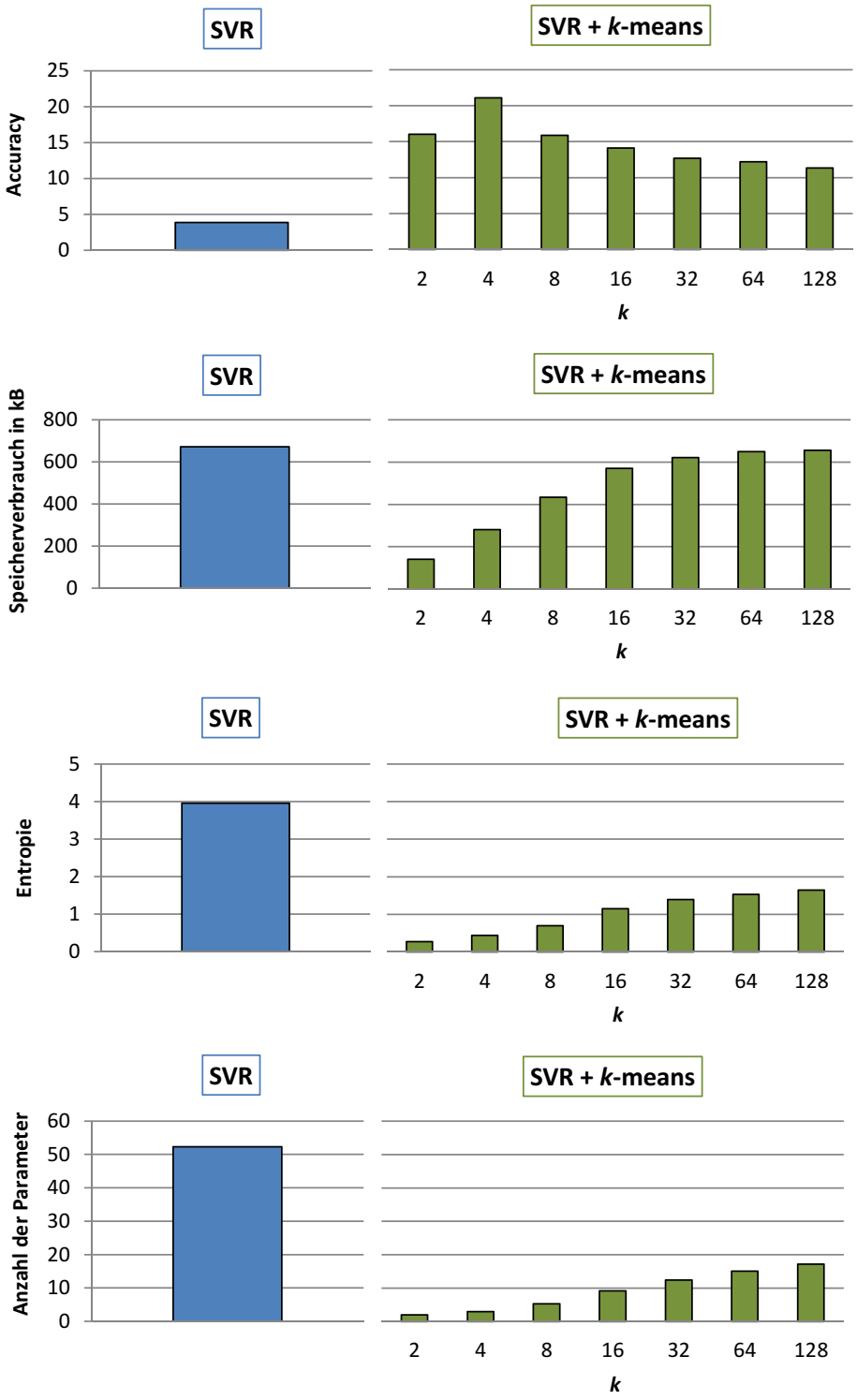

Abbildung 47: *Zusätzliche Diagramme für die SVR für den zweiten Datensatz von Insight.*

C CD-ROM