Online-Data-Mining auf Datenströmen: Methoden zur Clusteranalyse und Klassifikation

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik

der Otto-von-Guericke-Universität Magdeburg

von:	Diplom-Informatiker Jürgen Beringer
geboren am	09.01.1977 in Schlüchtern

Gutachterinnen/Gutachter:

Prof. Dr. Eyke Hüllermeier
Prof. Dr. Bernhard Seeger
Prof. Dr. Myra Spiliopoulou

Ort und Datum des Promotionskolloquiums: Magdeburg, den 24. August 2007

Jürgen Beringer
Online-Data-Mining auf Datenströmen:
Methoden zur Clusteranalyse und Klassifikation
Dissertation, Otto-von-Guericke-Universität
Magdeburg, 2007

Zusammenfassung

Die vorliegende Dissertation beschäftigt sich mit dem Thema „Data-Mining auf Datenströmen“, einem aktuellen Forschungsgebiet, dem in den letzten Jahren viel Aufmerksamkeit zuteil wurde. Durch neue technische Entwicklungen, wie Sensoren und Minicomputern, werden immer größere Datenmengen zeitnah erzeugt und erfasst. Um eine möglichst ebenso zeitnahe und effektive Verarbeitung solcher Daten zu ermöglichen, werden Online-Verfahren benötigt, die derartige Datenströme ohne externe Zwischenspeicherung verarbeiten.

In dieser Arbeit werden zunächst die Rahmenbedingungen und einige allgemeine Ansätze zur Verarbeitung von Datenströmen vorgestellt, sowie grundsätzliche Anforderungen für Data-Mining-Verfahren auf Datenströmen erläutert. Außerdem werden ausgewählte Problemstellungen für Data-Mining auf Datenströmen vorgestellt und bereits existierende Verfahren skizziert und diskutiert. Zwei allgemeine Aufgabenstellungen aus dem Bereich des unüberwachten und überwachten Lernens werden anschließend genauer untersucht. Zur Lösung dieser Aufgaben werden neue Verfahren entwickelt und evaluiert.

Das erste Verfahren beschäftigt sich mit einem neuartigen Szenario aus dem Bereich des Clustering, das in dieser Form noch nicht in der Literatur betrachtet wurde. Das Besondere hierbei ist, dass die Datenströme selbst als Objekte betrachtet werden, die gruppiert werden sollen. Diese Objekte verändern sich im Laufe der Zeit, wodurch sich auch die Clusterstruktur der Datenströme dynamisch verändert und damit adaptiert werden muss. Um ein effizientes Clustern von Datenströmen zu ermöglichen, müssen in diesem Kontext zahlreiche Teilprobleme gelöst werden; beispielhaft genannt seien hier die Adaption der Clusteranzahl und die Kompression der Datenströme.

Die zweite Problemstellung beschäftigt sich mit der Klassifikation von Datenströmen. Hierbei wird sowohl ein neues instanzbasiertes als auch ein regelbasiertes Verfahren vorgestellt. Die besondere Schwierigkeit besteht darin, Veränderungen im zu lernenden Konzept effizient zu erkennen und das gelernte Modell entsprechend anzupassen. Instanzbasierte Ansätze bieten bei der Aktualisierung des Modells eine höhere Effizienz und Flexibilität als regelbasierte Verfahren. Letztere sind dafür effizienter bei der Klassifikation von neuen Daten, weshalb es von der konkreten Anwendung abhängt, welcher Ansatz geeigneter ist. Die Evaluation der entwickelten Verfahren zeigt, dass sie erfolgreich und effizient ihre Aufgaben in unterschiedlichen Umgebungen erfüllen. Die Diskussion einiger Variationen spricht darüber hinaus für deren Flexibilität und Erweiterbarkeit.

Teile dieser Arbeit wurden publiziert in:

- J. Beringer and E. Hüllermeier.
  Efficient instance based learning on data streams.
  Intelligent Data Analysis, 2007 (to appear).

- J. Beringer and E. Hüllermeier.
  Adaptive optimization of the number of clusters in fuzzy clustering.
  In Proceedings FUZZ-IEEE 2007, London, UK, 2007 (to appear).

- J. Beringer and E. Hüllermeier.
  Fuzzy clustering of parallel data streams.
  In Jose Valente de Oliveira and Witold Pedrycz, editors,
  Advances in Fuzzy Clustering and Its Applications,
  pages 333-352. John Wiley and Sons, 2007.

- J. Beringer and E. Hüllermeier.
  Adaptive optimization of the number of clusters in fuzzy clustering.
  Proceedings 16th Workshop Computational Intelligence, pages 140-149, Dortmund, Germany, 2006.

- J. Beringer and E. Hüllermeier.
  Online clustering of parallel data streams.
  Data Knowledge Engineering, 58(2):180-204, 2006.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Zeitalter von Mobilität und Dynamik, Globalisierung und Vernetzung bestimmen Veränderung und Anpassung viele Alltagsbereiche. Menschen, Verkehrsmittel, Güter und Waren verändern ständig ihren Standort, Objekte und Systeme verschiedenster Arten unterliegen einer regelmäßigen und kontinuierlichen Veränderung. Dies betrifft fast alle Bereiche, egal ob es sich um ein menschliches Gehirn, Aktienkurse, einen Dieselmotor, Straßenverkehr, seismologische Aktivitäten, Keksproduktion, Klima oder die Natur handelt. Die Folge ist ein extrem großer zusätzlicher Aufwand für Überwachung, Kontrolle, Analyse und Steuerung, den man gerne weiter automatisieren würde. Es entstehen in kürzester Zeit Unmengen Daten, die aufgenommen, verwaltet und verarbeitet werden müssen. Einfache Aufgaben können hierbei meist mit relativ einfachen Methoden verwaltet und umgesetzt werden. Eine Analyse der Daten hingegen ist dagegen nicht auf einfache Weise umsetzbar, da normalerweise ausreichend Zeit und Ressourcen zur Verfügung stehen müssen. Messungen, Aufzeichnungen und Daten sind allerdings meist schon kurze Zeit nach ihrer Erfassung nicht mehr aktuell. Eine traditionelle Analyse auf statische Daten kann oftmals nur noch veraltetes und damit unbrauchbares Wissen extrahieren, wenn die enormen Daten überhaupt bewältigt werden können. Es ist notwendig, dass sich die zu erfassenden Daten ständig neu ermittelt werden und aktualisieren, um immer „up-to-date“ zu sein. Daraus ergeben sich Folgen von Daten und scheinbar unaufhörlich kommen neue, aktuellere Daten hinzu und man erhält *Datenströme*. Es bleibt keine Zeit, die Daten zunächst zu sammeln und zu speichern, um sie anschließend in aller Ruhe auszuwerten. Man will jederzeit eine aktuelle Analyse und aktuelles noch verwendbares Wissen, mit dem man Entscheidungen treffen und sinnvoll Einfluss in das aktuelle System nehmen kann. Die gewonnen Erkenntnisse und Resultate müssen sich ständig den neuen Daten anpassen. Dies kann mit existierenden Methoden des traditionellen statischen Data Mining nicht ausreichend oder gar nicht bewerkstelligt werden. Die Daten müssen direkt, wenn sie erfasst sind, ohne zu speichern weiter verarbeitet werden. Diesem Thema widmet sich ein relativ junger Forschungsbereich: *Data Mining auf Datenströmen*. Ziel ist es, immer aktuelles Wissen anbieten zu können und jederzeit in der Lage zu sein, neue Daten zu integrieren, daraus zu lernen und damit das extrahierte Wissen schnell und möglichst gut zu adaptieren. Unterschiedlichste Arten von Data Mining sind dabei betroffen und gefragt. Es gilt, Daten zu clustern, Entwicklungen zu erkennen und zu vergleichen, Modelle über den datengenerierenden Prozess zu lernen, Statistiken unterschiedlichster Art zu erheben, Zusammenhänge und Veränderungen zu erkennen und dies möglichst immer aktuell und genau, um zeitnah handeln zu können.

1.1 Ziel und Inhalt der Arbeit

Bei den meisten bisher entwickelten Ansätzen wurden nicht das allgemeine flexible Datenstrom-Modell als Grundlage genommen. Die Verfahren basieren auf sehr speziellen Problemstellungen, deren Datenströme spezielle Eigenschaften haben. Das Ergebnis sind

starre Verfahren, die sich nur bedingt auf andersartige Datenströme anwenden lassen. So muss in einer ähnlichen Umgebung nur die Geschwindigkeit, das zugrunde liegende Modell, die Größe der Objekte oder ein anderer der zahlreichen Faktoren verändert sein, und ein Verfahren ist nicht mehr sinnvoll anwendbar.

Die Folge wären zahlreiche spezielle Verfahren für Datenströme unterschiedlichster Art. Jedesmal wenn neue Datenstrom Szenarien auftreten, bedarf es einer neuen angepassten Variante um ein eigentlich bekanntes Problem zu lösen. Die Folge ist, dass Data Mining auf Datenströme immer ein sehr komplexe unüberschaubare Menge spezieller Algorithmen ist für unterschiedlichste Problemstellungen und Datenströme. Damit dieses eigenständige Gebiet interessant bleibt und ihre Akzeptanz behält, ist es notwendig, Verfahren für allgemeine Problemstellungen und nicht für spezielle Einzelfälle mit Sonderbedingungen zu entwickeln.

Ein wichtiges Ziel dieser Arbeit ist es, die entwickelten Verfahren möglichst allgemein zu halten. Dies geschieht dadurch, dass sie zum einen sehr flexibel und erweiterbar entwickelt und implementiert werden und zum anderen auf eine breite Palette von Problemstellungen anwendbar sein. Um die Erweiterbarkeit darzustellen, werden zu jedem Verfahren einige sinnvolle Varianten vorgestellt. Die Verfahren auf verschiedenartigen Datenströmen zu evaluieren, ist notwendig um eine breite Anwendbarkeit sicherzustellen. Insbesondere für allgemeine Probleme, zu denen es schon zahlreiche spezielle Lösungen gibt, ist eine derartige Evaluation sinnvoll.

Zunächst werden in dieser Arbeit der Bereich Data Mining auf Datenströmen und zugehörige Fragestellungen vorgestellt. Dazu gehört ein Überblick zu den bisherigen Veröffentlichungen sowie eine Diskussion zu den vorgestellten Lösungen. Probleme und Missverständnisse, die mit dem Thema verbunden sind, sollen hierbei angesprochen und, wenn möglich, geklärt werden. Desweiteren werden zwei allgemeine Aufgabenstellungen aus dem Bereich des unüberwachten und überwachten Lernens genauer untersucht und neu entwickelte Verfahren dazu vorgestellt und evaluiert.

Die erste Aufgabenstellung betrifft das Clustern von Datenströmen, wobei ein bisher unbehandeltes Setting betrachtet wird. Hierbei werden anstatt der Daten eines Datenstroms Datenströme direkt in Cluster eingeteilt. Da eine hohe Effizienz und Flexibilität benötigt wird, kommt man an den klassischen K-Means und Fuzzy C-Means Algorithmen kaum vorbei. Allerdings muss zunächst das Problem der über die Zeit flexiblen Clusteranzahl gelöst werden.

In der anderen Aufgabenstellung geht es um die Klassifikation auf Datenströmen. Das besondere Problem hierbei ist, dass sich durch Änderungen in den Daten auch das zu lernende Klassifikationsmodell ändern kann. Dies zu erkennen und anzupassen macht die Klassifikation auf Datenströmen deutlich aufwändiger als im statischen Setting. Zu diesem Bereich gibt es zwar schon mehrere Ansätze, allerdings sind sie meist durch die Entwicklung für spezielle Anwendungen oder Settings kaum vergleichbar oder gar universell

einsetzbar. Deshalb war das Ziel bei der Entwicklung der neuen Verfahren, möglichst einfache und flexible Ansätze zu erhalten, die in unterschiedlichsten Umgebungen verwendet werden können und mit unterschiedlichsten Daten und Datenveränderungen zurecht kommen. Die erste Variante beruht dabei aufgrund der Flexibilität auf einem instanzbasierten Ansatz. Die zweite Variante ist ein regelbasierter Ansatz. Das Problem geringer Flexibilität der meisten Regellerner für statische Daten wird durch eine Kombination mit einem instanzbasierten Verfahren behoben.

1.2 Gliederung

Im Anschluss an dieses Kapitel folgt eine Einleitung zu Data Mining, Datenströmen und Data Mining auf Datenströmen. Grundsätzliche Vorgehensweisen bei der Verarbeitung von Datenströmen und sinnvolle Anforderungen an Data Mining Verfahren für Datenströme werden vorgestellt. Im Kapitel 3 wird ein Überblick über bisher veröffentlichte Literatur im Bereich Datenströme und Data Mining auf Datenströmen vorgestellt, wobei auf einige Problemstellungen tiefer eingegangen wird. Kapitel 4 stellt dann ein Verfahren für eine neue Problemstellung aus dem Bereich unüberwachtes Lernen vor, wobei auch auf notwendige Zuarbeiten und deren Nutzen für die Clusteranalyse von statischen Daten eingegangen wird. Das Kapitel 5 beinhaltet einen neuen instanzbasierten Ansatz aus dem Bereich des adaptiven überwachten Lernens, der im daran anschließenden Kapitel zu einem regelbasierten Verfahren erweitert wird. Das letzte Kapitel beinhaltet eine ausführliche Zusammenfassung und gibt einen Ausblick für weitere Arbeiten.

Kapitel 2

Data-Mining und Datenströme

Durch die Entwicklungen im Bereich der Datenverarbeitung und Datenbank-Management-Systemen (DBMS) wurde es möglich, Daten automatisch zu verwalten und abzuspeichern. Die Folge war eine immer größere Sammlung von Daten, die eine manuelle Analyse immer schwieriger und aufwändiger machte. Die Entwicklung von automatischen Verfahren zur Analyse und Wissensentdeckung wurde notwendig. Seit dem wurden zahlreiche unterschiedliche Algorithmen für unterschiedliche Probleme entwickelt und auch in der Praxis angewendet, womit die Extraktion zahlreicher neuer Erkenntnisse ermöglicht wurde. Allerdings haben die meisten dieser Algorithmen gemein, dass die Daten schon komplett und persistent vorliegen müssen.

In den letzten Jahren werden nun auch immer mehr Sensoren eingesetzt, die ständig Daten sammeln und übermitteln. Die Entwicklungen von immer kleineren Sensoren und von Minicomputer ermöglichen immer weitere und komplexere Anwendungsgebiete, was zu einer ständig wachsenden Flut von gemessenen und erzeugten Daten führt. Nicht nur die Menge der Daten wächst dabei, sondern vor allem auch die Häufigkeit, in denen sie neu auftreten. Produktionsmaschinen, medizinische und technische Überwachung, Verkehr, Umwelt uvm. können ständig kontrolliert werden. Durch das permanente Ermitteln und Übertragen von Daten entstehen sogenannte Datenströme. Diese Daten enthalten wiederum zahlreiche Informationen, die man oft möglichst zeitnah weiter verwenden möchte. Durch das regelmäßige Auftreten neuer Daten und die ständige Veränderung der Umgebung können die bisherigen Verfahren für statische Daten allerdings nicht mehr direkt angewendet werden. Neue Algorithmen für die Gegebenheiten bei der Verarbeitung von Datenströmen wurden notwendig.

Dieses Kapitel gibt eine Einführung in die Bereiche Data Mining und Datenströme. Dabei wird zunächst jeder Bereich für sich vorgestellt bevor sie über die allgemeine Verarbeitung von Datenströmen in den Abschnitt zu Data Mining auf Datenströmen zusammengeführt werden.

2.1 Data-Mining

Ursprünglich stammt der Begriff „Data Mining“ aus dem Bereich der Statistik. Die heutige Verwendung ist allerdings nicht einheitlich, wobei häufig, wie auch hier, die Definition von Fayyad, Piatetsky-Shapiro und Smyth [FPSS96] benutzt wird. Danach versteht man unter Data Mining die (halb)automatische Suche nach Mustern in Daten.

Data Mining wird dabei als einzelner Schritt im „Knowledge Discovery in Databases“ Processing (KDD-Prozess) gesehen. Hierbei ist die allgemeine Vorgehensweise, dass zunächst Daten erfasst und gespeichert werden, z.B. in einer Datenbank oder einem Data-Warehouse. Im nächsten Schritt werden zunächst die für die Analyse interessanten Daten ausgewählt. Häufig ist zusätzlich noch eine Vorverarbeitung dieser Daten notwendig, be-

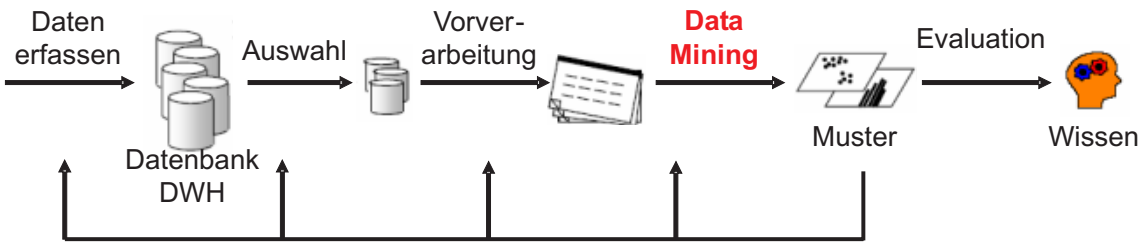

Abbildung 2.1: Zyklisches Prozessmodell des KDD-Prozesses

vor man sie mittels unterschiedlicher Data Mining Methoden analysieren und verarbeiten kann. Am Ende steht die Analyse und Weiterverarbeitung der Ergebnisse, um zu den gesuchten Erkenntnissen zu gelangen. Dies sieht nach einem linearen Ablauf aus, in der Praxis ist dies aber meistens anders. Nach jedem Schritt wird schon eine Analyse der bisherigen Ergebnisse durchgeführt, um eventuell den Ablauf zu ändern und die Analyse neu zu beginnen. Dies kann unter anderem bedeuten, dass bisher zuwenig Daten erfasst wurden, ungünstige Daten und Attribute ausgewählt wurden, die Vorverarbeitung verändert wernd muss oder gar die Anwendung einer anderen Data Mining Methode sinnvoller ist. Insgesamt erhält man einen zyklischen Ablauf der in Abbildung 2.1 nochmal dargestellt ist. Häufig wird hierbei auch von „Data-Mining Processing“ gesprochen.

In der Praxis tritt der Begriff Data Mining im Zusammenhang mit Data-Warehouse-Systemen und OLAP-Systemen (On-Line Analytical Processing) auf [CD97]. Auch hier ist eine eindeutige Abgrenzung nicht möglich. Ein Data-Warehouse dient zur Speicherung und Bereitstellung von großen Datenmengen zur Datenanalyse. OLAP Systeme dienen zur Entscheidungsunterstützung, wobei sie sich hauptsächlich auf graphische und textuelle Auswertungen konzentrieren. Daten aus mehreren Datenquellen werden in einem multidimensionalem Datenwürfel (OLAP-Cube) zusammengefasst. Durch Anwendung von vorgegebenen Operatoren (Aggregation, Verdichtung, Visualisierung usw.) können diese mit Hilfe von Tabellen und Graphiken dargestellt werden. OLAP Systeme beinhalten dabei aber nur sehr begrenzte und standardisierte Data Mining Verfahren. Die Anwendung verschiedenster Data Mining Techniken geht in der Regel weit über das, was OLAP Systeme anbieten, hinaus [HK00].

Data Mining Fragestellungen lassen sich in zwei grobe Bereiche einteilen: Vorhersage und Beschreibung. Allerdings ist die Grenze zwischen beiden Bereichen nicht klar gezogen. Fayyad, Piatetsky-Shapiro und Smyth [FPSS96] teilen Data Mining Probleme in die folgende sieben Typen auf:

- Klassifikation: Die Zuordnung von Daten zu verschiedenen Klassen soll gelernt werden. Dazu dienen markierte Beispiele für das Trainieren eines Klassifizierers.
- Regression: Im Gegensatz zur Klassifikation wird hier jedem Datensatz ein numerischer Wert zugeordnet. In der Regression soll eine Funktion anhand von Trainings-

daten gelernt werden, die aus einem Objekt die zugehörige Zahl berechnet.

- Clustering (Gruppierung): Daten sollen in Cluster eingeteilt werden, wobei im Gegensatz zur Klassifikation die Cluster noch völlig unbekannt sind und somit auch keine Trainingsdaten existieren. Hierbei sollen Daten innerhalb eines Clusters in einem gewissen Sinne ähnlich und Daten unterschiedlicher Cluster verschieden sein.

- (Wahrscheinlichkeits-)Dichte Schätzer (verwandt mit Clustering)

- Zusammenfassung (Summarization): Die Daten sollen kompakter dargestellt werden. Dies kann die Zusammenfassung einer Menge von Daten durch eine kompakte Darstellung sein (z.B. durch Mittelwert und Standardabweichung) oder die Suche nach Zusammenhängen zwischen den einzelnen Attributen zur Dimensionsreduktion.

- Abhängigkeitsanalyse (Dependency modeling): Hierbei gilt es, statistisch auffällige Zusammenhänge zwischen unterschiedlichen Attributen oder Attributmengen zu finden (z.B. Assoziationsregeln).

- Change and deviation detection: Hierunter versteht man die Erkennung von Änderungen innerhalb der Daten an unterschiedlichen Zeitpunkten bzw. über mehrere Zeitpunkte hinweg, wie sie z.B. in Zeitreihen existieren.

2.2 Datenströme

Ein Datenstrom kann sehr große Mengen von Daten liefern. Ein Multi-Sensornetzwerk mit 10000 Sensoren, wobei jeder Sensor jede Sekunde einen Messwert liefert, wäre ein Beispiel dafür. Im Bereich der Datenspeicherung, Verwaltung und Verarbeitung ergeben sich hieraus neue Probleme, die es zu lösen gilt.

Im Datenstrom-Modell geht man davon aus, dass Daten nicht jederzeit frei und beliebig verfügbar sind wie z.B. in einer Datenbank. Die Daten erscheinen in Form von aktiven Datenströmen. Das Datenstrom-Modell unterscheidet sich vom traditionellen relationalen Datenmodell in den folgenden Punkten [BBD+02]

- Die Daten treffen kontinuierlich in aktiver Art und Weise ein. Dies bedeutet, die einzelnen Elemente der Datenströme werden geliefert, wenn sie erzeugt werden und nicht, wenn sie benötigt werden, also nicht auf Anfrage.

- Die Reihenfolge, in der die Daten geliefert werden, ist beliebig und kann nicht durch das System bestimmt werden.

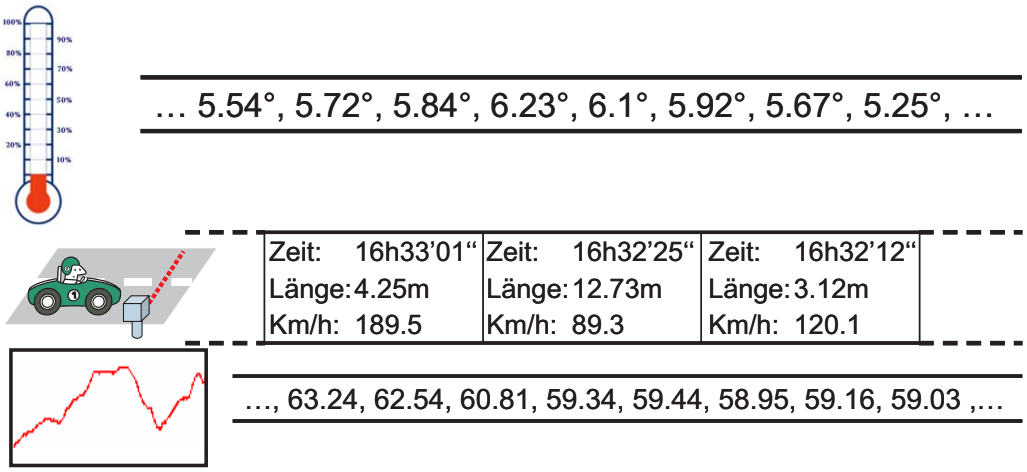

Abbildung 2.2: Beispiele für Datenströme

- Datenströme sind potentiell unendlich lang.
- Es ist nicht möglich, alle Daten zu speichern, somit ist kein freier Zugriff auf die bisherigen Daten möglich. Es ist nur möglich, ein kleine, begrenzte Auswahl der Daten oder eine Zusammenfassung (Synopse) zu speichern.
- Durch den begrenzten Speicher und die begrenzte Verarbeitungszeit sind vollständige und exakte Ergebnisse oft nicht möglich. Meist können nur approximative Ergebnisse berechnet werden.

Beispiele für Datenströme können Sensoren aller Art sein wie Temperaturmessungen, Luftfeuchtigkeit, Windgeschwindigkeit, seismische Wellen, Geschwindigkeitsmessungen, Körpersignale in der Medizin u.v.m. Dabei kann man zwischen periodisch eintreffenden Daten wie die Messung der Temperatur jede Sekunde und Ereignis gesteuerten Daten wie bei der Geschwindigkeitsmessung von Autos unterscheiden. In diesen Fällen werden immer einfache reelle Zahlen gemessen und geliefert. Ein Datenstrom kann allerdings auch aus komplexeren Objekten bestehen. Ein Sensor kann mehrere Dinge gleichzeitig messen bzw. mehrere Sensoren werden zusammengefasst, weil sie unterschiedliche Eigenschaften eines Objektes beschreiben (z.B. bei Autos: Geschwindigkeit, Länge, Kennzeichen, Farbe,...). Einige anschauliche Beispiele zeigt Abbildung 2.2.

Datenströme treten desweiteren bei Vebindungsdaten im Bereich Telekommunikation, Kreditkarten Transaktionen, Netzwerküberwachung und Traffic, Aktienkursen und Weblogs auf. Extrem große Datenmengen ("massive data"), bei denen wahlfreier Zugriff durch Art der Speicherung extrem teuer wäre, werden mittels Datenströmen verarbeitet. Hierbei wären zwar auch mehrere Zugriffe auf die Daten möglich, aber aus Effizienzgründen sollten Algorithmen nur einen Durchlauf benötigen. Natürlich existieren Datenströme

auch in vielen anderen Bereichen, aber diese Aufzählung soll zumindest zeigen, dass Datenströme und deren Verarbeitung in sehr vielen Bereichen von Belang sind.

2.2.1 Abgrenzung zu benachbarten Gebieten

Beim Thema Datenströme und Data Mining auf Datenströmen gibt es leider häufig Missverständnisse, vornehmlich auch in der Abgrenzung zu anderen Forschungsgebieten. Dieser Abschnitt soll klar stellen, was zum Forschungsgebiet Datenströme und Data Mining auf Datenströmen gehört und insbesondere welche mehr oder weniger verwandten Gebiete nicht dazu gehören bzw. wo es fließende Übergänge gibt.

Die am häufigsten auftretenden Missverständnisse gibt es im Zusammenhang mit dem Gebiet der Zeitreihenanalyse (Time Series Analysis), das oftmals mit der Analyse von Datenströmen gleichgesetzt wird. Auf den ersten Blick scheinen sich Zeitreihen und Datenströme auch sehr zu ähneln. Zeitreihen sind Daten, die ebenfalls einen zeitlichen Bezug haben, was normalerweise durch ein zeitliches Attribut gegeben ist. Zeitreihen sind in der Regel allerdings aufgezeichnete und abgespeicherte Daten. Es sind also persistente Daten, die meist in normalen Datenbanken oder einem Datawarehouse abgespeichert wurden. Sie unterscheiden sich von normalen Datensätzen des Data Mining nur darin, dass sie eine zeitliche Komponente haben, die eine zentrale Rolle bei der Analyse spielt. Im Gegensatz zur Analyse von Datenströmen hat man aber freien Zugriff auf die Daten oder kann die Daten zumindest beliebig oft durchlaufen. Man weiß zu Beginn der Analyse schon, um wie viele Datensätze es sich handelt, wie groß die Daten sind und ist komplett unabhängig von Datenraten und damit verbundenen kurzfristigen Zeitbeschränkungen. Mittlerweile gibt es in der Zeitreihenanalyse auch Bestrebungen, Online-Analyse zu betreiben, wodurch es zu Überschneidungen zum Data Mining auf Datenströmen kommt.

Viele irrtümlich als Datenstrom-Algorithmen benannten Verfahren missachten die Eigenschaft, dass die Daten nicht komplett abgespeichert werden können und damit auch kein weiterer Durchlauf möglich ist [MM02, Hid99]. Manche Ansätze stellen während der Lernphasen keine brauchbaren Ergebnisse zur Verfügung, sondern sehen ein Ergebnis erst für das Ende vor, womit die Eigenschaft unberücksichtigt bleibt, dass Datenströme potentiell unendlich lang sein können und somit potentiell niemals ein Ergebnis geliefert wird. Es treten sogar sogenannte Datenstrom-Verfahren auf, die die Anzahl der Daten schon während der Lernphase kennen [GMMO00]. Das relativ häufige Auftreten von Veröffentlichungen, in denen eine eigene meist nur intuitive Definition von Datenströmen verwendet wird, führt zu häufigen Missverständnissen.

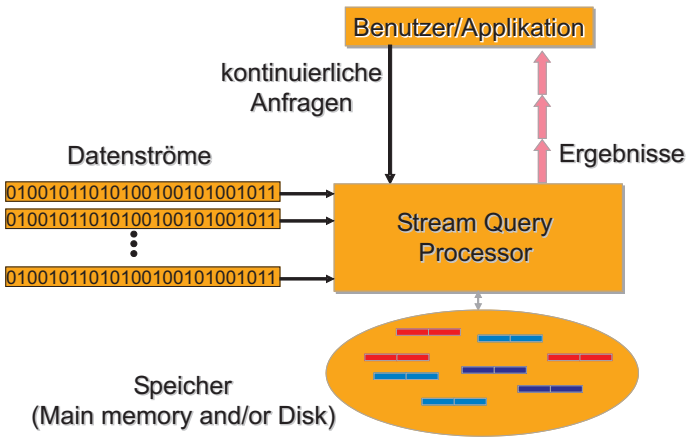

Abbildung 2.3: Datenstromanfrage Verarbeitung

2.3 Datenstrom-Management-Systeme (DSMS)

Datenströme dienen unter anderem als Datenquelle für Anfragen. Normalerweise sind diese Anfragen, wie die Datenströme selbst, kontinuierlich und bilden wiederum einen Datenstrom von Ergebnissen, das regelmäßig durch neue Eingaben aktualisiert wird. Wie in Abbildung 2.3 dargestellt, rufen Benutzer oder Anwendungen einen Verarbeitungsprozess für derartige Anfragen auf. Bei der Verarbeitung hat der Prozess die Möglichkeit, eine ihm zugewiesene Speichermenge zu verwenden, wobei in der Regel hauptsächlich Hauptspeicher verwendet wird. Hier können einzelne Daten, Zwischenergebnisse oder andere hilfreiche Zusammenfassungen oder Statistiken (sogenannte Synopsen) schnell abgespeichert und wieder abgerufen werden. Die Nutzung von Sekundärspeicher macht in der Regel nur bei Verfahren Sinn, bei denen die deutlich längeren Zugriffszeiten keine zeitkritischen Auswirkungen haben. Allerdings ist auch hier meist die Größe des verwendbaren Sekundärspeichers wegen der hohen Zugriffszeit durch die maximal zur Verfügung stehenden Verarbeitungszeit begrenzt.

Zur Verwaltung von Datenbanken und Anfragen auf den gespeicherten Daten existieren Datenbanksysteme (DBS), die aus der Datenbank (DB) und dem Datenbank Management System (DBMS) bestehen. Das Datenbank-Management-System kontrolliert alle Zugriffe und Operationen auf die Daten, überprüft Konsistenz, verwaltet Benutzer, kümmert sich um Datensicherheit, Wiederherstellung bei Verlust, Transaktionen aller Art uvm. Entsprechend dazu wurden Datenstrom-Management-Systeme (DSMS) für Datenströme in den letzten Jahren entwickelt.

Ein DSMS verwaltet ihm gestellte Anfragen, führt diese aus und liefert die Ergebnisse wie gewünscht, auf der Basis ihm zur Verfügung gestellten Datenstromquellen aus. Die Interaktionsmöglichkeiten für Benutzer und Anwendungen sind dabei zahlreich.

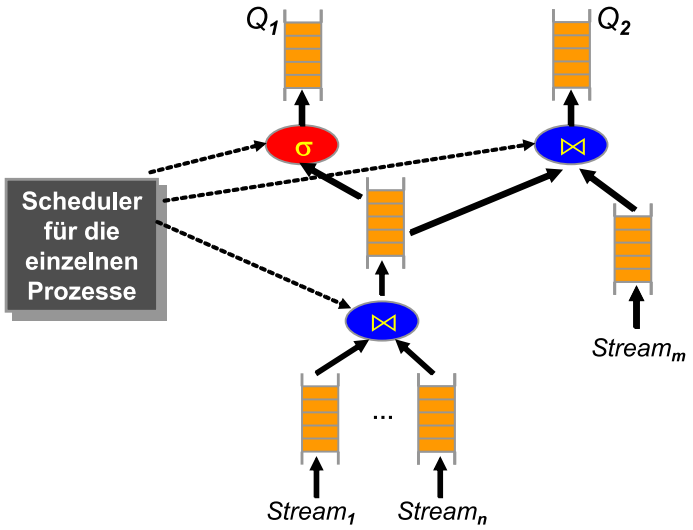

Abbildung 2.4: Beispiel eines Anfrageplans

Die wichtigsten Möglichkeiten der Anwender sind:

- Hinzufügen und Löschen von Datenstromquellen
- Hinzufügen und Löschen von kontinuierlichen Anfragen
- An- und Abmelden (subscribe und unsubscribe) von existierenden Anfragen

Anfragen auf Datenströmen sind immer in Echtzeit bzw. zeitnah zu beantworten und das solange, bis die Datenströme nicht mehr vorhanden sind oder die Anfrage gestoppt wird. Natürlich können auch mehrere Anfragen gleichzeitig auf den selben Datenströmen gestellt werden. Die Aufgabe des DSMS ist hierbei eine effiziente Ausführung der Anfragen durch Erkennung und Ausnutzung von identischen Teilanfragen, durch eine dynamische Optimierung aller Anfragen gemeinsam und durch die Sicherstellung der zeitnahen Beantwortung aller Anfragen gleichzeitig. Dazu ist die zeitliche Steuerung der einzelnen Operatoren sehr entscheidend. Ständig laufende Prozesse sind der dynamische Anfrageoptimierer, Speichermanager und der Scheduler. Während der Speichermanager den zur Verfügung stehenden Speicher auf die einzelnen Operatoren verteilt und optimiert, regelt der Scheduler die Ausführungszeit, die jedem Operator zur Verfügung steht, um möglichst effizient und früh Ergebnisse liefern zu können. Ein Beispiel eines einfachen Ausführungsplans ist in Abbildung 2.4 zu sehen.

Häufig benötigen DSMS auch persistente Daten aus einer Datenbank, die dann in das System integriert werden müssen. Anfragen können jederzeit hinzugefügt und entfernt werden. Benutzer können sich jederzeit für unterschiedliche Anfragen an- und abmelden. Falls kein Benutzer mehr für eine Anfrage angemeldet ist, muss sie zwar nicht mehr

berechnet werden, wenn sich aber wieder jemand anmeldet, müssen aktuelle Ergebnisse zeitnah wieder geliefert werden können. Nebenbei müssen natürlich auch Benutzersichten und Zugriffssicherheit verwaltet und kontrolliert werden. Die Aufgaben eines DSMS sind also deutlich komplexer und bedürfen einer erhöhten Aufmerksamkeit auf das dynamische Zeit- und Speichermanagement, als die eines DBMS.

Die Ansprüche, die ein allgemeines DSMS an die Datenströme stellen, sind meist sehr gering. Die einzelnen Daten können fast ein beliebiges Format haben. Doch je mehr Informationen zur Verfügung stehen, desto mehr Möglichkeiten können für Optimierungen genutzt werden. Was für vieler Algorithmen unvermeidbar ist, ist ein Zeitstempel, den jedes Objekt in der Regel besitzen sollte und der auch meist verlangt wird. Viele Anfragen beinhalten zeitliche Komponenten in Form von Ordnung oder Zeitfenstern. Der Zeitstempel gibt eine Ordnung für die Objekte an und gerade wenn mehrere Ströme verarbeitet werden, müssen die Elemente häufig auch zeitlich vergleichbar sein. Im Notfall wird ein Zeitstempel beim Eintritt des Objektes in das System generiert und angehängt. Für einfache Verfahren ist der Zeitstempel meist nur ein Zähler, der die Elemente des Stroms durchnummeriert, aber in der Praxis wird normalerweise eine absolute Zeit oder ein Zeitintervall gefordert. Der Zeitstempel repräsentiert den Zeitpunkt, in dem das Element auftrat oder gemessen wurde. Ein Zeitintervall spiegelt dann häufig einen Gültigkeitszeitraum für das Objekt wieder. Deshalb sollte auch jeder Datenstromoperator die Ergebnisse mit einem Zeitstempel versehen um eine sinnvolle Weiterverarbeitung zu ermöglichen.

Einige bisherige Projekte zu Datenstrom-Management-Systemen sind:

- STREAM (Stanford): Entwicklung eines DSMS [ABB+03]
- Cougar (Cornell): Sensoren [YG02]
- Aurora (Brown/MIT): Monitoring, Datenfluss [ACC+03]
- Hancock (AT&T): Telekomunikation [CFPR00]
- Niagara (OGI/Wisconsin): Internet XML Datenbanken [CDTW00]
- OpenCQ/WebCQ (Georgie Tech): Trigger, Inkrementelle Sichten [LPT00]
- Telegraph (Berkeley): Adaptive Umgebung für Sensoren [CCD+03]
- Tribeca (Bellcore): Netzwerk Monitoring [Sul96]
- Streaminer & MAIDS (UIUC & NCSA): Data Mining auf Datenströmen [CCP+04]
- PIPES (Marburg, Deutschland): generische Infrastruktur zur Datenstromverarbeitung [KS04]

2.4 Data-Mining auf Datenströmen

Klassisches Data-Mining analysiert persistent gespeicherte Daten. Durch wiederholtes Anwenden der Verfahren mit jeweils neu angepassten Parametern lassen sich die Ergebnisse optimieren. Anschließend können die gewonnen Erkenntnisse genutzt werden. Über die Zeit haben sich aber viele Szenarien ergeben, die flexiblere Data-Mining Verfahren benötigen. Schlagwörter, die sich für flexiblere Eigenschaften gebildet haben und im Umfeld von Datenströme eine wichtige Rolle spielen, sind:

- **Inkrementell:** Inkrementelle Verfahren verarbeiten die einzelnen Daten nacheinander und passen ihr Modell jeweils neu an. Derartige Algorithmen können damit jederzeit neue Daten effizient integrieren und lernen.
- **Anytime:** Anytime Systeme sind dadurch gekennzeichnet, dass sie während der Verarbeitung jederzeit in der Lage sind, schon nutzbare Zwischenergebnisse zu liefern. Dies ermöglicht dem Nutzer, das Verfahren abzubrechen, wenn die Ergebnisse schon ausreichend gute Qualität haben, oder die Entwicklung der Ergebnisse über die Zeit zu beobachten. Inkrementelle Algorithmen sind meist auch anytime, damit man die Zwischenergebnisse bis zum Input des nächsten Elements nutzen kann.
- **Interaktiv:** Interaktive Algorithmen ermöglichen es dem Benutzer jederzeit, Einfluss auf den Ablauf zu nehmen. Es besteht die Möglichkeit, noch während der Verarbeitung der Daten relevante Einstellungen zu ändern. Häufig sind diese Verfahren auch anytime, damit man die Zwischenergebnisse beobachten kann und anhand dieser dann sinnvoll Änderungen vornehmen kann.
- **Online:** Der Begriff Online ist der am wenigsten klar definierte Begriff. Er wird oft im Zusammenhang mit anytime und inkrementellen Verfahren verwendet, spielt aber auch bei interaktiven Anwendungen eine Rolle. Normalerweise tritt das Schlagwort „Online“ in vernetzten und interaktiven Umgebungen auf und steht für eine direkte zeitnahe Verbindung zwischen Generierung bzw. Auftreten der Daten und ihrer Verwendung.
- **Adaptiv:** Adaptive Verfahren besitzen die Möglichkeit, zeitliche Änderungen in den Daten zu erkennen und sich anzupassen. Sie machen hauptsächlich nur im Zusammenhang mit inkrementellen anytime Verfahren Sinn. Insbesondere kann bisher gelerntes Wissen veralten, so dass mit neuen Daten das Ergebnis nicht immer nur detaillierter und genauer wird, sondern es sich auch komplett ändern kann.

Da Datenströme permanent neue Daten liefern, macht eine Verarbeitung oder Analyse nur Sinn, wenn sie ebenso permanent läuft. Wie schon erwähnt, hat man meist auch gar nicht die Möglichkeit, die Daten abzuspeichern, und erst wenn keine neuen Daten

mehr auftreten, die Analyse durchzuführen. Eine Data-Mining Analyse hat häufig auch den Zweck, Prozesse zu kontrollieren und gegebenenfalls korrigierend einzugreifen. Dazu sind aber möglichst schnelle, zeitnahe und genaue Ergebnisse notwendig. Dadurch entstehen natürlich Anforderungen an Data-Mining Methoden, die für Datenströme verwendbar sein sollen. Domingos und Hulten listen folgende Anforderungen auf ([DH03]):

- **konstante Zeit:** Die Verarbeitung eines Elementes muss in konstanter Zeit in Abhängigkeit der bisherigen Anzahl an Elementen des Datenstroms stattfinden. Ansonsten kommt irgendwann der Zeitpunkt, an dem die Verarbeitung weiterer Elemente nicht mehr möglich ist.
- **konstanter Speicher:** Genauso darf ein Element nur eine konstante Menge Speicher verwenden. Auf der einen Seite ist der Speicher natürlich auch begrenzt, aber vor allem steht aus zeitlichen Gründen nur eine konstante Anzahl an Speicherzugriffen zur Verfügung. Hierbei gilt ebenso, dass der Speicherbedarf trotzdem wieder in Abhängigkeit von anderen Faktoren sehr interessant ist.
- **One Pass:** Datenströme müssen in einem Durchlauf verarbeitet werden. Diese Bedingung wird schon alleine durch die Grundeigenschaften von Datenströmen verlangt. Datenströme sind flüchtige Daten und liegen nicht persistent vor.
- **Anytime:** Der Algorithmus muss jederzeit ein zeitnahes Ergebnis zur Verfügung stellen, damit das Ergebnis noch Gültigkeit hat und verwendbar ist. Viele Datenstrom Verfahren verarbeiten zwar die Daten aus Effizienzgründen in Blöcken, diese sollten aber nie so groß werden, dass der dadurch entstehende zeitliche Verzug problematisch wird oder gar wichtige Ergebnisse übersprungen und damit nicht erkannt werden.
- **Quality of Service:** Das Ergebnis sollte möglichst ähnlich zu dem berechneten Ergebnis ohne die zeitlichen und speichertechnischen Einschränkungen sein. Oftmals lassen sich zwar durch die Einschränkungen bei der Verarbeitung von Datenströmen nur approximative Ergebnisse berechnen, trotzdem gilt es, immer eine möglichst hohe Qualität zu liefern.
- **Adaptiv:** Das Verfahren soll robust gegenüber Fehlern in den Daten (*Rauschen*) aber sensibel gegenüber Änderungen des datengenerierenden Prozesses (*Concept Drift*, siehe Abschnitte 3.4.1 und 5.1.3) sein. Falls Concept Drift vorhanden ist, muss sich das Modell möglichst schnell und effizient anpassen. Wissen, das nicht mehr gilt, muss gelöscht werden. Weiterhin gültiges Wissen darf hingegen nicht gelöscht werden, damit die Qualität der Ergebnisse sich nicht unnötig verschlechtert.

Für Data-Mining auf Datenströmen gibt es grundsätzlich zwei verschiedene Szenarien, die in Abbildung 2.5 dargestellt werden. Der am häufigsten verwendete Ansatz ist,

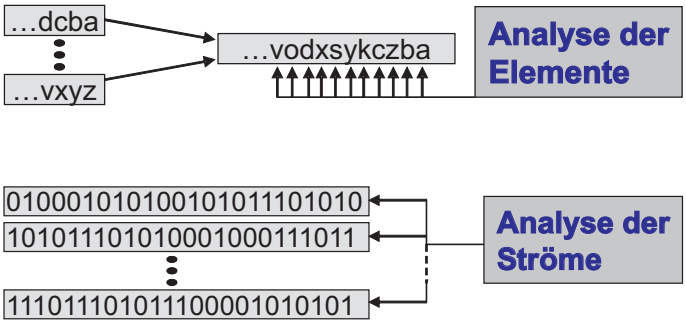

Abbildung 2.5: Data-Mining auf Datenstrom Szenarien

die einzelnen Elemente eines Datenstromes als Objekte zu betrachten und zu analysieren. Darüber hinaus gibt es auch noch die Möglichkeit, die Datenströme an sich als sich verändernde Objekte zu betrachten [BH06]. Beispiele hierfür sind die Kurse von Aktien. Jede Aktie wird dabei durch einen Datenstrom aus seinen Kurswerten beschrieben. Verschiedene Aktien lassen sich anhand ihres Verlaufs vergleichen und analysieren.

2.4.1 Adaptives Data-Mining

Die Anwendung eines adaptiven Data-Mining Verfahrens besteht aus der Initialisierungsphase und der Update-Query Phase. In der Initialisierungsphase erhält das Verfahren notwendige Parameter und häufig auch eine kleine Initialisierungsmenge von Elementen. Dies ist notwendig, damit von Anfang an sinnvoll Anfragen beantwortet werden können, was mit einem komplett leeren Modell nicht möglich ist. Meist ist diese Initialisierungsmenge nichts anderes als die Menge der letzten Daten des zu verarbeitenden Datenstroms. Die Update-Query Phase besteht aus Update und Query Aufrufen in beliebiger Reihenfolge. Jedesmal, wenn der zu verarbeitende Datenstrom ein neues Objekt liefert, wird die Update Funktion mit diesem Element aufgerufen. Query-Aufrufe sind dagegen jederzeit möglich. Häufig werden die Query-Aufrufe auch automatisiert, indem das Verfahren einfach nach jedem Update ein aktualisiertes Ergebnis liefert. Die Ergebnisse ergeben dann wiederum einen Datenstrom von meist komplexen Objekten. Bei Assoziationsregeln sind dies z.B. die Menge der aktuell gefundenen Regeln und beim Clustering die aktuelle Clusterstruktur. Komplizierter ist dies bei Klassifizierern, da das Ergebnis ein gelerntes Modell ist, mit dessen Hilfe neue Objekte klassifiziert werden können. Bei derartigen statusbehafteten Anfragen (Anfragen mit Parametern) und Verfahren mit komplexeren Query Funktionen, macht eine Beantwortung auf Anfrage sicherlich aber mehr Sinn als die Ausgabe der Ergebnisse als Datenstrom.

In der Vorgehensweise ähneln adaptive Algorithmen den inkrementellen anytime Verfahren. Beide Phasen laufen auch ähnlich ab, der eigentliche Unterschied liegt in der

Bedeutung der Update Funktion. Im inkrementellen Fall ist die Aufgabe, das bisherige Wissen mit den neuen Informationen zu verfeinern, im Gegensatz zum adaptiven Fall veraltet aber kein bisheriges Wissen. Ein adaptives Update muss nicht nur neues Wissen hinzufügen, sondern auch das bisherige Wissen auf Gültigkeit überprüfen und gegebenenfalls veraltetes Wissen wieder vergessen. Da sich in einigen Situationen der datengenerierende Prozess auch zyklisch entwickeln kann, macht es teilweise Sinn, veraltetes Wissen aufzuheben, um es bei wieder auftreten schnell reaktivieren zu können.

Ein adaptiver Data-Mining Algorithmus ist durch Funktionen für Initialisierung, Update und Query beschrieben. Meist sind allerdings Initialisierung und Query Funktionen trivial, so dass nur der Algorithmus für ein Update angegeben wird. Durch die adaptive Anpassung des Modells im Update Schritt unterscheiden sich die adaptive Verfahren von nicht adaptiven.

2.4.2 Standard Techniken für Datenströme

Im Umgang mit Datenströmen gibt es einige immer wieder benutzte Techniken. Bei diesen handelt es sich meist um eine Art Grundrezept, um mit Datenstrom spezifischen Problemen umzugehen. Im Folgenden wird dazu die Sliding Window Technik und die blockweise Verarbeitung genauer vorgestellt.

Sliding Window Unter einem Sliding Window versteht man ein Zeitfenster über die letzten Elemente eines Datenstroms. Die Größe dieses Fenster ist häufig in der absoluten Größe, manchmal aber auch durch ein zeitliches Intervall definiert. Die einfachste Variante eines Sliding Windows mit konstanter Größe entfernt jedesmal, wenn ein neues Element eingefügt wird, das älteste Element. Verfahren für statische Daten benötigen häufig freien Zugriff (random access) auf eine feste Datenmenge. Da dies nicht für allgemeine Datenströme gegeben ist, werden Sliding Windows benutzt, die komplett in den Hauptspeicher passen. Dadurch ist eine traditionelle Analyse für diese aktuelle Teilmenge des Datenstroms möglich. Häufig will man auch nur einen aktuellen Teil des Datenstroms analysieren, wofür sich Sliding Windows ebenso eignen. Wenn sich die Größe des zur Zeit interessanten Fensters ändert, benutzt man oftmals „Adaptive Sliding Windows“, also Sliding Windows mit sich adaptiv verändernden Größen.

Blockweise Verarbeitung Ein Update bei jedem neuen Datenelement durchzuführen, ist gerade bei Datenströmen mit hohen Datenraten zu aufwändig. Da also nicht in jedem Schritt ein Update möglich ist, wird erst dann, wenn eine feste Anzahl neuer Daten angekommen sind, ein Update mit diesem Block von Daten durchgeführt. Da die Analyse damit nicht in jedem Schritt aktualisiert wird, erhält man eine zeitliche Ungenauigkeit

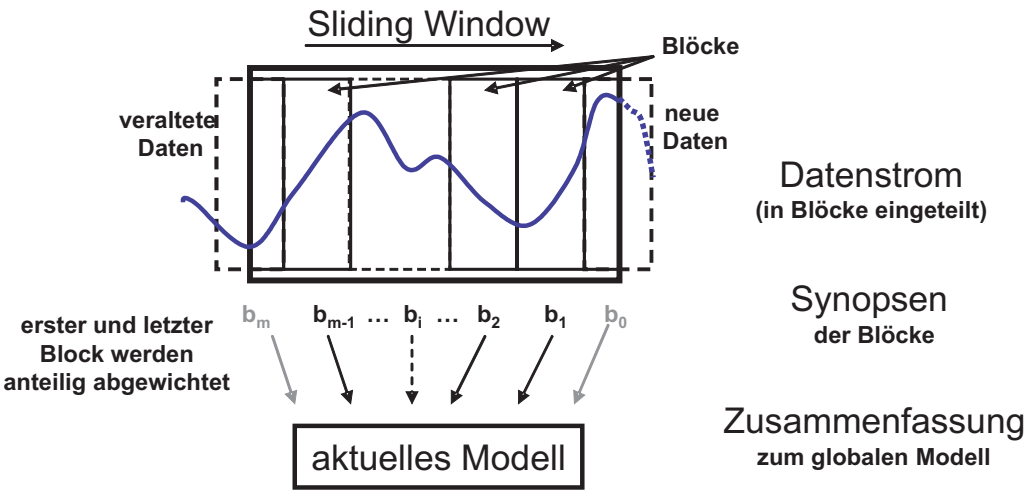

Abbildung 2.6: Blockweise Verarbeitung

bzw. Approximation, die aber oft akzeptierbar ist.

Eine weitere Variante der blockweisen Verarbeitung wird oftmals in Verbindung mit Sliding Windows benutzt. Um nicht bei jedem Update eine komplette Analyse auf dem gesamten Sliding Window auszuführen, wird dieses in Blöcke aufgeteilt. Für die einzelnen Blöcke können Vorverarbeitungen stattfinden und Zusammenfassungen berechnet werden, die, solange der Block im Fenster bleibt, auch wiederverwendet werden können. Bei jedem Update müssen dann nur noch die Ergebnisse der einzelnen Blöcke betrachtet, kombiniert und analysiert werden. Besonders ideal sind Verfahren, bei denen die eigentlichen Daten eines Blockes nicht mehr gespeichert werden müssen, um eine Analyse durchzuführen. Dadurch ist ein deutlich größeres Sliding Window möglich. Updates können hierbei blockweise stattfinden, d.h. jedesmal, wenn genügend neue Daten für einen Block vorhanden sind, wird ein Update ausgeführt und das Ergebnis aktualisiert. Oft sind aber auch Updates bei jedem neuen Datenelement möglich, wobei der aktuellste Block auch schon im nur teilweise gefüllten Zustand verarbeitet werden kann. Da die Daten allerdings meist nur blockweise aus dem Sliding Window gelöscht werden können, wird in diesen Fällen der älteste Block, der sich nur noch zum Teil im Sliding Window befindet, entsprechend diesem Anteil bei der Analyse abgewichtet. Ein Beispiel für die blockweise Verarbeitung ist in Abbildung 2.6 dargestellt.

2.5 Resümee

Neben einer kurzen Einführung zu Data-Mining wurden wichtige Aspekte und Charakteristika von Datenströmen dargestellt und Eigenschaften der Verarbeitung von Daten-

strömen diskutiert. Für die folgenden Kapitel wurden ebenso die Bedingungen für Data-Mining-Verfahren auf Datenströmen diskutiert sowie später zu verwendende Techniken wie Sliding Window und blockweise Verarbeitung angesprochen. Das nächste Kapitel gibt nun eine Übersicht über eine ausgewählte Menge von Fragestellungen für Datenströme, bevor die neu entwickelten Verfahren vorgestellt werden.

Kapitel 3

Data-Mining auf Datenströmen: Überblick

Um Applikationen und bisher entwickelte Data-Mining-Verfahren auf Datenströmen geht es in diesem Kapitel. Auch wenn sich das Thema Datenströme erst in den letzten 5 bis 10 Jahren im Bereich der Forschung verbreitet hat, gab es einzelne Ansätze auch schon deutlich früher. In den letzten Jahren sind aber kontinuierliche Daten durch Sensoren und Vernetzung sehr viel häufiger aufgetreten. Dadurch ist der grundsätzliche Bedarf an einer breiteren Forschung und Entwicklung deutlich gestiegen. Desweiteren können durch die Verwandtschaft mit inkrementellen anytime bzw. online Ansätzen viele Ideen aus diesem Bereich übernommen oder relativ einfach angepasst werden.

Mittlerweile gibt es Arbeiten aus zahlreichen Bereichen wie Assoziationsregeln, Frequent-Itemsets, Change-Detection, Klassifikation, Entscheidungsbäume, Clustering, Sampling, Histogramme, Quantils [MRL98], Pattern-Mining, Recognition u.v.m.. Das Folgende Kapitel gibt einen Überblick zu einigen ausgewählten Bereiche. Dabei werden von den wichtigsten existierenden Verfahren nur die entscheidenden Ideen grob erläutert. Für eine detaillierte Beschreibung sei auf die jeweils angegebenen Veröffentlichungen verwiesen.

3.1 Sampling, Quantil und Histogramm

Sampling ist eine einfache aber auch wichtige Data-Mining-Applikation, gerade auf Datenströmen. Mit einer repräsentativen Stichprobe lassen sich viele Anfragen schnell und effektiv approximativ beantworten. Die einfachste Variante ist das “Simple Random Sampling” [Vit85]. Bei der Stichprobe von n Objekten eines Datenstroms, der bisher t Elemente geliefert hat, soll jedes Element x aus dem Datenstrom mit der gleichen Wahrscheinlichkeit $\frac{n}{t}$ in der Stichprobe vorkommen. Die Update Funktion nimmt dabei mit der vorgegebenen Wahrscheinlichkeit ein neues Element auf, wobei es mit einem zufällig ausgewählten Element aus der bisherigen Stichprobe ausgetauscht wird.

Wenn die Aufgabe allerdings darin besteht, für ein Zeitfenster fester Größe w eine zufällige Stichprobe zu ziehen, funktioniert dieser Ansatz nicht mehr. Wenn ein Element der Stichprobe aus dem Zeitfenster herausfällt, muss es durch ein neues aus dem aktuellen Zeitfenster ersetzt werden. Da man aber vermeiden will, das aktuelle Zeitfenster komplett zu speichern (sonst bräuchte man keine Stichprobe), kann man darauf nicht mehr zugreifen. Für diesen Fall gibt es das Chain-Sample-Verfahren ([BDM02]). Hierbei wird ausgenutzt, dass man jedesmal, wenn ein neues Element in die Stichprobe eingefügt wird, schon weiß, dass es zu einem Zeitpunkt i wieder aus dem Zeitfenster und damit auch aus der Stichprobe gelöscht wird. Das Element, mit dem es ersetzt wird, muss also im Fenster von $i + 1$ bis $i + w$ liegen. Nun wird beim Einfügen eines Elements schon entschieden, durch welches Element es später ersetzt wird. So kann dieses Element, wenn es auftritt, zwischengespeichert und später in die Stichprobe aufgenommen werden.

Quantil-Zusammenfassung Eine der häufigsten Anwendungen für ein Sampling ist die Quantil-Berechnung. Eine Anfrage besteht aus einem absoluten oder relativen Rang, wobei als Antwort das Element auf diesem Rang zu liefern ist. Das r-Quantil einer endlichen numerischen Datenmenge $X = \{v_1, \ldots, v_n\}$, wobei $1 \leq r \leq n$ der Rang ist, ist der Wert $q_r \in X$, der an der r-ten Position der sortierten Menge X steht, also für den

$$|\{v \in X | v \leq q_r\}| = r$$

gilt. Für einen relativen Rang r mit $0 \leq r \leq 1$ gilt für den Quantilwert q_r:

$$\frac{|\{v \in X | v \leq q_r\}|}{|X|} = r.$$

Im Bereich der Quantil-Berechnung auf Datenströmen gibt es 3 verschiedene Anwendungsszenarien:

- **Datenstrom Modell:** Eine Quantil-Anfrage bezieht sich zu einem Zeitpunkt t auf den vollständigen, bisher gesehenen Datenstrom $v_0, \ldots, v_t$. Die absolute Anzahl der zugrunde liegenden Datenmenge für die Quantil-Berechnung vergrößert sich dabei ständig, ohne dass eine obere Schranke existiert.
- **Sliding Window Modell:** Die Anfrage bezieht sich immer auf die letzten N Daten eines Datenstroms $(v_t - N + 1, \ldots, v_t)$. Dieses Fenster verschiebt sich bei jedem neu eintreffenden Element.
- **n-of-N Modell:** Die Grundlage dieses Modells ist wieder ein Sliding Window über die letzten N Elemente. Eine Anfrage bezieht sich allerdings nicht auf das gesamte Window. Das Ziel ist, zu jeder Zeit für jedes $n \leq N$ ein gewünschtes Quantil unter den letzten n Elementen auf Anfrage berechnen zu können.

Grundlage für die Berechnung von Quantilen ist immer eine Zusammenfassung (auch Sample, Summary, Sketch oder Synopsis genannt) der betroffenen bzw. interessanten Daten. Desweiteren besteht der Wunsch nach garantierten Eigenschaften, insbesondere bezüglich des möglichen Fehlers. Typischerweise werden hierbei ϵ-approximative Antworten erwartet, d.h. für den wahren Quantilwert q' einer Anfrage, die mit q beantwortet wurde, gilt $q' \in [q - \epsilon, q + \epsilon]$ (siehe auch [MRL98]).

Derartige Fehlerbegrenzungen können allerdings nicht mit den oben beschriebenen zufälligen Samples ermöglicht werden. Daher werden in der Quantil-Berechnung mittlerweile deterministische Algorithmen benutzt. Der Zusammenhang zwischen garantiertem Fehler ϵ und dem dafür benötigten Speicher spielt eine zentrale Rolle. Dieser sollte für einen konstanten garantierten Fehler möglichst klein sein.

Manku, Rajagopalan und Lindsay [MRL98] präsentierten einen One-Pass-Algorithmus, der einen festen Fehler garantiert und $O(\frac{1}{\epsilon}\log^2(\epsilon N))$ Speicher benötigt. Allerdings ist die Kenntnis von N bereits vorab notwendig, was bei Datenströmen allgemein nicht möglich ist. Ihr Ansatz benutzt dabei zahlreiche Stichproben, die bei Bedarf vereinigt werden.

Greenwald und Khanna [GK01] geben einen weiteren Algorithmus an, der einen maximalen Fehler ϵ garantiert und dafür nur $O(\frac{1}{\epsilon}\log(\epsilon N))$ Speicher benötigt. Insbesondere ist die Kenntnis von N nicht mehr im Voraus notwendig. Die gespeicherten Daten sind von der Form $(v_1, r_1^-, r_1^+), \ldots, (v_n, r_n^-, r_n^+)$, wobei r_i^- eine garantierte untere Grenze für den Rang von v_i ist und r_i^+ der maximal mögliche Rang. Nun muss beim Update sichergestellt werden, dass $r_i^+ - r_{i-1}^- < 2\epsilon$ bleibt, was mit dem angegebenen Speicherbedarf gewährleistet werden kann.

Da der Speicherbedarf aber immer noch von N, der Anzahl aller bisher gesehenen Daten, abhängt, widerspricht er den Anforderungen von Domingos und Hulten (Abschnitt 2.4) und damit ist diese Variante auch nicht wirklich für potentiell unendlich lange Datenströme geeignet. Um dies zu umgehen, wurde dieser GK-Algorithmus von Lin, Lu, Xu und Xu Yu [LLXY04] zu einer Sliding Window Variante auf Datenströmen erweitert. Der Speicherbedarf hängt hierbei nur noch von der konstanten Fenstergröße N ab ($O(\frac{1}{\epsilon^2} + \frac{1}{\epsilon}\log(\epsilon^2 N))$). Hierbei werden die Daten in Blöcke unterteilt und für jeden Block wird mit dem GK-Algorithmus eine Quantil-Zusammenfassung berechnet. Die Blöcke des aktuellen Fensters werden dann geeignet zusammengefasst, um Anfragen zu beantworten. Für die n-of-N Variante werden Blöcke unterschiedlicher Größe benutzt, wobei die aktuelleren die kleineren sind. Ältere Blöcke werden bei Bedarf zusammengefasst. Bei einer Anfrage werden nun die betroffenen Blöcke kombiniert und daraus dann die Antwort berechnet. Arasu und Manku [AM04] verbesserten den Speicherbedarf auf Sliding Windows der Größe N später auf $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon})\log(N))$.

3.2 Frequent-Itemsets und Assoziationsregeln

Bei der Analsyse von Frequent-Itemsets und Assoziationsregeln sucht man nach aussagekräftigen Beziehungen zwischen sogenannten Items. Zunächst aber erst einmal eine formale Definition der Problemstellung:

Definition 3.1 *Gegeben sei eine Menge von* Items $I = \{i_1, i_2, \ldots, i_n\}$. *Eine* Transaktion T *ist eine Teilmenge von Items* $T \subset I$ *und*

$$D = \bigcup T$$

die Menge aller Transaktionen. Eine Assoziationsregel $X \Rightarrow Y$ *ist eine Regel, für die gilt,*

dass

$$X \subset I, Y \subset I \text{ und } X \cap Y = \emptyset.$$

Der Support *(Unterstützung) einer Menge von Items* X *(*Itemset *genannt) ist definiert mit*

$$support(X) = |T \in D | X \subset T|.$$

Der Support einer Assoziationsregel $X \Rightarrow Y$ *ist*

$$support(X \Rightarrow Y) = support(X \cup Y)$$

und die Confidence *(Vertrauen) ist definiert mit*

$$conf(X \Rightarrow Y) = \frac{support(X)}{support(X \cup Y)}.$$

Gesucht sind nun in den Transaktionen auftretende Itemsets mit hohem Support (sogenannte *Frequent-Itemsets*) und Assoziationsregeln mit hoher Unterstützung und Vertrauen. Eine ausführlichere Problembeschreibung ist u.a. in [Mit97] zu finden.

Mit der Suche nach Frequent-Itemsets in Datenströmen wurde schon früh begonnen, wobei typischerweise das Einkaufswagenszenario betrachtet wird. Hierbei wird das Kaufverhalten von Kunden anhand der Waren in den Einkaufswagen analysiert. Die Waren eines Kaufhauses entsprechen den Items und die Waren eines Einkaufswagen bilden ein Itemset. Man sucht dabei Kombinationen von Waren (Frequent-Itemsets), die besonders häufig von Kunden zusammen eingekauft werden. Jedesmal, wenn ein Kunde durch die Kasse kommt, entsteht ein neuer Datensatz, wodurch automatisch ein Strom von Daten entsteht. Falls man diese Daten online analysieren möchte, ist eine spezielle Datenstromvariante notwendig, da die Standardverfahren (z.B. Apriori-Algorithmus [AIS93]) zahlreiche Durchläufe erfordern. Ein großes Problem bei der Analyse von Frequent-Itemsets und vor allem auch bei Assoziationsregeln auf Datenströmen ist, dass man nicht weiß, welche und wieviel Warenkörbe später noch kommen, so dass man zwischenzeitlich auch keine Kombination von Waren endgültig ausschließen kann. Dies muss man aber natürlich trotzdem tun, da nicht alle möglichen Kombinationen von Waren verfolgt und gespeichert werden können. Es ist also nicht möglich, absolut korrekte Ergebnisse zu garantieren, weshalb man sich auf approximative Ergebnisse beschränken muss. Assoziationsregeln haben zusätzlich das Problem, dass sie typischerweise erst nach der Generierung der Frequent Itemsets erzeugt werden. Gerade die Berechnung der Konfidenz von potentiellen Ergebnissen erfordert eigentlich einen kompletten zusätzlichen Datendurchlauf. Aus diesen Gründen haben Datenstrom-Verfahren die Option eines zweiten Durchlaufs, der Support und Konfidenz der potentiellen Regeln nochmal exakt berechnet. Beim Einkaufswagen-Szenario möchte man zwar die Daten Online analysieren, sie werden dann aber häufig trotzdem in einem Data Warehouse abgespeichert, wodurch ein nachträglicher Durchlauf doch noch möglich ist.

Im statischen Szenario wird häufig der Apriori-Algorithmus als Grundlage benutzt, bei dem über die Länge der Frequent Itemsets iteriert wird und jedesmal ein kompletter Datendurchlauf notwendig ist. Daher ist er für das Online Szenario absolut ungeeignet. Ein Hidber präsentiert in [Hid99] einen neuartigen Datenstrom-Ansatz. Dieser erlaubt es dem Benutzer zusätzlich, jederzeit die Grenzen für Support und Konfidenz zu ändern. Als Grundlage für den Algorithmus dient eine Schätzfunktion der oberen Grenze des Supports eines Itemsets v zum Zeitpunkt n:

$$\mathrm{support(v)} \leq \mathrm{average}(\lceil \mathrm{s_n} \rceil) + \frac{|\mathrm{v}| - 1}{\mathrm{n}}.$$

Hierbei ist $\lceil s_n \rceil$ die kleinste monoton fallende Sequenz, die bis zum Zeitpunkt n punktweise größer als die bisherigen Grenzwerte s_i für den Support ist und $|v|$ bezeichnet die Größe des Itemsets v. Mithilfe dieser Schätzfunktion sichert das Verfahren ab, zu jedem Zeitpunkt alle potentiellen häufigen Itemsets im Speicher zu halten. Zusätzlich wird natürlich auch noch ausgenutzt, dass alle Teilmengen mindestens den gleichen Support besitzen müssen, also als obere Schranke dienen können. Jederzeit kann dann mit einer durchschnittlichen Schätzung für den Support eine aktuelle Itemset-Menge angegeben werden, aus dieser sich wiederum Assoziationsregeln ableiten lassen. Eine Beschreibung des Verfahrens ist in [Hid99] zu finden.

Manku und Motwani [MM02] veröffentlichten das Verfahren *Loss Counting*, um Frequent Itemsets auf Datenströmen zu finden. Das Verfahren hatte allerdings das Problem, dass der Speicherbedarf von der Anzahl der bisher gesehenen Datenstrom-Elemente abhängt und damit nur bedingt für Datenströme geeignet ist. Es eignete sich allerdings sehr gut als Grundlage für eine etwas veränderte und einfachere Problemstellung, der Suche nach Frequent Itemsets in festen Zeitfenstern (Sliding Window) eines Datenstroms. Hier ist durch die Größe des Zeitfensters die Anzahl der betrachteten Daten begrenzt und bekannt, so dass auf Grundlage von Loss Counting hierfür mehrere Verfahren entwickelt wurden [AM04, CL03, CL04]. Darüber hinaus wurden in den letzten Jahren noch weitere Verfahren für diese Problemstellung entwickelt [CL05, CWYM04].

3.3 Clustering

Unter Clustering versteht man die „sinnvolle“ Einteilung von Daten in Gruppen, die Cluster genannt werden. Sinnvoll bedeutet hier, dass Daten innerhalb eines Clusters ähnlich sind, während Daten unterschiedlicher Cluster unähnlich sind. Für das traditionelle Clustering auf statischen Daten existieren eine Vielzahl unterschiedlicher Ansätze (z.B. partitionierende Ansätze, hierarchische Clusterverfahren, dichtebasierte Verfahren, Verfahren auf Grundlage von Neuronalen Netzen und graphentheoretische Verfahren), wobei je nach Ansatz unterschiedliche Gruppierungen als ideal betrachtet werden.

Für Clustering mit Datenströmen sind grundsätzlich zwei verschiedene Szenarien denkbar (siehe Abschnitt 2.4):

1. Szenario: Clustering auf Datenströmen In dem ersten möglichen Szenario werden die Daten eines Datenstroms in Cluster eingeteilt. Es können dabei alle Daten berücksichtigt werden oder nur die Daten innerhalb eines Zeitfensters. In beiden Fällen muss jedesmal, wenn ein neues Element hinzukommt, die Clusterstruktur angepasst werden.

In den letzten Jahren wurden einige One-Pass-Algorithmen für dieses Szenario entwickelt und veröffentlicht. Allerdings verlangten die ersten Verfahren noch die Kenntnis der gesamten Datenanzahl im Voraus [GMMO00, GMM+03]. Dadurch sind sie nicht wirklich auf Datenströme anwendbar. Sie sind weder online noch permanent benutzbar. Mittlerweile gibt es auch einige flexiblere Ansätze und Erweiterungen, die ebenfalls die Voraussetzungen für Datenstrom-Verfahren erfüllen, wobei zwei grundsätzlich verschiedenen Vorgehensweisen zu erkennen sind. In der einen Strategie wird die aktuelle Clusterstruktur bei jedem neuen Elementen aktualisiert, indem man das neue Element einem der bisherigen Cluster zuordnet oder ein ganz neues Cluster erzeugt. Das Verfahren von Aggrarwal [AHWY03] geht diesen Weg, allerdings werden hierbei deutlich mehr als die geforderten K Cluster erzeugt. Daher werden diese kleinen Cluster "Microcluster" genannt. Die Anzahl dieser Microcluster ist nur durch die Speichergröße und die Verarbeitungszeit begrenzt. Falls eine konkrete Clusterstruktur gewünscht wird, müssen die Microcluster nochmals mit einem auf K-Median basierten Clusterverfahren zu einem endgültigem Ergebnis gruppiert werden. Ein weiterer Ansatz mit einer derartigen Strategie ist *DenStream* [CEQZ06], das den DBSCAN-Algorithmus als Grundlage verwendet, um Cluster zu erzeugen.

Bei der anderen Strategie wird das Clusterverfahren zunächst nur auf Teilblöcke des Datenstroms angewendet. Die Clusterstrukturen der einzelnen Teilblöcke werden anschließend durch die rekursive Anwendung des Clusterverfahrens auf die erzeugten Cluster vereinigt. An der Spitze dieses hierarchischen Ansatzes steht das letztendliche Clusterergebnis. Das Verfahren aus [GMM+03] benutzt diese Variante auf Basis des K-Means Algorithmus. Allerdings bedarf es der Kenntnis der Anzahl von Objekten, die verwendet werden sollen, um die Blockgrößen zu bestimmen. Dadurch ist es nicht für Datenströme geeignet. Es gibt aber eine Erweiterung in Zusammenarbeit mit Babcock [BDMO03], die den Ansatz auf Sliding Windows erweitert. Eine Sonderrolle spielt hierbei der Block, dessen zugehörige Daten nur noch zum Teil im aktuellen Fenster liegen. Die Cluster dieses Blocks werden deshalb mit einem entsprechenden Faktor abgewichtet. Weitere Verfahren, die diese Strategie-Variante benutzen sind *ICFR* [MMS04] unter Verwendung einer Regressionsanalyse und *DUCStream* [GLZT05], das den CCA-Algorithmus zum Clustern verwendet.

Verfahren, die K-Means bzw. K-Median-Algorithmus als Grundlage benutzen, profi-

tieren zwar von dessen Effizienz, haben aber auch den großen Nachteil, dass die Anzahl der Cluster K konstant ist und vorgegeben werden muss. Gerade beim Clustern hat man meistens kaum Vorkenntnisse der Daten, was im Datenstrom-Szenario noch dadurch verstärkt wird, dass man die Daten noch nicht einmal vorliegen hat. Weder die Anzahl der Daten noch die Frequenz sind bekannt. Insbesondere braucht es gar keine "korrekte" Clusteranzahl über den gesamten Datenstrom zu geben, da diese sich auf völlig unbekannte Weise entwickeln und damit auch verändern kann.

Desweiteren wird bei den Verfahren von einem konstanten wahren Konzept oder einer gleichbleibenden Veränderung innerhalb eines Zeitfensters ausgegangen. Eine direkte Erkennung von derartigem Concept Drift, der dem Concept Drift beim überwachten Lernen ähnelt (siehe Abschnitte 3.4.1 und 5.1.3), ist im unüberwachten Lernen allerdings noch deutlich schwieriger als beim überwachten Lernen. Deshalb wurde bisher noch keine Strategie zur Erkennung von Concept Drift entwickelt.

2. Szenario: Clustering von Datenströmen In diesem Szenario sind die Datenströme selbst die Datenobjekte, die in Cluster aufgeteilt werden. Ein Datenstrom repräsentiert hierbei die Entwicklung eines Wertes, wobei ähnliche Entwicklungsmuster gefunden werden sollen. In diesem Szenario kommen die Objekte nicht sukzessive hinzu, sondern sie verändern sich permanent, wodurch sich auch die Clusterstruktur und Clusteranzahl ständig ändern kann. Hierbei ist ebenso denkbar, dass neue Datenströme jederzeit hinzugefügt und alte Ströme entfernt werden können.

Für dieses Szenario gab es bislang noch keine veröffentlichten Algorithmen, was zur Entwicklung eines solchen motivierte. Eine Clusteranalyse ganzer Datenströme macht immer dann Sinn, wenn eine große Anzahl paralleler homogener Ströme vorhanden sind, z.B. Aktienkurse, in der Medizin, beim Wetter usw. Das Verfahren bedarf hoher Flexibilität, da alle Objekte sich ständig verändern, und damit sich auch die Clusterstrukturen verändern können. Im Kapitel 4 wird das neu entwickelte Verfahren vorgestellt.

3.4 Klassifikation

Das Problem der Klassifikation gehört zu den überwachten Lernverfahren. Anhand einer Menge von Attributen soll der Wert eines weiteren diskreten Attributs, dessen Ausprägungen Klassen oder Label genannt werden, vorhergesagt werden. Zum Lernen dient eine Trainingsmenge, für die dieses Klassenattribut schon gegeben ist. Derartige Daten werden hier auch als gelabelte oder markierte Daten bezeichnet. Nach dem Lernen soll das Verfahren in der Lage sein, neue Daten mit unbekanntem Klassenattribut möglichst korrekt zu klassifizieren. Für die traditionelle Klassifikation gibt es zahlreiche verschiedene Ansätze, z.B. instanzbasierte Verfahren, Entscheidungsbäume, Regellerner, kernelbasier-

te Ansätze und Neuronale Netze.

Das Problem der Klassifikation scheint auf den ersten Blick mit Datenströmen kaum vereinbar zu sein. Klassifikation ist von Natur aus in zwei separate Phasen unterteilt. Zunächst wird aus einer Traingsmenge ein Modell gelernt, mit dessen Hilfe dann neue Daten klassifiziert werden können. Für die zu klassifizierenden Daten sind auch bisher schon statische Mengen genauso denkbar wie Datenströme, aber diese zweite Phase ist für das Maschinelle Lernen auch der weniger interessante Teil. Das Erlernen eines möglichst guten Modells ist das Hauptproblem. Möchte man dies auf das Datenstrom-Szenario übertragen, muss es einen nicht endenden Datenstrom von Trainingsdaten geben. Sind auch die in der Zukunft erscheinenden Daten Trainingsdaten, kann zwar ein Modell sukzessive gelernt werden, da sie aber schon markiert sind, ist eine Klassifikation nicht mehr notwendig, und so auch das Lernen des Modells überflüssig. Damit Klassifikation auf Datenströmen Sinn macht, muss es ein Szenario geben, in dem jederzeit markierte neue Trainingsdaten und zu klassifizierende Daten auftauchen. Auf den ersten Blick scheint dies schwer möglich zu sein. Allerdings gibt es für Trainingsdaten viele verschiedene Arten von Quellen. Eine Möglichkeit wäre, dass es zwar zuverlässige Klassifzierer gibt, die aber sehr teuer, aufwändig oder aus sonstigen Gründen nicht zur Klassifikation aller Daten geeignet sind. Beispiele dafür sind menschliche Experten, ausführliche Tests oder Analyse des Objektes, so dass es anschließend nicht mehr brauchbar ist. In diesen Fall lohnt es sich nur für eine geringen Anteil der Objekte, die wahre Klasse zu bestimmen. Eine weitere Möglichkeit wäre, dass die wahre Klasse neuer Daten erst nach deren Verwendung oder aus sonstigen Gründen erst nach einiger Zeit vorhanden ist, die Klassifikation aber schon sofort benötigt wird. Ein bekanntes Beispiel hierfür ist die Spamerkennung, wo der Benutzer erst im Nachhinein falsch klassifizierte Mails richtig kennzeichnet. Ein anderes Beispiel ist die Prozesskontrolle: Wenn eine falsche Klassifikation zu einem Fehler in einem Prozess, den es zu kontrollieren gilt, führt, kann man im Nachhinein immer die wahre Klasse der Daten bestimmen. Es sind also durchaus einige realistische Szenarien denkbar, in denen Klassifikation auf Datenströmen sinnvoll ist.

3.4.1 Concept Drift

Dieser Abschnitt gibt nur eine sehr kurzen, für diese Übersicht notwendige, Einleitung über Concept Drift. Eine deutlich detailliertere Beschreibung folgt im Kapitel über Adaptives Lernen im Abschnitt 5.1.3.

Der entscheidende Unterschied zur Klassifikation auf statischen Daten ist ein zeitliches Phänomen, der Concept Drift. Da die Daten über die Zeit erzeugt werden, kann sich auch der zugrunde liegende Prozess ändern. Die Folge ist Concept Drift, eine Veränderung der Daten und des zugrunde liegenden wahren Konzepts. Dies hat bei der Klassifikation die Folge, dass sich auch das gelernte Modell dem anpassen muss. Ohne Concept Drift

hätte man den deutlich einfacheren Fall der inkrementellen Klassifikation. Hierbei wird das Modell durch zusätzliche Daten nur verfeinert, es muss also nur neues Wissen zu dem bisherigen hinzugefügt werden. Dies ist mit Concept Drift komplett anders. Bisher gültiges Wissen kann plötzlich veraltet und inkorrekt sein. Beim Update muss nicht nur neues Wissen hinzugefügt werden, das bisher erlernte Wissen muss auch auf Gültigkeit untersucht werden und kann nicht automatisch als richtig übernommen werden. Bei einer abrupten größeren Änderung des wahren Konzepts spricht man auch von Concept Shift.

3.4.2 Allgemeine Ansätze

Für die Klassifikation auf Datenströmen gibt es mittlerweile einige Verfahren. Unabhängig von dem letztendlich verwendeten Lernverfahren (Entscheidungsbäume, Regellerner, instanzbasierte Verfahren, ...) lassen sich verschiedene allgemeine Ansätze zum Umgang mit Concept Drift erkennen. Die ersten und einfachsten Ansätze vergessen pauschal regelmäßig Daten, ohne darauf zu achten ob, wo und wann Concept Drift auftritt. Das bekannteste und einfachste Verfahren, die aktuellen Daten zu wählen, sind Sliding Windows, die einfach nur die letzten n Datenelemente zum Lernen benutzen (siehe Abschnitt 2.4.2). In jedem Schritt kommt ein neues Element dazu und eins wird unabhängig vom Concept Drift vergessen [WK92, KW95, SRG86, HSD01]. Zwei weitere Ansätze für die Wahl eines aktuellen Datenpools kommen aus dem Bereich der instanzbasierten Ansätze [Sal97], wobei Daten aufgrund ihres Alters oder ihrer Umgebung gelöscht werden. Diese Verfahren werden in Abschnitt 3.4.4 genau beschrieben.

Eine erste Strategie, das Vergessen von Daten an den vorhandenen Concept Drift anzupassen bestand darin, die Größe des verwendeten Sliding Window dynamisch zu ändern. Dabei besteht die Möglichkeit, die aktuelle Klassifikationsrate zu Hilfe zu nehmen [WK96, HSD01]. Falls sich die Klassifikationsrate nicht ändert oder gar verbessert, kann man das Fenster größer werden lassen. Da alte Daten nicht gespeichert werden, ist dies pro Element nur um eins möglich, indem man das älteste Element des Fensters nicht löscht. Wenn sich die Klassifikationsrate verschlechtert, ist das ein Hinweis darauf, dass sich der Daten generierende Prozess geändert hat, und dass das Fenster verkleinert werden sollte. Dies kann wiederum pauschal (z.B. um ein Viertel der Größe des Sliding Window) oder anhand der Verschlechterung individuell geschehen. Eine andere Möglichkeit, die Fenstergröße zu bestimmen, ist, anhand mehrerer Alternativen die zur Zeit beste auszuwählen [Kli04, LVB04]. Hierbei wechselt man die Fenstergröße, weil man weiß, dass man dadurch die Klassifikationsrate auf dem Strom der Trainingsdaten verbessert (dies muss in der ersten Variante nicht der Fall sein), aber das Auswerten verschiedener Fenstergrößen bedeutet auch einen deutlich größeren Aufwand.

Ein ganz anderer allgemeine Ansatz versucht Concept Drift mittels Change Detection [KBDG04] zu erkennen. Das Augenmerk liegt dabei auf der Klassifikationsrate.

Normalerweise, wenn sich das wahre Konzept nicht ändert, verbessert sie sich oder verschlechtert sich zumindest nicht, da neue Daten nur zusätzliches Wissen bedeuten, mit dem das gelernte Modell verfeinert werden kann. Falls sich die Klassifikationsrate aber signifikant verschlechtert, wird angenommen, dass sich der Daten generierende Prozess geändert hat womit Concept Drift vorliegt. Am einfachsten lernt man das gesamte Modell komplett mit aktuelleren Daten neu [CGM03,GMCR04]. Eine erweiterte Variante ist, den Test auch auf Teile des Datenraums anzuwenden, um die Veränderung besser lokal einzukreisen und das Update dann nur lokal durchzuführen [GMR05].

3.4.3 Entscheidungsbäume

Schon in der Klassifikation auf statisch Daten spielen Entscheidungsbäume durch ihre gute Interpretierbarkeit eine wichtige Rolle. Mittels Pruning-Strategien ist auch eine gute Generalisierung gegeben, was zu hohen Klassifikationsraten führt. Allerdings ist eine Veränderung eines Entscheidungsbaums meist auch sehr umfangreich. Falls sich z.B. die Splitentscheidung in der Wurzel ändert, ist der ganze bisher gelernte Baum fast nutzlos und muss neu aufgebaut werden.

Ein Verfahren, das derartige Umbrüche optimiert, ist BOAT [GGRL99]. Hierbei wird versucht, ähnliche Teilbaumstrukturen zu vereinigen oder mehrere Knoten zu rotieren, wenn sich der optimale Split eines Knotens ändert. Damit wird zwar garantiert, dass jederzeit der optimale Split benutzt wird, aber das Verfahren ist trotzdem sehr teuer und die Lastverteilung sehr ungleich. Solange sich kein Split ändert, ist es natürlich sehr effizient, aber wenn ein Splitkriterium ausgetauscht werden muss, tritt eine starke Verzögerung durch die komplexen Veränderungen auf.

Das erste Entscheidungsbaum-Verfahren auf Datenströmen, das sich in relativ kurzer Zeit etabliert hat, ist CVFDT (Concept-adapting Very Fast Decision Tree) von Hulten, Spencer und Domingos [HSD01], eine Weiterentwicklung von VFDT (Very Fast Decision Tree) [DH00]. VFDT basiert auf einen Hoeffding Tree und wurde als rein inkrementelles Verfahren für extrem große Datensätze (Massive Datasets) entworfen. Ein Hoeffding Tree nutzt die Hoeffding Schranke [Hoe63], um zu entscheiden, wie viele Elemente in einem Knoten ausreichen, um sich für ein Splitattribut zu entscheiden. Die Schranke besagt für eine Zufallsvariable a, deren Wertebereich eine Größe R hat: Für den Mittelwert $\overline{a}$ einer Stichprobe von n Beobachtungen ist der (wahre) Erwartungswert von a mit einer Wahrscheinlichkeit von $1-\delta$ größer als $\overline{a}-\epsilon$ ist, wobei gilt

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

Der Hoeffding Tree entscheidet sich in einem Knoten genau dann für einen Split, wenn genügend Daten vorhanden sind, so dass die Gütefunktion G des besten Splitattributs

mit einer vorgegebenen Wahrscheinlichkeit um einen konstanten Wert ϵ größer ist als die des zweitbesten Splitattributs ($G(A) - G(B) > \epsilon$). Als Gütefunktion dienen wie üblich die Entropie-Funktion (Informationsgewinn) oder der Gini-Index. Dies passiert rekursiv an jedem neu entstandenen Knoten. Wenn kein weiterer Split mehr möglich ist, wird der Lernvorgang beendet, unabhängig davon, ob noch Trainingselemente zur Verfügung stehen.

Im VFDT wird als Erweiterung nicht zwischen fast identischen Attributen unterschieden, da dies unnötig viele Daten erfordert, ohne dass ein angemessener Nutzen vorhanden ist. Die Splits an den Knoten werden regelmäßig überprüft und bei Notwendigkeit gelöscht und neu gelernt. Desweiteren beinhaltet der VFDT ein eigenes Speicher-Management, das bei Bedarf Blätter mit dem geringsten Nutzen löschen oder deaktivieren und auslagern kann.

Die Datenstrom-Variante CVFDT benötigt deutlich mehr Flexibilität. Dazu werden an jedem Knoten neben den eigentlichen Splits auch alternative Splits im Hintergrund gelernt. Jeder aktuelle Split wird ständig überprüft und kann gegebenenfalls ohne zusätzlichen Aufwand schnell ausgetauscht oder gelöscht werden. CVFDT arbeitet außerdem mit einem festen Zeitfenster. Über zusätzlich implementierbare Monitorfunktionen wäre auch ein Zeitfenster möglich, dessen Größe dynamisch anpassbar wäre.

Mit diesen Änderungen ist CVFDT zwar in der Lage, mit Concept Drift umzugehen, aber er bleibt spezialisiert für extrem große Datenmengen bzw. Datenströmen mit hoher Datenfrequenz. Die Hoeffding Schranke bedarf einer ausreichend großen Menge an Daten, um sich für ein Splitattribut zu entscheiden. VFDT und CVFDT können ebenfalls nur mit kategoriellen Attributen, die eine geringe Anzahl von Ausprägungen haben, umgehen, da sie, statt die Daten zu speichern, in jedem Knoten alle auftretenden Kombinationen von Attributausprägungen zählen. Trotzdem ist CVFDT ein sehr verbreitetes Verfahren, da es das erste Verfahren auf Datenströmen ist, für das eine Implementierung zur Verfügung steht. In der Literatur wird es deshalb häufig als Vergleichsverfahren herangezogen.

Eine Weiterentwicklung für numerische Daten stammt von Gama et al. [GMR05, GMCR04] und nennt sich UFFT (Ultra Fast Forest Trees). Hierbei wird die Klassifikationsrate in jedem Knoten der Bäume beobachtet. Beim Anschein einer Verschlechterung wird ein alternativer Teilbaum gelernt, der bei signifikanter Verschlechterung den alten ersetzt. Ansonsten gibt es noch zahlreiche Ensemble-Varianten, die auf der Basis von Entscheidungsbäumen arbeiten [KM03, Sta03, SK01, WFYH03]. Dabei werden allerdings meist klassische Entscheidungsbäume (z.B. C4.5) benutzt.

3.4.4 Regel- und Instanzbasiertes Lernen

Zu einem der ersten Lernverfahren, die für Concept Drift entwickelt wurden, gehören die Regellerner STAGGER und FLORA. Diese waren allerdings nur für binäre Klassifikationsprobleme auf kategoriellen Daten anwendbar. STAGGER wurde bereits 1986 von Schlimmer und Granger [SRG86] entwickelt und konnte nur mit ausschließlich booleschen Attributen umgehen. Die dabei verwendeten Regeln, die auch Konzepte oder Charakteristika genannt werden, sind Kombinationen aus Konjunktionen und Disjunktionen von Attributen, z.B. $(A \wedge B) \vee (C \wedge D)$. Jede Regel besitzt Gewichte für Unterstützung und Notwendigkeit, die ständig aktualisiert werden. Zusätzlich wird ein Erwartungswert aus den neuen Instanzen berechnet. Regelmäßig wird die Regelmenge angepasst und neu strukturiert, um auf Änderungen des Konzeptes zu reagieren.

Das FLORA System wurde von Widmer und Kubat zwischen 1992 und 1996 entwickelt und erweitert [WK92, WK93, WK96]. Die erzeugten Regeln werden bei diesem Verfahren in 3 Beschreibungsmengen aufgeteilt. Die Beschreibungsmengen umfassen Regeln, die nur positive Instanzen erfüllen, Regeln, die nur von negativen Instanzen erfüllt werden, und Regeln, die von Instanzen aus beiden Klassen erfüllt werden. Der Update-Prozess besteht einmal aus dem Extrahieren von Regeln aus neuen Daten und dem Vergessen von möglicherweise veralteten Regeln. Als Grundlage benutzt FLORA die Daten eines Sliding Windows fester Größe. Beginnend mit FLORA2 wird die Größe des Fensters auf Basis der Klassifikationrate und der Komplexität des gelernten Konzepts angepasst. In FLORA3 kam die Wiedererkennung von schon einmal existierenden Konzepten hinzu und in FLORA4 wurde eine Strategie für den robusteren Umgang mit Rauschen eingeführt.

Ein relativ neues Regellernverfahren für Datenströme ist FACIL von Ferrer-Troyano, Aguilar-Ruiz und Riquelme (siehe [FTARR06]). Es basiert auf dem AQ11-Verfahren von Maloof [MM04, Mal03]. Die Regeln werden hierbei als Hyperrechtecke dargestellt, die durch Intervalle für jedes Attribut beschrieben werden. Bei diskreten Attributen wird anstatt eines Intervalls eine Menge von Ausprägungen verwendet. Dieses Verfahren ist zunächst ein rein inkrementelles Verfahren, da Regeln nur erweitert werden. Regeln können nicht verfeinert, sondern, falls notwendig, nur gelöscht werden. Dabei besitzt jede Regel ein Fenster von Instanzen, das eine konstante Größe hat. Durch diese vorgegebene feste Größe der Zeitfenster werden Daten wieder automatisch vergessen. Über eine komplexere Bedingung wird zusätzlich versucht, nicht relevante Daten zu löschen. Eine Überlappung von Regeln verschiedener Klassen wird hierbei vermieden. Insgesamt ist FACIL ein relativ schnelles Verfahren, dass sich allerdings nur sehr langsam an Concept Drift adaptiert. Deshalb wird es auch auf eine sich nur langsam verändernde Variante des synthetischen Hyperplane-Datenstroms getestet (siehe Abschnitte 3.5 und 5.4.1).

Instanzbasierte Ansätze gab es trotz ihrer Einfachheit für Datenströme bisher kaum. Der trivialste Ansatz, den man sich denken kann, besteht darin, die Daten eines festen Sli-

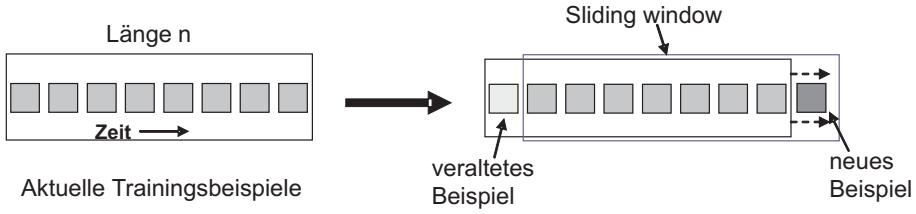

Abbildung 3.1: Beispiel für ein Sliding Window Update

ding Windows als Instanzbasis zu verwenden (siehe Abbildung 3.1). Ein Update aus Lernen und Vergessen ist hierbei extrem einfach und effizient umsetzbar. Im Gegensatz dazu ist die Klassifikation mit der Suche der nächsten Nachbarn aufwändiger als bei Regelverfahren oder Entscheidungsbäumen. Da die Instanzbasis bereits dem Modell entspricht, ist die Auswahl der verwendeten Daten von noch höher Bedeutung als bei Regelverfahren, bei denen durch die Regeln ein weiterer Generalisierungsschritt vorhanden ist.

Salganicoff [Sal97] stellt neben dem Nearest-Neighbor-Verfahren mit Sliding Windows noch zwei weitere Ansätze zur Wahl der Instanzbasis vor. Beim Time-Weighed-Forgetting (TWF) besitzt jedes Element ein Gewicht, wobei neue Elemente immer mit Gewicht 1 eingefügt werden. Bei einem Update werden dann alle Elemente um einen konstanten Faktor abgewichtet. Fällt das Gewicht unter eine benutzerdefinierte Schranke, wird das Element gelöscht (siehe Abbildung 3.2). Dieser Ansatz ähnelt sehr einem festen Sliding Window, da nach einer konstanten Zahl von Updates das Gewicht unter die Schranke fällt, die Daten also nach einer festen Zeit gelöscht werden. Einen zusätzlichen Nutzen hat man nur durch die altersabhängigen Gewichte bei der Aggregation mehrerer nächster Nachbarn (k-NN mit $k > 1$). Das zweite Verfahren arbeitet ebenso mit Gewichten für jedes Element. Bei einem Update werden allerdings die k nächsten Nachbarn des neuen Elements abgewichtet, unabhängig von deren Alter (siehe Abbildung 3.3). Der Faktor für den i-ten nächsten Nachbarn berechnet sich aus

$$\tau + (1 - \tau)\frac{d_i^2}{d_k^2},$$

wobei d_i die Distanz des i-ten nächsten Elements zum neuen Element und τ eine benutzerdefinierte Konstante ist. Der Vorteil dieses Ansatzes ist die Lokalität. Es werden nur dort Daten gelöscht, wo auch neue Daten hinzugefügt werden. Durch eine dynamische Anpassung von k in Abhängigkeit der Anzahl der aktuellen Daten, wird die Größe der verwendeten Instanzmenge bis auf kleine Schwankungen konstant gehalten. Allerdings geschieht das Löschen von Daten komplett unabhängig von der Klassifikationsrate und von Concept Drift und bleibt damit extrem abhängig von dem jeweiligen Datenstrom und der Wahl der benutzerdefinierten Konstanten.

Einen ganz anderen instanzbasierten Ansatz veröffentlichten Law und Zaniolo in [LZ05]. Hierbei wird der gesamte Raum in Parzellen eingeteilt, wozu jedes numerische

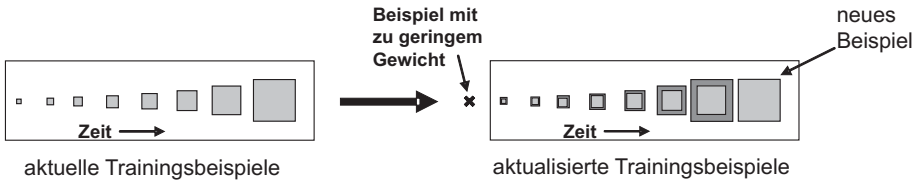

Abbildung 3.2: Update Beispiel für zeitliches Gewichten (TWF)

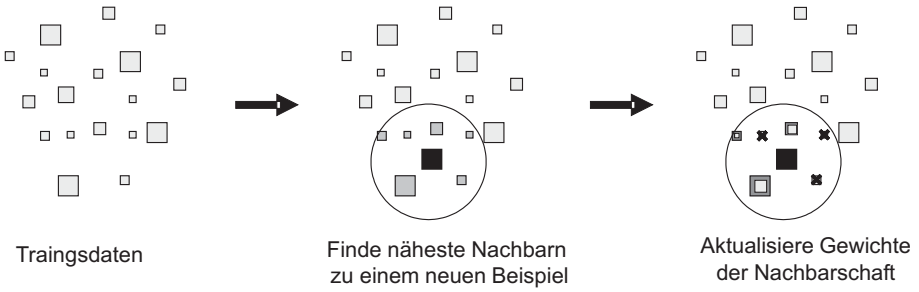

Abbildung 3.3: Update Beispiel für lokales Gewichten (LWF)

Attribut in gleich große Intervalle diskretisiert wird. Für jeder Parzelle wird zunächst ein Präferenzelement erzeugt, das in der Mitte der Zelle liegt. Dieser wird die Klasse zugeordnet, die in der Parzelle momentan am häufigsten auftritt. Für die Behandlung von Concept Drift werden die auftretenden Trainingsdaten wie bei TWF exponentiell gewichtet, womit wieder pauschal auf eventuell auftretenden Concept Drift reagiert wird. Die Menge der Präferenzelemente bilden nun die Fallbasis für die Klassifikation neuer Daten. Dieser Ansatz ist auch erweiterbar durch komplexere Klassifizierer für jede Parzelle oder durch mehrfache parallele Anwendung des Verfahrens mit versetzten Parzellen. Der Hauptnachteil dieses Verfahrens ist neben dem pauschalen Vergessen, dass für eine sinnvolle Klassifikation für jede Parzelle ausreichend Daten vorhanden sein müssen. Bei sehr kleinen Parzellen werden deshalb sehr viele Daten benötigt, womit die Anpassung bei Concept Shift nur sehr träge stattfinden kann. Bei wenigen großen Parzellen hingegen ist das Modell sehr eingeschränkt. Damit ergeben sich sehr ähnliche Eigenschaften wie bei CVFDT.

3.4.5 Weitere Ansätze

Kubat und Widmer entwickelten neben FLORA auch das Verfahren FRANN (Floating Rough Approximation in Neural Networks) [KW95]. Datengrundlage bildet hierbei ein dynamisches Sliding Window. Ein Update findet nur nach jeweils N neuen Trainingsdaten statt. Dabei werden sukzessive Daten aus dem Sliding Window ausgewählt, um eine optimierte Datenbasis zu erhalten. Mit diesen Daten wird dann ein “RBF-Netzwerk” trainiert, das dann auch zur Klassifikation dient. Eine Anpassung der Größe des Sliding Windows

findet in Abhängigkeit der Klassifikationsrate über die letzten M Trainingsdaten statt.

Klinkenberg [Kli04] verwendet Support Vector Machines (SVM) zur Klassifikation auf Datenströmen. Zur Wahl der optimalen Größe des Sliding Windows werden verschiedene Größen getestet und die Größe mit dem besten Ergebnis ausgewählt. Das Verfahren wurde später auf beliebige Klassifikationsverfahren erweitert [SK07].

Insgesamt lassen sich die existierenden Verfahren anhand der zum Testen verwendeten Datenströme in zwei Klassen einteilen. Ein Teil der Verfahren, wie CVFDT und FACIL, verwenden Datenströme, mit vergleichsweise hoher Dimension ($>> 10$), sehr hohe Datenraten besitzen und dessen wahres Konzept sich nur relativ langsam und selten ändert (der Hyperplane-Datenstrom ändert z.B. das wahre Konzept bei CVFDT nur alle 50000 Instanzen geringfügig). Andere Ansätze wie FLORA, FRANN, TWF und LWF dagegen verwenden Datenströme mit wenigen Dimensionen (2-4), die sich sehr oft und schnell verändern (der STAGGER-Datenstrom wechselt z.B. das komplette Konzept bei FLORA schon nach jeweils 40 Instanzen).

3.5 Testdaten

Fast alle Verfahren werden auf synthetischen Daten evaluiert, hauptsächlich deshalb, weil passende Real-World-Daten kaum verfügbar sind. Andererseits besitzen synthetische Daten auch diverse Vorteile. Bei synthetischen Daten kann man den Daten generierenden Prozess genau kontrollieren, man kennt das wahre Konzept und kann so das erzeugte Ergebnis besser beurteilen. Somit sind deutlich gründlichere Tests und Analysen möglich. Typische Daten generierende Prozesse sind Random Walks [ZS02] oder zufällig nach einem zur Problemstellung gehörenden Konzept erzeugte Punkte in einem Datenraum, die einer vorgegebenen Verteilung folgen. Als Real-World-Daten werden oftmals Aktienkurse [ZS02], Weblogs [HSD01, AHWY03] oder aus großen Datenbeständen extrahierte Daten (z.B. [LLXY04]) verwendet.

Bei der Quantil-Berechnung werden häufig Daten synthetisch erzeugt, die innerhalb eines Datenraums einer vorgegebenen Verteilung unterliegen. Benutzte Verteilungen sind hierbei Normalverteilung, Gleichverteilung und exponentielle Verteilungen. In [LLXY04] wird auch ein Real-World-Datensatz benutzt, der auf archivierten Nachrichten der Nachrichtenagentur Reuters beruht, allerdings ist dieser nicht frei zugänglich.

Für Assoziationsregeln wird meist der IBM Testdaten-Generator [AS94] benutzt, um synthetische Daten zu erzeugen. Dabei gibt es die Möglichkeit, die Anzahl der Items und die durchschnittliche Größe von Frequent Itemsets frei zu bestimmen, und damit unterschieldiche Konfigurationen zu testen.

Synthetische Datensätze zur Evaluation von Clusterverfahren werden in der Regel

auf der Grundlage von bekannten Clustern mit Zentrum und fest vorgegebener Verteilung (meist gleichverteilt oder normalverteilt) erzeugt [OMM+02]. Zusätzlich wird eine geringe Menge Rauschen hinzugefügt. Es werden sowohl niedrig dimensionale (<10 Dimensionen) als auch hochdimensionale Daten (>100 Dimensionen) verwendet. Als Real-World-Datensatz werden in [AHWY03] die Aufzeichnungen von Netzwerkzugriffen verwendet. Die zu erkennenden Cluster werden durch unterschiedliches Benutzerverhalten gekennzeichnet, wobei die Hauptunterscheidung zwischen normalen Zugriffen und Attacken stattfindet. Dabei beinhaltet jede einzelne Kategorie meist wieder zahlreiche Unterkategorien, die durch eigene Cluster beschrieben werden.

Im Bereich des adaptiven Klassifizierens mit Concept Drift wird auch hauptsächlich auf künstlichen Datenströmen gearbeitet. Die ersten kategoriellen Daten für STAGGER [SRG86] bestanden aus drei Attributen: Größe (klein, mittel, groß), Farbe (rot, grün) und Form (zirkulär, nicht zirkulär). Das wahre Modell wechselte nach allen 2000 Daten zwischen (Farbe=grün $\vee$ Form=zirkulär) und (Größe= (mittel $\vee$ groß)). In FLORA wurden dann auch numerische Daten verwendet. Der Datenraum ist immer $[0,1]^2 \subset \mathbb{R}^2$. Das zugrunde liegende Konzept besteht jeweils aus zwei Klassen, die durch verschobene Sinusfunktionen mit unterschiedlichen Frequenzen getrennt werden. Nach jeweils 50 Datensätzen werden die Klassen einfach getauscht. Alle benutzten Datenströme beinhalteten nur Concept Shift.

Einen Datenstrom mit echtem Concept Drift wurde für CVFDT [HSD01] verwendet, der Hyperplane-Datensatz genannt wird. Durch einen d-dimensionalen Raum wird eine $(d-1)$-dimensionale Hyperebene gelegt, die den Raum in zwei Klassen teilt. Die Hyperebene wird durch den Normalenvektor und den Mittelpunkt im Raum eindeutig definiert. Um Concept Drift zu erhalten, wurde nun nach jeweils 50000 Daten das Gewicht des Normalenvektors für eine Dimension verändert, wodurch sich die Hyperebene langsam dreht. Auffällig ist hierbei, dass sich das Konzept bisher nach 50 oder 2000 Daten änderte, bei Tests von CVFDT aber erst nach 50000 und dann nur minimal. Bei der Evaluation weiterer Verfahren wurden meist die gleichen Datenströme oder leicht abgewandelte Formen benutzt, wobei allerdings sehr unterschiedliche Werte für die Häufigkeit von Concept Drift gewählt wurden.

Real-World-Daten kamen bisher in der Klassifikation kaum zum Einsatz. Für die Evaluation von CVFDT [HSD01] wurden Weblogs benutzt. Diese wurden durch Sortierung und Löschung von Attributen nachträglich stark verändert, damit überhaupt eine Art von Concept Drift vorhanden ist. Deshalb kann man hierbei nicht mehr wirklich von einem Real-World-Problem reden.

Insgesamt ist zu beobachten, dass die Analyse hauptsächlich mit künstlichen Datensätzen vorgenommen wird. Diese bieten zwar auch mehr Möglichkeiten, trotzdem kommt man bei der Evaluation eines Data Mining Verfahrens nicht umhin, Real-World-Daten zu benutzen, um die praktische Tauglichkeit zu zeigen. Dies geschieht bislang meist nicht

ausreichend, obwohl fast überall darauf hingewiesen wird, wie zahlreich Datenströme in der Praxis auftauchen und deshalb die Verfahren notwendig und relevant sind. Der Grund hierfür liegt in der Schwierigkeit, an interessante und sinnvolle Datenströme zu kommen und in ihrer Verbreitung. Zunächst sind Datenströme in abgespeicherter Form extrem groß und lassen sich damit nicht so einfach verbreiten wie z.B. die gebräuchlichen statischen UCI-Datensätze. Bei synthetischen Datenströmen reicht dagegen die Beschreibung des Daten generierenden Prozesses. Relevante und interessante Datenströme beinhalten auch deutlich mehr Wissen als statische Daten, was oftmals der Grund ist, warum sie nicht frei zur Verfügung gestellt werden. Datenströme wie sie in der Produktion, bei der Beobachtung von Patienten und kritischen Systemen wie z.B. Atomkraftwerken oder auch nur bei Zugriffen auf Webseiten entstehen, enthalten aus Sicht der Eigentümer oft schützenswertes oder unternehmensrelevantes Wissen und werden deshalb nicht frei zugänglich gemacht.

Unabhängig von zur Verfügung stehenden Datensätzen ist aber die Evaluation aufgrund der Stream-Umgebung ein besonderes Problem. Anders als im statischen Fall lässt sich das Ergebnis einer Anfrage nicht in einer eindimensionalen Größe, wie z.B. der Klassifikationsrate, zusammenfassen. Vielmehr werden zu jedem Zeitpunkt eigenständige komplexe Ergebnisse erzeugt, die sich oftmals nur schwer mit den einzelnen Ergebnissen anderer Verfahren vergleichen lassen. Eine gründliche Evaluation ist daher sehr aufwändig und wird nur sehr selten durchgeführt. Viele Verfahren sind zusätzlich nur auf bestimmten Datenströmen mit besonderen Eigenschaften sinnvoll anwendbar, so dass ein allgemeiner Vergleich gar nicht möglich ist.

3.6 Resümee

In diesem Kapitel wurden zahlreiche Verfahren des Data-Mining auf Datenströmen aus den wichtigsten Bereichen vorgestellt. Dieser Bereich ist sehr umfangreich und vielseitig da er zahlreiche verschiedene Fragestellungen mit verschiedenen Varianten umfasst. Dies kommt daher, dass bei der Übertragung von Fragestellungen des statischen Data-Mining auf den Bereich der Datenströme häufig zahlreiche verschiedene Möglichkeiten entstehen, die alle von Interesse sind. Auf der anderen Seite ist die Forschung in diesem Bereich noch relativ jung, weshalb es nicht verwunderlich ist, dass momentan meist nur wenige Ansätze zu den relativ speziellen Problemstellungen existieren. Dadurch entstehen auch Probleme bei der Evaluation, da oft keine oder eigentlich für andere Bedingungen gedachte Verfahren als Vergleichsverfahren existieren. Ein weiteres Problem entsteht durch den Mangel an zur Verfügung stehenden Real-World-Datensätzen. Viele Autoren müssen sich leider damit begnügen, potentielle praktische Anwendungsgebiete zu diskutieren, ohne ihr Verfahren auf echte Probleme anwenden zu können. Aus Sicht des Autors macht Forschung ohne spezielle Anwendungen jedoch nur dann Sinn, wenn

die entwickelten Verfahren möglichst flexibel und unter möglichst unterschiedlichsten Bedingungen anwendbar sind, um somit eine generelle Anwendbarkeit zu ermöglichen. Aus diesem Grund wurde bei der Entwicklung der Verfahren, die in den folgenden Kapiteln vorgestellt werden, insbesondere auf Flexibilität, Erweiterbarkeit und allgemeine Anwendbarkeit geachtet.

Kapitel 4

Clustern paralleler Datenströme

In diesem Kapitel wird ein Clusterverfahren für Datenströme in einem neuartigen Setting vorgestellt. Nach der genauen Problembeschreibung folgt eine Diskussion verschiedener Anwendungsmöglichkeiten eines solchen Verfahrens. Für die detailierte Vorstellung wird anschließend das hierfür zugrunde liegende effiziente Standard Clusterverfahren K-Means sowie dessen Fuzzy-Variante vorgestellt, die zu den partitionierenden Clusteransätzen gehören. Im Hinblick auf die adaptiven Anforderungen für diese Anwendung, wurde Fuzzy-C-Means adaptiv erweitert. Diese Erweiterung ist zwar auch für statische Daten anwendbar, aber die Stärken liegen vor allem in einer inkrementellen Umgebung. Nach einer ausführlichen Beschreibung der notwendigen Vorverarbeitung der Daten, wird die eigentliche Online-Variante des adaptiven Verfahrens präsentiert und evaluiert. Am Ende folgt eine Diskussion über sinnvolle und mögliche Erweiterungen dieses Verfahrens.

4.1 Problemstellung

Wie im Abschnitt 3.3 bereits erwähnt, beschreibt Clustering die Gruppierung oder Partitionierung einer Menge von Objekten in Klassen oder „Cluster“ derart, dass Objekte innerhalb einer Klasse in einem gewissen Sinne ähnlich, und die Objekte unterschiedlicher Klassen unähnlich sind. Das Ergebnis eines Clusterings nennt man Clusterstruktur oder Partition. Die Angabe der Ähnlichkeit zwischen diesen Objekten, z.B. in Form einer Funktion oder Ähnlichkeitsmatrix, ist hierbei eine Voraussetzung. Normalerweise wird dazu ein Ähnlichkeits- oder Abstandsmaß benutzt.

Das Ziel des in diesem Kapitel vorgestellten Verfahrens ist es, eine Menge von Datenströmen über einen aktuellen Zeitabschnitt zu clustern. Im Gegensatz zu den in Abschnitt 3.3 vorgestellten Verfahren, werden hier mehrere Datenströme aufgrund ihres Verlaufs innerhalb eines Zeitfensters (Sliding Window) verglichen, um so zu jedem Zeitpunkt eine aktuelle Partition (Gruppierung) der Ströme bezüglich ihrer Tendenz zu bekommen. Voraussetzung sind mehrere sich über die Zeit entwickelnde homogene Datenströme. Homogenität bezieht sich hierbei darauf, dass verschiedene Datenströme innerhalb eines Zeitfensters vergleichbar sind und gleichzeitig neue Daten für jeden Datenstrom erscheinen. Anhand der aktuellen Entwicklung jedes Datenstroms sollen diese in Cluster eingeteilt werden, wobei die Entwicklung der Ströme innerhalb eines Clusters ähnlich ist. Hierbei wird zunächst verlangt, dass diese ähnliche Entwicklung der Ströme zeitgleich geschieht, wobei kleine Verzögerungen trotzdem meist erkannt werden. Allerdings können in einigen Umgebungen auch zeitliche Verschiebungen bei ansonst ähnlichen Verläufen von Interesse sein, weshalb am Ende dieses Kapitels in Abschnitt 4.6.1 eine Erweiterung mit der Einbeziehung von derartigen Time-Shifts vorgestellt wird. Eine idealisierte Veranschaulichung ist in Abbildung 4.1 zu sehen.

Da sich der Verlauf der Ströme innerhalb des aktuellen Zeitfensters ständig verändert,

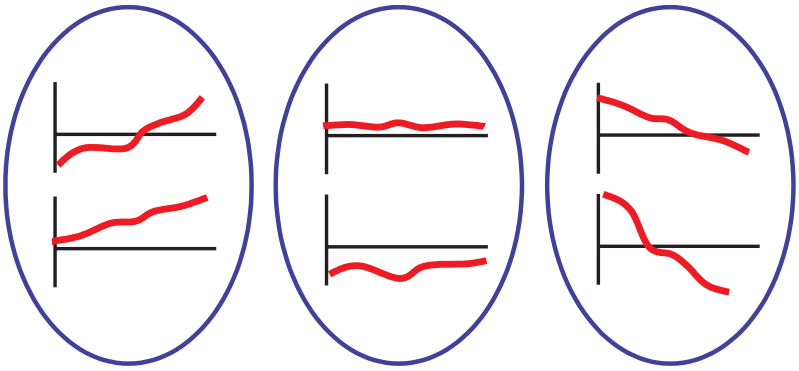

Abbildung 4.1: Beispiel: Clustering von Datenströmen
Die Ströme sind hierbei in „steigend“, „konstant“ und „fallend“ aufgeteilt.

kann sich die Partition ebenso über die Zeit verändern. Datenströme können das Cluster wechseln und es können jederzeit neue Cluster entstehen oder existierende verschwinden. Daher ist ein adaptives Clusterverfahren notwendig, das inkrementell angewendet werden kann und dessen Clusteranzahl variabel und selbst optimierend ist.

4.1.1 Praktische Szenarien

Das oben skizzierte Szenario hat viele praktische Anwendungsmöglichkeiten. Eine allgemeine Anwendung, bei der zahlreiche ständig aktualisierte Messwerte auftreten, ist die System- und Prozessüberwachung. Dies kann technische Bereiche betreffen, wie die Kontrolle einer einzelnen Maschine, einer Produktionsstraße oder eines Motors. Aber es kommen auch komplexere Systeme wie bei der Überwachung von Wetter, Flutwarnsystemen, Straßen- und Flugverkehr in Frage. Bei komplexen Hard- und Softwaresysteme wie großen Datenbanken, Netzwerke und Firmensoftware entstehen zahlreiche Messwerte und Kontrolldaten wie Logfiles zur Überwachung. Eine dynamische Clusteranalyse der zahlreichen Messreihen fasst Parameter zusammen, die sich temporär ähnlich entwickeln. Veränderungen der Clusterstruktur können einen Hinweis auf Systemereignisse wie Fehler oder Zustandsänderungen geben. Durch die gemittelten Clusterverläufe kann die Zahl der zu kontrollierten Parameter verringert werden, wobei nur bei auffallenden Änderungen die einzelnen Parameter direkt betrachtet werden müssen. Desweiteren können Zusammenhänge zwischen den Parametern in Abhängigkeit von unterschiedlichen Systemzuständen beobachtet und analysiert werden.

Eine weitere Anwendungsmöglichkeit sind Multi-User-Systeme, bei denen viele Benutzer zeitgleich über eine längere Zeit agieren. Darunter fallen Online Spiele, bei denen die unterschiedlichen Strategien und Verhalten der Spieler analysiert werden, Internetdienstleister, die an den grundsätzlich unterschiedlichen Verhaltensweisen von Benutzern

und Änderungen von diesen interessiert sind, oder sonstige Webcommunities und Mehrbenutzersysteme. Parallele homogene Datenströme, die für eine Analyse interessant sein können, sind auch Aktienkurse. Gerade zur Überwachung und Analyse von zahlreichen Kursen von unterschiedlichen Börsenstandorten können derartige Analysemethoden von Interesse sein.

4.2 Clustering mit statischen Daten

Dieser Abschnitt gibt eine kurze Einleitung zu K-Means und Fuzzy C-Means. Zunächst ist allerdings noch die formale Definition der Notation sinnvoll und notwendig. Beim Clustern wird eine endliche Datenmenge I aus einem Datenraum D in Cluster eingeteilt, die eine Clusterpartition bilden.

Definition 4.1 *Sei I eine endliche Teilmenge aus dem Datenraum D. Ein* Cluster C *ist eine Teilmenge aus I. Eine* Clusterpartition *(oder* Clusterstruktur*)* $A = \{C_1, \ldots, C_n\}$ *ist eine endliche Menge von Clustern, für die gilt:*

$$\forall C_i, C_j \in A : C_i \cap C_j \neq \emptyset \Rightarrow i = j \tag{4.1}$$

und

$$\bigcup_{C \in A} C = I \tag{4.2}$$

Gemäß dieser Definition teilt eine Clusterstruktur nur die gegebene Datenmenge in Cluster auf. Eine andere Variante wäre eine Aufteilung des gesamten Datenraums D in Cluster, die nur auf Grundlage der Datenmenge I bestimmt wurde. Eine derartige Definition ist sinnvoll, wenn die Datenmenge nur die Rolle einer Trainingsmenge spielt, und die eigentliche Aufgabe in der Aufteilung des Datenraums liegt. Praktisch ist dies auch in Szenarien sinnvoll, in denen neue Daten hinzukommen und direkt einem der bestehenden Cluster zugeordnet werden müssen, wie es beim Clustern auf Zeitfenstern geschieht.

Die Grundidee des Fuzzy-Clusterings ist, dass Objekte zu einem gewissen Grad mehreren Clustern angehören können. Dies erscheint insbesondere dann sinnvoll, wenn auch Daten zwischen den Clustern auftreten können. Im Nicht-Fuzzy-Fall könnte dann zwischen Elementen, die relativ nah sind, ein abrupter Wechsel der Clusterzugehörigkeit auftreten. Beim Fuzzy-Clusterings ist der Übergang hingegen fließend. Die hier verwendete Definition für den Fuzzy-Fall lautet:

Definition 4.2 *Sei I eine endliche Teilmenge aus dem Datenraum D. Ein* Fuzzy-Cluster *C über der Menge I ist eine Funktion $C : I \mapsto [0, 1]$. Eine* Fuzzy-Clusterpartition *(oder* Fuzzy-Clusterstruktur*)* $A = \{C_1, \ldots, C_n\}$ *ist eine endliche Menge von n Fuzzy-Cluster,*

für die gilt:

$$\forall x \in I : \sum_{C \in A} C(x) = 1 \tag{4.3}$$

Von der typischerweise verwendeten Notation über eine Matrix μ von Zugehörigkeitsgraden wurde hier Abstand genommen, da die Verwendung einer Matrix auf einer statischen Menge von Daten beruht. In einer adaptiven Umgebung hingegen können sich die Daten und die Datenmenge jederzeit verändern, so dass die Indizes regelmäßig umbenannt werden müssten. Eine funktionale Schreibweise, ist dahingegen deutlich flexibler und einfacher handhabbar.

Falls, wie im Großteil dieses Kapitels, im Umfeld nur von Fuzzy-Clustering die Rede ist, werden häufig die kürzeren Begriffe Cluster und Clusterpartition (oder Clusterstruktur) ohne das vorangestellte „Fuzzy“ verwendet.

Die Aufgabe eines (Fuzzy-)Clusterverfahrens bzw. eines (Fuzzy-)Clusteralgorithmus ist die Berechnung einer (Fuzzy-)Clusterpartition auf Basis einer gegebenen Datenmenge. Hierfür wird ein Ähnlichkeitsmaß bzw. eine Distanzfunktion benötigt, um entscheiden zu können, welche Daten ähnlich und unähnlich sind. Ziel dabei ist es, dass Objekte innerhalb eines Clusters homogen und Objekte verschiedener Cluster heterogen sind.

4.2.1 K-Means

Eine der bekanntesten Cluster Verfahren ist der K-Means Algorithmus. Aufgrund seiner Einfachheit und Effizienz wird er in vielen Anwendungen verwendet. Die Grundidee des Verfahrens ist die Definition eines Clusters über ein Clusterzentrum und ein Distanzmaß mit den folgenden zwei Bedingungen:

1. Ein Objekt gehört zu dem Cluster, dessen Zentrum das nächste ist.
2. Die Clusterzentren wiederum sind die Mittelpunkte aller Daten eines Clusters.

Dadurch ergibt sich ein iteratives Verfahren zur Ermittlung einer stabilen Clusterpartition, die beide obigen Bedingungen erfüllt, aus einer beliebigen vorgegebenen Partition mit K Clustern und Zentren:

Meist wird als Input eine zufällig erzeugte Clusterpartition gewählt, es gibt aber auch andere Möglichkeiten der Initialisierung. Es kann gezeigt werden, dass die Iteration immer konvergiert [PR67].

Die Objekte bestehen in der Regel aus numerischen Attributen, die einen Vektor $x \in \mathbb{R}^n$ bilden. Als Distanzmaß wird meist der Euklidische Abstand benutzt. Das oben

Algorithmus 4.1 : K-Means Iteration

Input : Clusterpartition mit K Clustern
Data : Objektmenge I
Data : Distanzmaß d

repeat
Zuweisung der Objekte aus I zu dem jeweiligen Cluster mit dem nächsten Clusterzentrum;
Neuberechnung der K Clusterzentren als Mittelpunkte der Objekte der einzelnen Cluster;
until *Clusterpartition bleibt unverändert*;

beschriebene iterative Verfahren minimiert die Zielfunktion

$$\sum_{k=1}^{K} \sum_{x \in C_k} ||x - c_k||^2$$

wobei C_k das k-te Cluster ist und c_k das zugehörige Clusterzentrum. Diese Zielfunktion misst die Ähnlichkeit der Daten zu seinem Cluster, also die Homogenität der Cluster. Die Heterogenität der Cluster, also inwieweit Daten sich aus verschiedenen Clustern unterscheiden, wird hierbei nicht beachtet. In jedem Durchlauf wird diese Zielfunktion kleiner und damit optimiert. Wenn keine Veränderung mehr stattfindet, bricht das Verfahren ab. Allerdings garantiert das Verfahren nicht das Auffinden des globalen Minimums, es konvergiert aber garantiert in einem lokalen Minimum. Welches lokale Minimum erreicht wird, hängt von der Initialisierung der Clusterpartition ab. Daher gilt es, die Clusterpartition möglichst optimal, also möglichst nah am globalen Minimum zu initialisieren. Aus diesem Grunde ist die Initialisierung ein großes Problem bei der Anwendung von K-Means.

Das zweite große Problem ist die Anzahl der Cluster K, die vorab festgelegt werden muss. Die Kenntnis dieser Zahl kann man aber nicht voraussetzen und es ist häufig auch schwierig, einen sinnvollen Wert zu finden. Diesen beiden Problemen, speziell in Hinsicht auf Datenströme, widmen sich zwei noch folgende Abschnitte. Zunächst wird jedoch die Fuzzy-Variante von K-Means vorgestellt.

4.2.2 Fuzzy-C-Means

K-Means weist, wie jedes Nicht-Fuzzy-Clusterverfahren, jedem Objekt genau ein Cluster zu. Allerdings stellt sich dabei die Frage, ob man jedes Objekt sinnvoll genau einem Cluster zuweisen kann und ob jedes Objekt auch genau zu nur einem Cluster gehört. Oftmals lassen sich die Cluster innerhalb eines Datensatzes nur durch die Dichte der auftretenden Objekte festlegen. Cluster stellen typischerweise Regionen mit einer hohen Dichte

von Objekten dar, aber auch zwischen den Clustern existieren Objekte, die den Clustern zugeordnet werden müssen. Zugehörigkeitsgrade zu den einzelnen Clustern können diese Szenarien besser widerspiegeln. Hierbei kann jedes Objekt i zu einem gewissen Grad $C_k(x_i)$ zu jedem Cluster C_k gehören. Bei dem Standardverfahren ist dabei die Summe der Zugehörigkeiten eines Objekts x_i konstant 1 ($\sum_{k=1}^{K} C_k(x_i) = 1$). Der Übergang zwischen zwei Cluster ist hierbei fließend und für zwei Objekte mit geringem Abstand gilt, auch wenn sie weiter entfernt von Clusterzentren liegen, immer, dass ihre Clusterzugehörigkeiten ebenso ähnlich sind. Außerdem beinhalten die Clusterzugehörigkeiten auch bei relativ klar getrennten Clustern zusätzliche räumliche Information. Man kann zum Beispiel die relative Lage der Cluster ablesen. Die Zugehörigkeitsgrade, auch Fuzzy-Zugehörigkeiten genannt, können je nach Anwendung als Präferenz für das wahre Cluster oder als Grad, inwieweit ein Objekt typisch für ein Cluster ist, interpretiert werden.

Die zu optimierende Zielfunktion des Fuzzy-C-Means Algorithmus lautet

$$J = \sum_{k=1}^{K} \sum_{i=1}^{n} C_k(x_i)^m ||x_i - c_k||^2, \tag{4.4}$$

wobei $m \in (1, \infty)$ für den Fuzzifier steht. Mit dem Fuzzifier lässt sich steuern, inwieweit sich die Cluster überlagern dürfen. Für $m = 1$ würde dabei immer eine Nicht-Fuzzy-Lösung die Funktion optimieren. Je größer m ist, desto stärker überschneiden sich die Cluster. Hierbei ist $m = 2$ ein typisch benutzter Wert. Der Algorithmus Fuzzy-C-Means sieht damit wie folgt aus:

Algorithmus 4.2 : Fuzzy-C-Means Iteration

Input : initialisierte Fuzzy-Clusterpartition mit K Fuzzy-Clustern
Data : Objektmenge I
Data : Distanzmaß d

repeat
    Berechne für jedes Objekt die Zugehörigkeiten für die Fuzzy-Cluster;
    Berechne die Clusterzentren der neuen Fuzzy-Cluster;
until $\Delta J < \epsilon$;

Mit ΔJ ist hierbei die Differenz der Optimierungsfunktion J vor und nach dem Schleifenblock gemeint und ϵ ist eine benutzerdefinierte Schranke, die angibt, wann die Lösung ausreichend gut ist. Die Zugehörigkeiten für die Fuzzy-Cluster werden mit folgender Formel berechnet:

$$C_k(x_i) = \left(\sum_{j=1}^{K} \left(\frac{d^2(x_i, c_k)}{d^2(x_i, c_j)} \right)^{\frac{1}{m-1}} \right)^{-1} \tag{4.5}$$

Die Clusterzentren lassen sich berechnen mit:

$$c_k = \frac{\sum_{i=1}^{n} C_k(x_i)^m \cdot x_i}{\sum_{i=1}^{n} C_k(x_i)^m} \tag{4.6}$$

Das „C“ in der Benennung des Fuzzy-C-Means Algorithmus hat sich in der Literatur etabliert. Trotzdem wird die Anzahl der Cluster weiterhin mit K bezeichnet, um keine Konflikte bei Formeln, die auch für den Nicht-Fuzzy-Fall gelten, zu erzeugen. Mit der Schreibweise „(Fuzzy) K-Means“ sind beide hier beschriebenen Clustervarianten gemeint.

4.2.3 Initialisierung

Die Initialisierung bestimmt eine Partition in Form von Clusterzentren, die durch den (Fuzzy-) K-Means-Algorithmus wie beschrieben sukzessive optimiert wird. Die Wahl der Initialisierung hat entscheidenden Einfluss auf die Laufzeit des Algorithmus und vor allem auf das Ergebnis. Dabei ist die Art der Initialisierung bei K-Means und Fuzzy-C-Means identisch. Die am häufigsten benutzte Lösung zur Initialisierung ist die zufällige Wahl von K verschiedenen Objekten als Clusterzentren. Diese Lösung ist eine der schnellsten aber auch kritischsten. Im schlimmsten Fall gehören alle zufällig gewählten Zentren zum selben Cluster der optimalen Lösung. Ziel einer guten Initialisierung sollte es aber sein, möglichst keine Zentren zu bekommen, die in der optimalen Lösung zu einem Cluster gehören.

Eine andere randomisierte Lösung ist die freie Wahl der Clusterzugehörigkeiten aller Objekte. Daraus können die zugehörigen Clusterzentren für die Initialisierung berechnet werden. Allerdings ist dieser Ansatz, wie der vorhergehende auch, nicht nur effizient sondern auch kritisch. Darüber hinaus gibt es auch einige besser randomisierte und deterministische Verfahren [SD04], die versuchen, entweder K Objekte mit möglichst großen Abstand zu finden [BW69], oder die Daten möglichst gut in K Teilmengen aufzuteilen, was aber zu einem deutlich erhöhten Aufwand führt.

Eine einfache und oft benutzte Möglichkeit, das Problem lokaler Optima zu verringern, ist die mehrfache Anwendung von (Fuzzy-) K-Means, wobei die insgesamt beste Lösung letztendlich gewählt wird [HK00, WF05]. Eine optimierte Lösung dazu wird in [BF98] vorgestellt, in der das Clusterverfahren zunächst nur auf kleine Teilmengen der Daten mehrfach angewendet wird. Das beste Ergebnis dient anschließend als Initialisierung für das Clustern der gesamten Datenmenge.

4.3 Bestimmung der Clusteranzahl

Die richtige Wahl der Clusteranzahl ist ein weiteres großes Problem des (Fuzzy-) K-Means-Algorithmus, da dieser Wert schon vor dem Clustern endgültig festgelegt werden muss. In den meisten Anwendungen ist dies aber genauso wenig bekannt wie die eigentlichen Cluster. Zur Lösung dieses Problems gibt es zahlreiche Ansätze, wobei sich die meisten ähneln. Zunächst wird der Clusteralgorithmus für alle möglichen in Frage kommenden Werte der Clusteranzahl ausgeführt. Aus den einzelnen Ergebnissen wird dann auf unterschiedliche Weise das vermeintlich beste Ergebnis ermittelt. Es gibt alternativ noch eine hierarchische Vorgehensweise, bei der das Clusterverfahren zunächst mit einer maximalen Clusteranzahl angewendet wird. Anschließend wird diese durch Zusammenfassen von ähnlichen Clustern sukzessive verringert, bis sich das Clusterergebnis nach einem vordefiniertem Kriterium wieder verschlechtert [KP99, FK97]. Diese zweite Variante findet sich allerdings selten im Zusammenhang mit K-Means-Verfahren. Wenn eine hierarchische Herangehensweise gewünscht wird, kann schließlich direkt auf hierarchische Clusterverfahren zurückgegriffen werden.

Die Strategien zur Auswahl des besten Clusterergebnisses sind sehr verschieden. Visuelle Auswahl, graphische Statistiken und zahlreiche Funktionen werden verwendet. Meist wird dabei auf Gütefunktionen zurückgegriffen, die für ein Clusterergebnis einen vergleichbaren Gütewert liefern. Der Vorteil von Gütefunktionen ist die vollautomatische, einfache und effiziente Anwendung. Als Ergebnis erhält man keine Graphiken oder Statistiken, sondern einfach eine Zahl, die möglichst minimal (oder maximal) sein soll. Leider ist die Bestimmung einer Gütefunktion auch nicht einfach. Gerade für den K-Means-Algorithmus wurden sehr viele Funktionen entwickelt, die je nach Eigenschaft der Daten unterschiedlich gut sind. Damit ist die Auswahl schwierig, vor allem wenn man kaum Wissen über die Art der Daten hat.

4.3.1 Bekannte Gütefunktionen

Für die Wahl einer Gütefunktion gilt zunächst, dass die Zielfunktion des K-Means Algorithmus nicht verwendbar ist, da sie mit der Anzahl der Cluster monoton fällt. Aus den zahlreich veröffentlichten Gütefunktionen werden im Folgenden die bekanntesten für K-Means und Fuzzy-C-Means vorgestellt.

- **Dunn's Index** (nicht Fuzzy)

$$D = \min_{1 \le i < K} \left\{ \min_{i < j \le K} \left(\frac{d(C_i, C_j)}{\max_{1 \le k \le K} diam(C_k)} \right) \right\}$$

  wobei das Unähnlichkeitsmaß $d(C_i, C_j)$ zwischen Clustern definiert ist als

$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} ||x-y||$ und der Diameter eines Clusters mit $diam(C) = \max_{x,y \in C} ||x-y||$

- **Davies-Bouldin's Index** (nicht Fuzzy)

$$DB = \frac{1}{K} \sum_{i=1}^{K} \max_{1 \leq j \leq K, j \neq i} \left\{ \frac{s_{i,q} + s_{j,q}}{||c_i - c_j||} \right\}$$

wobei

$$s_{i,q} = \left(\frac{1}{|C_i|} \sum_{x \in C_i} ||x - c_i||^q \right)^{\frac{1}{q}}$$

und $q \geq 1$ eine frei wählbare ganze Zahl ist.

- **Chou-Su-Index** (nicht Fuzzy)

$$CS = \frac{\frac{1}{K} \sum_{i=1}^{K} \left\{ \frac{1}{|C_i|} \sum_{x \in C_i} \max_{y \in C_i} ||x-y|| \right\}}{\frac{1}{K} \sum_{i=1}^{K} \min_{1 \leq j \leq K, j \neq i} ||c_i - c_j||}$$

- **LCS-Index** (nicht Fuzzy)

$$LCS = \frac{2 \sum_{k=1}^{K} \left(|C_k| \sum_{x \in C_k} (||x - c_k|^2 - m) \right)}{s \sqrt{\left(\sum_{k=1}^{K} |C_k|^2 \right) \left(N^2 - \sum_{k=1}^{K} |C_k|^2 \right)}}$$

wobei

$$m = \frac{1}{N} \sum_{x=1}^{N} ||x_i - M||, s = \sqrt{\frac{1}{N} \sum_{x=1}^{N} ||x_i - M||^2}$$

und M das Zentrum aller Daten ist. Mittelwert und Standardabweichung für den Abstand der Daten zum Zentrum sind dabei m und s.

- **Partition-Koeffizient (PC) und -Entropie (PE)** (Fuzzy)

$$PC = \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{N} C_k(x_i)^2$$

$$PE = -\frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{N} C_k(x_i)^2 \log_2(C_k(x_i))$$

Beide Maße betrachten nur die Zugehörigkeiten, wobei diese möglichst extrem sein sollen, also nahe 0 und 1. Deshalb kann man sie auch nur für Fuzzy-Clustering verwenden. Desweiteren haben beide Maße den Nachteil, dass ihr Wertebereich von der Anzahl der Cluster abhängt, und man so verschiedene Partitionen mit unterschiedlichen Clusteranzahlen nur bedingt vergleichen kann. Darum wurden für beide Maße normalisierte Varianten entworfen:

$$PC_{norm} = 1 - \frac{c}{c-1}(1 - PC)$$

$$PE_{norm} = \frac{1}{\log_2(c)} PE$$

- **Xie-Beni-Index** (Fuzzy)

$$\frac{\sum_{k=1}^{K}\sum_{i=1}^{N} C_k(x_i)^m ||x_i - c_k||^2}{N \min_{1 \le i < j \le K} ||c_i - c_j||^2}$$

Der Xie-Beni-Index ist die vielleicht am häufigsten in der Praxis verwendete Fuzzy-Gütefunktion. Sie lässt sich effizient berechnen und liefert in Experimenten häufig die erwartete Clusteranzahl. Darüber hinaus hat sie die schöne Eigenschaft, die Fuzzy-C-Means Zielfunktion zu beinhalten und damit Widersprüche der beiden Maße für die Präferenz zweier Partitionen mit gleicher Clusteranzahl gering zu halten.

- **Fukuyama-Sugeno-Index** (Fuzzy)

$$FS = \sum_{k=1}^{K}\sum_{i=1}^{N} C_k(x_i)^m \left(||x_i - c_k||^2 - ||c_k - M||^2 \right)$$

M bezeichnet das globale Zentrum aller Daten. Die Eigenschaften dieses Maßes ähneln sehr dem des Xie-Beni-Index. Allerdings kommt es nach [Bor05] nicht gut mit sehr großen und kleinen Werten für den Fuzzifier m zurecht. Große, vom globalen Zentrum weit entfernte Cluster werden stark bevorzugt.

Natürlich kann man die Nicht-Fuzzy-Gütefunktionen auch für Fuzzy-Clustering verwenden, in dem man zunächst das Fuzzy-Ergebnis defuzzifiziert. Dies geschieht sinnvollerweise dadurch, dass jedes Objekt dem Cluster mit dem größten Zugehörigkeitswert zugeordnet wird. Dadurch geht aber das zusätzliche Wissen über die Fuzzy-Zugehörigkeiten verloren. Sinnvoller ist deshalb die Verwendung von (echten) Fuzzy-Gütefunktionen.

4.3.2 Dynamische Bestimmung der Clusteranzahl

Die bisherigen Ansätze zur Bestimmung der Clusteranzahl haben zahlreiche Nachteile. Zunächst muss der Clusteralgorithmus für sehr viele verschiedene Clusteranzahlen ausgeführt werden. Dadurch wird der Vorteil von Fuzzy-C-Means, sehr effizient zu sein, stark beeinträchtigt. Hinzu kommt noch das Problem der lokalen Minimas. Das heißt, selbst für eine feste Anzahl von Clustern liefert der Algorithmus zahlreiche verschiedene Ergebnisse. Um die Wahrscheinlichkeit für eine Clusteranzahl zu erhöhen, auch das dafür beste Ergebnis zu bekommen, muss man Fuzzy-C-Means selbst für diese Clusteranzahl mehrfach ausführen. All dies führt zu einem deutlich uneffizienteren Verfahren, aber vor allem wird aus einem einfachen eleganten Ansatz ein unschönes Verfahren, das alle Möglichkeiten durchtesten muss.

Im Hinblick auf das Ziel, Datenströme adaptiv zu clustern, ist auch eine dynamische Anpassung der Clusteranzahl erforderlich. Dies ist mit den existierenden Ansätzen nicht möglich, da in jedem Schritt das Clusterverfahren mit den Daten mehrfach angewendet werden müsste, um die optimale Clusteranzahl neu zu bestimmen. Ziel eines geeigneten Verfahrens muss es sein, die optimale Clusteranzahl durch wenige Fuzzy-C-Means-Aufrufe und auf inkrementelle Weise zu bestimmen.

Bei der im Folgenden beschriebenen Idee wird die Clusteranzahl nicht mehr aus zahlreichen, unabhängig voneinander berechneten Partitionen ausgewählt, sondern es wird von einer mit einer initialen Clusteranzahl erzeugten Partition ausgehend versucht, diese sukzessive zu verbessern. Aber wie kann man die nicht optimalen Strukturen in einer Partition erkennen? Ausgehend von der optimalen Partition gibt es dazu zunächst zwei grundsätzliche Möglichkeiten. Entweder können Elemente eines Clusters der optimalen Partition mehreren verschiedenen Clustern zugewiesen sein, oder Elemente mehrerer Cluster könnten zu einem Cluster zusammengefasst sein. In beiden Fällen lassen sich die Strukturen durch Löschen bzw. Hinzufügen von einem bzw. mehreren Clustern verbessern. Wenn ein erzeugtes Cluster eigentlich aus zwei Clustern der optimalen Partition besteht, muss ein neues Cluster hinzugefügt werden, um die optimale Partition zu erreichen. Dazu ist es nur notwendig, ein beteiligtes Element als neues Clusterzentrum hinzuzunehmen und die gesamte Partition mit dem Clusteralgorithmus zu optimieren. Im anderen Fall, wenn zwei Cluster der erzeugten Partition in der optimalen Partition nur ein Cluster bilden, kann durch Löschen eines dieser erzeugten Cluster und durch erneute Optimierung durch Fuzzy-C-Means die optimale Partition hergestellt werden.

Natürlich können auch gerade bei größeren Clusteranzahlen komplexere Unterschiede zur optimalen Struktur entstehen, die sich schwieriger erkennen lassen. Solange sich aber die erzeugte Partition und die Clusteranzahl nur gering vom optimalen Ergebnis unterscheiden, desto wahrscheinlicher lässt sich durch Löschen eines nicht optimalen Clusters die Heterogenität der angrenzenden Cluster verringern, oder durch Hinzufügen eines neuen Clusters die Homogenität der nicht optimalen Cluster erhöhen, um so der optimalen

Partition „näher“ zu kommen. Die Kernidee des Verfahrens ist es also, durch Löschen und Hinzufügen von Clustern möglichst die optimale Clusterstruktur zu erreichen. Zur Bestimmung der besseren Clusterstruktur dient hierbei wieder eine Gütefunktion, die dabei auch die besondere Aufgabe hat, zwischen zwei nicht optimalen Partitionen zu entscheiden, welche besser und damit näher am optimalen Ergebnis ist. Eine formale Beschreibung des Verfahrens ist in Algorithmus 4.3 dargestellt. Hierbei wird nach jeder Fuzzy-C-Means Optimierung versucht, die Clusterstruktur durch Hinzufügen oder Löschen eines Clusters zu optimieren. Wenn sie sich nicht mehr verbessern lässt, bricht das Verfahren ab.

Algorithmus 4.3 : Update der Clusteranzahl

Input : Clusterpartition P
Output : Clusterpartition mit optimierter Clusteranzahl

$K \leftarrow$ Clusteranzahl von P;
`count(·)` $\leftarrow 0$;
best $\leftarrow P$;
repeat
  repeat
    Berechne für jedes Objekt die Zugehörigkeiten für die Cluster in P;
    Berechne die Mittelpunkte der neuen Cluster von P;
  until $\Delta J < \epsilon$;
  Ermittle beste K-1 Partition $\rightarrow P_{-1}$;
  Ermittle beste K+1 Partition $\rightarrow P_{+1}$;
  $P \leftarrow$ beste Partition aus $\{P, P_{-1}, P_{+1}\}$;
  $K \leftarrow$ Clusteranzahl von P;
  `count(`K`)` $\leftarrow$ `count(`K`)`+1;
  if P *besser als* best **then** best $\leftarrow P$;
until K *unverändert oder* `count(`K`)` $>K$;
return best;

Da der Fuzzy-C-Means Algorithmus die Zielfunktion und nicht die Gütefunktion optimiert, kann es passieren, dass die Entwicklung der Clusteranzahl in einen Zyklus gerät und der Algorithmus nicht terminiert. Diesem Fall dient die Abbruchbedingung: Falls eine Clusteranzahl K in der Entwicklung häufiger als K Mal auftaucht, bricht man ab und wählt die bisher beste Clusterpartition. Da die Anzahl der möglichen „Korrekturen“ für eine Clusterstruktur mit der Anzahl der Cluster zunimmt, ist es sinnvoll, die maximale Anzahl, in der eine Clusteranzahl wiederholt auftauchen darf, in Abhängigkeit von K festzulegen.

Wie die meisten Hill-Climbing-Verfahren hat dieses Verfahren allerdings auch das Problem, dass man nur ein lokales Minimum finden kann. Der Datensatz in Abbildung 4.2 hat sicherlich lokale Minima für die Clusteranzahl 3 und 9 und es ist nicht zu erwarten,

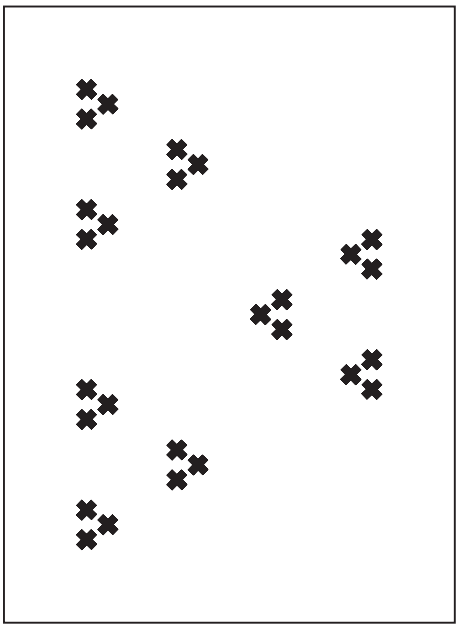

Abbildung 4.2: Beispiel: Datensatz mit mehreren lokalen Minima

dass das vorgestellte Verfahren mit einer sinnvollen Gütefunktion von 3 nach 9 Cluster gelangt oder umgekehrt. Es ist auch zu erwarten, dass die Wahrscheinlichkeit, ein lokales Minimum zu erreichen, mit der Entfernung zur optimalen Clusteranzahl zunimmt, was durch die spätere Evaluation bestätigt wird.

Die Gefahr, in einem lokalen Minimum zu landen, kann durch die Wahl einer geeigneten Gütefunktion allerdings noch verringert werden, weshalb es Sinn macht, die Wahl der Gütefunktion genauer zu betrachten.

4.3.3 Wahl der Gütefunktion

Bei dem beschriebenen Verfahren zur Anpassung der Clusteranzahl hat die Wahl der Gütefunktion sehr großen Einfluss. Allgemein ist für eine Gütefunktion entscheidend, dass sie für die optimale Clusteranzahl minimal bzw. maximal ist. In diesem Ansatz spielen aber noch folgende zusätzliche Anforderungen eine Rolle:

- Da sich die Clusteranzahl immer nur um ± 1 dem optimalen Wert annähert, muss die Gütefunktion in einer möglichst großen Umgebung um die optimale Clusteranzahl monoton steigend (fallend) sein, wodurch die Zahl lokaler Minima minimiert wird.
- Es ist generell möglich, dass durch Optimieren der Clustering-Zielfunktion für eine feste Clusteranzahl die Gütefunktion schlechter wird. Dadurch kann es zu einer

zyklischen Endlosschleife kommen. Um diese Gefahr zu minimieren, sollten die Gütefunktion und die Zielfunktion möglichst „verträglich“ sein.

- Da die Gütefunktion häufig ausgewertet werden muss, sollte sie effizient berechenbar sein.

Die zwei letzten Eigenschaften werden glücklicherweise von der gebräuchlichsten Gütefunktion, dem Xie-Beni-Index, erfüllt:

$$XB = \frac{\sum_{k=1}^{K} \sum_{x=1}^{N} C_k(x_i)^m ||x_i - c_k||^2}{N \min_{1 \leq i < j \leq K} ||c_i - c_j||^2}.$$

Diesen Index gilt es dabei zu minimieren. Der Zähler entspricht genau der Zielfunktion, die hierbei die Homogenität innerhalb der Cluster repräsentiert. Die Heterogenität zwischen den Clustern wird durch den minimalen Abstand zweier Cluster im Nenner ausgedrückt. Die erste Forderung ist allerdings aufgrund dieser Minimum-Berechnung problematisch. Die Abbildung 4.3 zeigt ein typisches Beispiel für den Verlauf der Xie-Beni-Gütefunktion. Hierbei wird ein synthetisch erzeugter 2-dimensionaler Datensatz mit der Clusteranzahl 5 verwendet. Generiert wurden die Daten mit dem Verfahren, das im statischen Experiment (Abschnitt 4.3.6) beschrieben wird. Auffallend hierbei ist der zackige Verlauf für Clusteranzahlen größer als 6 und die damit verbundene hohe Anzahl lokaler Minima.

4.3.4 Modifizierte Gütefunktion

Um nun den Minimum-Effekt der Xie-Beni-Formel zu vermeiden und die Gütefunktion damit zu glätten, wird sie durch eine neue glattere Aggregation der Unähnlichkeiten zwischen allen Clustern ausgetauscht. Der reine Abstand zwischen zwei Clustern ist außerdem ein schlechtes Maß für die Heterogenität oder Unähnlichkeit zwischen den Clustern, da eigentlich auch die Größe bzw. Dichte der einzelnen Cluster eine wichtige Rolle spielt. Ein geringer absoluter Abstand von Clusterzentren muss nicht direkt für ähnliche Cluster sprechen, wenn die Daten der Cluster noch viel näher beieinander liegen. Deshalb wurde zunächst die Unähnlichkeit zwischen zwei Clustern neu definiert, wobei auf ein einfaches Maß geachtet wurde. Anschließend wird die Aggregation der Unähnlichkeit verändert, um einen glatteren Verlauf der Kurve zu garantieren. Bei der Aggregation ist es wichtig, dass die ähnlichsten Cluster den größten Einfluss haben, da diese entscheidend für die Heterogenität zwischen allen Clustern sind. Zunächst sollte man den Abstand der Clusterzentren mit den Dichten der Cluster in Beziehung bringen. Die Dichte D_k eines Clusters C_k wird über den durchschnittlichen Abstand der Elemente zum Clusterzentrum

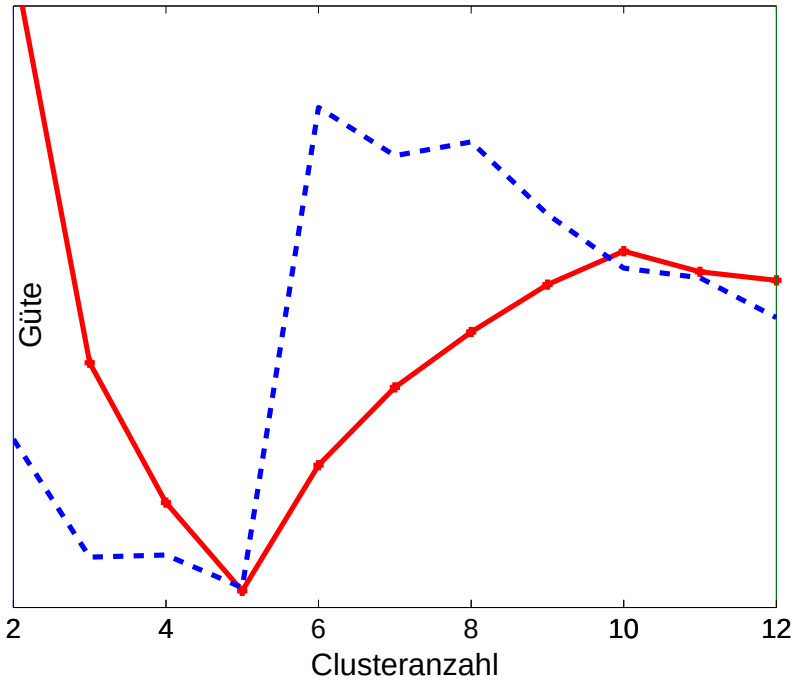

Abbildung 4.3: Vergleich der Xie-Beni-Gütefunktion (blau gestrichelt) mit der neuen Gütefunktion (rot durchgehend).

definiert:

$$D_k = \frac{\sum_{i=1}^{N} C_k(x_i)^m ||x_i - c_k||^2}{\sum_{x=1}^{N} C_k(x_i)^m}.$$

Die Unähnlichkeit zweier Cluster wird nun definiert durch:

$$H(C_k, C_l) = \frac{||c_k - c_l||^2}{D_k \cdot D_l}.$$

Als Heterogenitätsmaß der gesamten Clusterstruktur werden die Unähnlichkeiten aller Clusterpaare wie folgt aggregiert:

$$\sum_{1 \leq k < l \leq K} \frac{1}{H(C_k, C_l)}.$$

Durch die Summation der inversen Unähnlichkeiten haben die geringsten Unähnlichkeiten das größte Gewicht. Die komplette Gütefunktion lautet nun:

$$\frac{1}{N}\sum_{k=1}^{K}\sum_{x=1}^{N} C_k(x_i)^m ||x_i - c_k||^2 \sum_{1\leq k<l\leq K} \frac{1}{H(C_k, C_l)}$$

4.3.5 Bestimmung der besten Clusterpartitionen

Ausgehend von einer Partition mit K Clustern eine bestmögliche Partition mit $K-1$ oder $K+1$ Clustern zu finden, ist sicherlich nicht trivial. Im ersten Fall muss ein Cluster gelöscht, im zweiten ein neues Cluster hinzugefügt werden. Die Frage ist nun, welches Cluster entfernt bzw. hinzugefügt werden soll. Hierbei ist die $K-1$ Partition sicherlich einfacher zu bestimmen, da man nur K verschiedene Cluster als Möglichkeit zum Löschen hat. Dazu wird nacheinander jedes Cluster aus den K Clustern testweise gelöscht und die beste Partition aus diesen K Partitionen gewählt. Allerdings ist direkt nach dem Löschen eines Clusters die neue Partition, vor allem in der Umgebung des gelöschten Zentrums, nicht optimal, weshalb noch M Fuzzy-C-Means-Optimierungsschritte ausgeführt werden. Eine komplette Fuzzy-C-Means-Optimierung durchzuführen, wäre sicher zu aufwändig. Anschließend kann man die Gütefunktion berechnen und alle K Werte vergleichen.

Algorithmus 4.4 : Optimiertes Entfernen eines Clusters

Input : T ist Partition mit K Clustern
Output : N ist Partition mit K-1 Clustern

foreach *Cluster c* **do**
  S_c=T ohne Cluster c;
  for $i = 1$ **to** M **do**
    Berechne für jedes Objekt die Zugehörigkeiten für die K-1 Cluster in S_c;
    Berechne die Mittelpunkte der neuen Cluster in S_c;
  end
  Berechne Gütefunktion von S_c;
end
Setze N = Nach Gütefunktion beste Partition S_c;

Die Suche nach einer guten $K+1$ Partition ist deutlich schwieriger. Es muss ein neues Cluster hinzugefügt werden, wofür es viel mehr Möglichkeiten als beim Löschen eines Clusters gibt. Zunächst stellt sich die Frage, wie überhaupt ein neues Cluster entstehen kann. Dabei bietet es sich an, wie bei der Initialisierung einen Datenpunkt als neues Clusterzentrum zu verwenden. Allerdings gibt es dafür immer noch N Möglichkeiten. Man benötigt deshalb eine möglichst gute Auswahl der Daten, die als neue Clusterzentren in Frage kommen, also solche Daten, die am wahrscheinlichsten in ein neues Cluster

gehören. Ein guter Indikator hierfür ist der Abstand eines Datenpunktes zu seinem Clusterzentrum bzw. die gewichtete Summe dieser Abstände im Fuzzy-Fall. Eine einfache Variante besteht darin, die Daten mit dem größten Abstand als neue Zentren zu benutzen, was allerdings ein Problem bei verrauschten Daten und Ausreißern bilden kann. Eine andere Variante ist die gewichtete Zufallsauswahl. Als Gewicht für ein Objekt x ist $\sum_{k=1}^{K} ||x - c_k||C_k(x)^m$ ein sinnvoller Wert. Wenn man auf diese Art und Weise eine Auswahl getroffen hat, funktioniert die weitere Wahl der besten $K + 1$ Partition genauso wie im $K - 1$ Fall. Es werden alle ausgewählten Daten als neues Clusterzentrum ausprobiert und anschließend das beste Clusterergebnis gewählt.

4.3.6 Evaluation

Die Evaluation teilt sich in zwei Abschnitte auf: Zunächst wird die Eignung des Verfahrens für die Bestimmung der Clusteranzahl im statischen Fall untersucht, anschließend für eine dynamische Anwendung mit sich verändernder Clusteranzahl.

Statisches Experiment Für den statischen Fall wird als Vergleich eine der am häufigsten verwendeten Strategien gewählt. Zur Vermeidung von lokalen Minima wird dabei der Fuzzy-C-Means-Algorithmus 10 Mal ausgeführt. Das gemäß der Zielfunktion beste Ergebnis wird am Ende ausgewählt. Zur Bestimmung der Clusteranzahl wird dies dann für alle möglichen Clusteranzahlen wiederholt. Die Gütefunktion bestimmt anschließend die beste Clusterstruktur.

Die verwendeten Datensätze sind künstlich erzeugte Daten, die sehr einfache Clusterstrukturen enthalten. Die Clusteranzahl wird zwischen 2 und 19 festgelegt. Für jeden einzelnen Test wird zufällig ein Datensatz mit 1000 Daten erzeugt. Auf diesen werden dann der vorgestellte Ansatz und die Standard-Strategie angewendet. Für die Erzeugung der Datensätze wird zunächst die Clusteranzahl festgelegt. Innerhalb des Datenraums ($[0, 200]^d$) mit Dimension d werden zufällig Clusterzentren für jedes Cluster bestimmt, wobei darauf geachtet wird, dass der Abstand zweier Clusterzentren jeweils mindestens der 6fachen Standardabweichung der Cluster entspricht. Danach werden die 1000 Datenpunkte auf alle Cluster gleichverteilt bestimmt. Der Abstand zum jeweiligen Clusterzentrum ist normalverteilt mit Standardabweichung 5. Zusätzlich wird der Standard-Fuzzy-C-Means Algorithmus mit der bei der Generierung verwendeten Clusterstruktur initialisiert und ausgeführt. Dieses Ergebnis dient als Vergleichsresultat und soll von den anderen Verfahren erreicht oder überboten werden. Am Ende wird die Gütefunktion für alle Ergebnisse aller Verfahren ausgewertet und jeweils gezählt, wie häufig das Ergebnis der Verfahren mindestens so gut war wie das Vergleichsresultat. Zusätzlich werden auch die Laufzeiten der einzelnen Varianten gemessen. Das Experiment wird für die Dimensionszahlen $2, 5, 10$ und 25 und für jede Clusteranzahl mit je 50 generierten Datensätzen

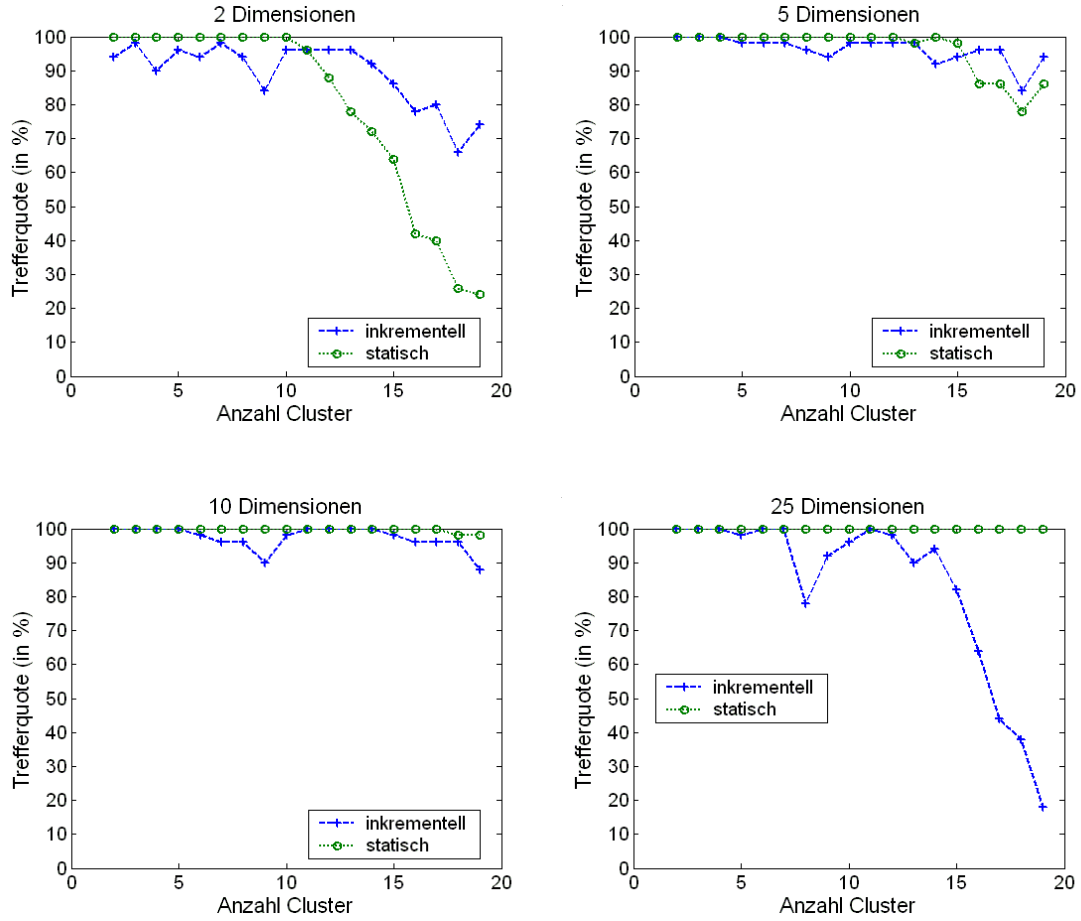

Abbildung 4.4: Vergleich der Trefferquote

durchgeführt. Desweiteren wird das Experiment zusätzlich mit dem Xie-Beni-Index ausgeführt, um ihn mit der neuen Gütefunktion vergleichen zu können.

Bei der Trefferquote war zu erwarten, dass die Standard-Strategie besser ist, da alle in Frage kommenden Clusteranzahlen ausprobiert werden. Die Ergebnisse zur Trefferquote in Abbildung 4.4 zeigen allerdings, dass der neue inkrementelle Ansatz häufig durchaus mithalten kann und vereinzelt sogar bessere Ergebnisse liefert. Dies kann nur daran liegen, dass in diesen Fällen die Wahrscheinlichkeit, mit Fuzzy-C-Means in einem lokalen Minimum zu landen, relativ hoch sein muss. Unser inkrementelles Verfahren kann dagegen durch Löschen und Hinzufügen von Clustern diese Minimas auflösen, während der Standard-Ansatz mit 10-maliger Neuinitialisierung oft nicht ein einziges Mal das globale Minimum erreicht. Dagegen scheint unser Ansatz besonders dann Probleme zu haben, wenn bei Daten höherer Dimension die wahre Clusteranzahl deutlich größer als die initi-

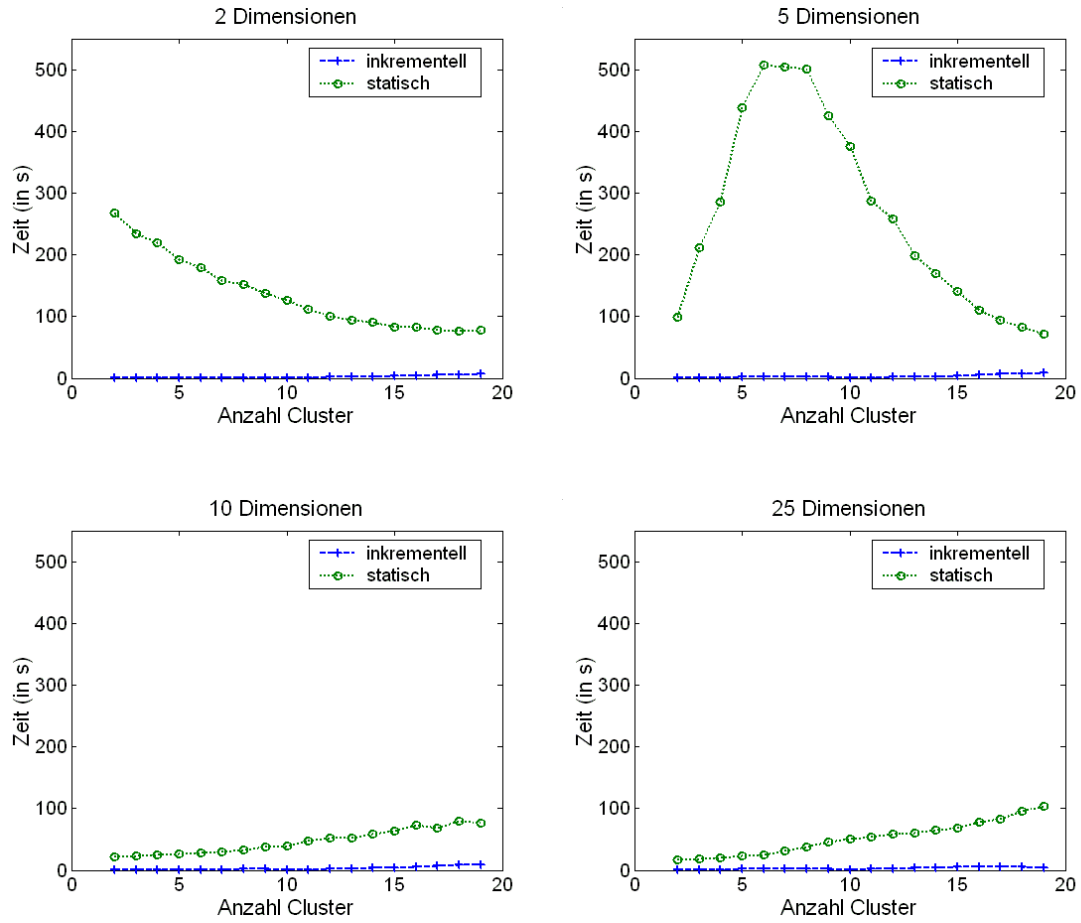

Abbildung 4.5: Vergleich der Laufzeiten

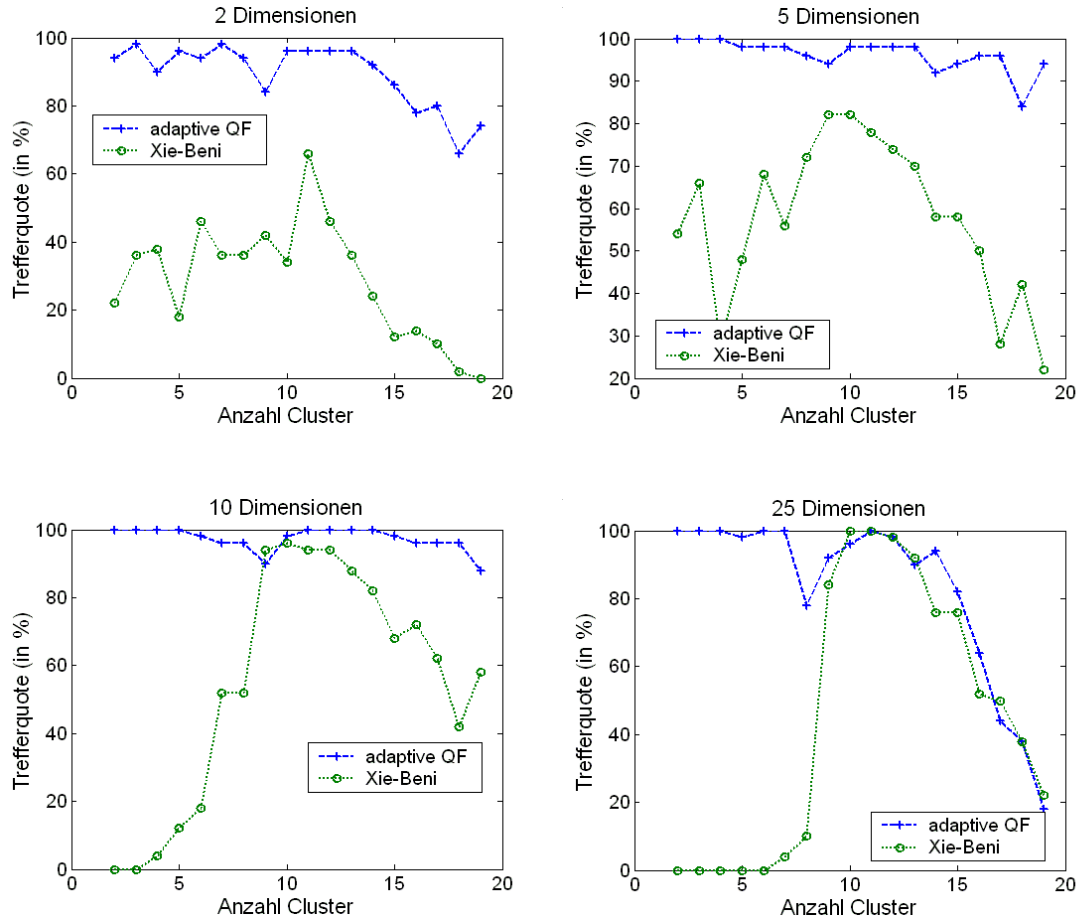

Abbildung 4.6: Vergleich des Xie-Beni-Index mit der neuen Gütefunktion (adaptive QF)

ale Clusteranzahl ist.

Die Laufzeiten sind, wie in Abbildung 4.5 zu erkennen ist, allerdings sehr unterschiedlich. Während unser Ansatz unabhängig von der Anzahl der Dimensionen und leicht steigend mit der Anzahl der Cluster auf niedrigem Niveau bleibt, sind die Laufzeiten mit dem Standard-Verfahren deutlich höher. Besonders hoch sind sie für niedrige Dimensionen und niedrige bis mittlere Clusteranzahl. Trotzdem entwickeln sich die Laufzeiten des Standard-Verfahrens für 2 und 5 Dimensionen deutlich anders als erwartet. Die hohen Laufzeiten bei 2 Dimensionen und wenigen Clustern lassen sich damit erklären, dass K-Means sehr langsam konvergiert, wenn bei wenigen Dimensionen das Clusterverfahren mit deutlich zu hoher Clusteranzahl angewendet wird. Für dem dreieckigen Verlauf bei 5 Dimensionen wurde noch keine plausible Erklärung gefunden, vermutlich handelt es sich um eine Mischung verschiedener Effekte.

In Abbildung 4.6 ist der Vergleich mit dem Xie-Beni-Index dargestellt. Wie zu erwarten ist der Xie-Beni-Index besonders dann schlechter, wenn die tatsächliche Clusteranzahl von der initialen Clusteranzahl 10 weiter entfernt ist. Bei wenigen Dimensionen spielt zusätzlich noch die hohe Wahrscheinlichkeit, dass Fuzzy-C-Means nur ein lokales Minimum findet, eine entscheidende Rolle. Auch wenn die tatsächliche Clusteranzahl nah an der initialen liegt, werden lokale Minima doch deutlich seltener entdeckt.

Adaptives Experiment Beim adaptiven Experiment geht es um die Frage, ob das Hinzufügen eines neuen Clusters oder das Löschen eines Clusters erkannt wird. Als Grundlage dienen wiederum die oben beschriebenen synthetischen Daten. Für eine Clusteranzahl $c \in [2, 18]$ wird zunächst wie im vorherigen Experiment ein Datensatz mit c Clustern erzeugt. Anschließend werden die Daten auf zwei Varianten erweitert. Im ersten Fall werden vom letzten Cluster noch einmal Daten erzeugt, im zweiten Fall wird noch ein weiteres Cluster hinzugefügt. Nun hat man zwei Datensätze mit gleich vielen Daten, wobei der erste c Cluster und der zweite $c + 1$ Cluster beinhaltet. Für den Übergang von c auf $c + 1$ Cluster wird zunächst Fuzzy-C-Means mit dem ersten Datensatz und den ursprünglichen c Clustern als Initialisierung angewendet. Nun werden die Daten durch den zweiten Datensatz aktualisiert und das Updateverfahren ausgeführt. Beim Übergang von $c + 1$ auf c Cluster wird das Verfahren mit vertauschten Datensätzen angewendet. Gemessen wird der Anteil der richtig erkannten Änderungen. Dieses Experiment wurde für unterschiedliche Dimensionen und jede Clusterzahl c jeweils 200mal durchgeführt.

Die Ergebnisse aus Tabelle 4.1 zeigen sehr gute Quoten für die Erkennung einer Änderung der Clusteranzahl. Wie zu erwarten war, ist dabei die Quote bei Verringerung der Clusteranzahl besser, da hierbei alle Cluster testweise gelöscht werden. Bei der Erhöhung der Clusteranzahl werden nur 10 Datenpunkte zufällig ausgewählt und als neue Clusterzentren getestet, wodurch es wahrscheinlicher wird, die Änderung nicht zu erkennen. Die höhere Quote bei höheren Dimensionen hängt mit der schon im ersten Experiment fest-

Dimensionen	$c+1 \rightarrow c$	$c \rightarrow c+1$	Gesamtquote
2	1.0	0.987	0.993
5	0.999	0.989	0.994
10	1.0	0.999	1.0
25	1.0	0.999	1.0

Tabelle 4.1: Quoten bei dynamischer Erhöhung und Verringerung der Clusteranzahl.

gestellten geringeren Wahrscheinlichkeit von lokalen Minima zusammen. Dieses Experiment zeigt, dass das Verfahren gerade in Umgebungen, bei denen sich die Clusteranzahl nicht abrupt um größere Sprünge ändert, sehr gut geeignet ist.

4.4 Clusteranalyse von parallelen Datenströmen

Wenn man die Clusteranalyse von Datenströmen betrachtet, stellt sich zunächst die Frage, wie man eine Ähnlichkeit bzw. einen Abstand zwischen Datenströmen sinnvoll berechnen kann. Dazu macht es Sinn, sich zu überlegen, was der Grund sein soll, dass zwei Datenströme zu einem Cluster gehören. Bei dem hier beschriebenen Ansatz steht die Entwicklung der einzelnen Datenströme im Vordergrund. Zwei Datenströme sollen ähnlich sein, wenn sie sich ähnlich entwickeln, also ihr aktueller Verlauf sich ähnelt. Der aktuelle Verlauf wird durch die letzten Werte beschrieben, weshalb es sinnvoll ist, ein aktuelles Sliding Window des Datenstroms zu betrachten. Da die Werte, je älter sie werden, immer weniger über den aktuellen Verlauf aussagen, ist zusätzlich eine Gewichtung in Abhängigkeit von der Zeit sinnvoll.

Da wir ein einfaches Abstandsmaß benötigen, bietet sich der euklidische Abstand der gewichteten Zeitfenster an. Dieses Maß erfüllt die oben benannten Eigenschaften und hat eine enge Beziehung zur statistischen Korrelation zwischen zwei Strömen. Genauer gesagt, es gibt einen einfachen linearen Zusammenhang des euklidischen Abstands zur Korrelation von normalisierten Zeitreihen mit Mittelwert 0 und Standardabweichung 1. Je nach Anwendung können auch andere Maße besser geeignet sein, aber der euklidische Abstand ist als sehr verbreitetes Maß eine gute Wahl. Außerdem ist das eigentliche adaptive Clusterverfahren unabhängig vom gewählten Abstandsmaß, wodurch auch andere Maße verwendet werden können. Wie das Verfahren dazu angepasst werden muss, wird in Abschnitt 4.6.2 erklärt. In der folgenden Vorstellung des Verfahrens wird aber nur der euklidische Abstand benutzt.

4.4.1 Vorverarbeitung

Bei der Verarbeitung von Datenströmen ist jedes Mal, wenn neue Daten auftauchen, ein Update notwendig. Da die Frequenz der Daten aber durchaus sehr hoch sein kann und ein Updateschritt in der hier betrachteten Anwendung sicherlich nicht ganz ohne Aufwand ablaufen kann, ist es sinnvoll, das Hilfsmittel der blockweisen Verarbeitung (siehe Abschnitt 2.4.2) der Daten zu nutzen. Die Hauptaufgabe der Vorverarbeitung ist es, eine effiziente Abstandsberechnung für die Clusteranalyse zu ermöglichen. Zunächst gehört dazu, die Daten des aktuellen Sliding Windows zu gewichten und zu normalisieren. Da die Dimensionalität der Abstandsberechnung der Größe des Sliding Window entspricht, kann diese extrem groß werden, womit auch die Abstandsberechnung sehr aufwändig wird. Aus diesem Grund macht eine vorherige Komprimierung der Daten Sinn, wobei eine Komprimierung mit der Diskreten Fourier Transformation (DFT) viele Vorteile hat. In den nächsten Abschnitten wird zunächst die blockweise Verarbeitung mit Gewichtung, die Normalisierung und schließlich die DFT-Komprimierung behandelt.

4.4.2 Zeitfenster, Gewichtung und Blockverarbeitung

Ein Sliding Window eines Datenstroms der Größe w beinhaltet die letzten w Daten des Datenstroms (siehe Abschnitt 2.4.2). Wenn ein neues Element erscheint, wird das bisher älteste Element gelöscht und das neue Element eingefügt. Ein Sliding Window kann auch als w dimensionaler Vektor X betrachtet werden mit $X = (x_0, x_1, \ldots, x_{w-1})$. Für eine effizientere Verarbeitung wird das Sliding Window in Blöcken aktualisiert. Hierbei beinhaltet ein Block eine Anzahl von v Elementen, womit das Sliding Window aus $m = \frac{w}{v}$ Blöcken besteht. Das aufgeteilte Fenster sieht dann wie folgt aus:

$$X = (\underbrace{x_0, x_1, \ldots, x_{v-1}}_{B_1} | \underbrace{x_v, x_{v+1}, \ldots, x_{2v-1}}_{B_2} | \ldots | \underbrace{x_{(m-1)v}, x_{(m-1)v+1}, \ldots, x_{w-1}}_{B_m}).$$

Die Datenströme werden blockweise aktualisiert, also jedesmal, wenn ein neuer Block aus v neuen Elementen vorhanden ist. Dieser Ansatz steigert die Effizienz, da die Anzahl der Updates um einen Faktor v reduziert werden. Allerdings sind die Clusterstrukturen damit nicht immer aktuell. Da die Verzögerung aber nur maximal einen Block betragen kann, ist dieser Nachteil durch die Wahl kleiner Blöcke begrenzt. Desweiteren wird angenommen, dass ein Datenstrom sich nur langsam und nicht abrupt ändert, sich zwei nacheinander eintreffende Werte eines Stromes also nur gering unterscheiden. Dadurch ändern sich die Clusterstrukturen ebenfalls langsam und die Verzögerung einer blockweisen Verarbeitung ist unproblematisch.

Es wird vorausgesetzt, dass die Datenelemente synchron erscheinen, jeder Datenstrom also zu den selben Zeitpunkten neue Daten liefert. Falls dies nicht gegeben ist, kann dies

Symbol	Bedeutung
X	Datenstrom
X^N	normalisierter Datenstrom
x_i	$(i+1)$.tes Element im aktuellen Zeitfenster
x_{ij}	$(j+1)$.tes Element von Block B_i
w	Größe des Sliding Windows
v	Größe eines Blocks
m	Anzahl von Blöcken in einem Fenster ($w = mv$)
c	Konstante für die Gewichtung
V	Gewichtsvektor
$\overline{x}$	Mittelwert eines Datenstroms
s	Standardabweichung eines Datenstroms

Tabelle 4.2: Clustering: verwendete Notationen

aber jederzeit über Interpolation simuliert werden. Das Update des Vektors X für das Sliding Window eines Datenstroms durch einen neuen Block erfolgt durch eine Shift Operation:

$$\begin{array}{lllll} X^{old}: & B_1| & B_2|B_3|\ldots|B_{m-1}|B_m & \\ X^{new}: & & B_2|B_3|\ldots|B_{m-1}|B_m & |B_{m+1} \end{array}$$

Um auch innerhalb des zu betrachtenden Zeitfensters den aktuelleren Daten eine höhere Bedeutung zukommen zu lassen, erhält jedes Element ein Gewicht. Jedesmal, wenn ein neues Element eingefügt wird, werden die Gewichte der anderen Elemente erniedrigt. Das Gewicht eines Elementes x_i wird definiert mit c^{w-i-1}, wobei $0 < c \leq 1$ eine Konstante ist. Mit V wird der gesamte Gewichtsvektor bezeichnet $V = (c^{w-1}, c^{w-2}, \ldots, c^1, c^0)$ und das punktweise Produkt von V und X mit $V \odot X$:

$$V \odot X = (c^{w-1}x_0, c^{w-2}x_2, \ldots, c^0 x_{w-1})$$

4.4.3 Normalisierung

Da der relative Verlauf eines Datenstroms von Interesse ist, muss der originale Datenstrom in einem ersten Schritt normalisiert werden. Durch eine lineare Transformation wird der Datenvektor auf Mittelwert 0 und Standardabweichung 1 normalisiert. Dies geschieht, indem man jedes Element wie folgt transformiert:

$$x_i^N = \frac{x_i - \overline{x}}{s},$$

wobei $\overline{x}$ der Mittelwert und s die Standardabweichung des originalen Datenstroms auf dem aktuellen Zeitfenster sind. Aufgrund der Gewichtung muss hierbei der gewichtete

Mittelwert und die gewichtete Standardabweichung benutzt werden, die sich wie folgt berechnen:

$$\begin{aligned}\overline{x} &= \frac{1-c}{1-c^w}\sum_{i=0}^{w-1} x_i c^{w-i-1},\\ s^2 &= \frac{1-c}{1-c^w}\sum_{i=0}^{w-1}(x_i-\overline{x})^2 c^{w-i-1}\\ &= \frac{1-c}{1-c^w}\sum_{i=0}^{w-1} x_i^2 c^{w-i-1} - \overline{x}^2.\end{aligned}$$

Wegen der blockweisen Verarbeitung müssen auch $\overline{x}$ und s blockweise aktualisiert werden. Für ein Zeitfenster X, das in die Blöcke $B_1, \ldots, B_m$ eingeteilt ist, sei x_{ij} das $(j+1)$-te Element des i-ten Blocks. Der erste Block ist somit $B_1 = (x_{10}, x_{11}, \ldots, x_{1(v-1)})$. Der bei einem Update neu hinzukommende Block sei B_{m+1}. Zur Berechnung von Mittelwert und Standardabweichung werden zunächst folgende Werte ermittelt:

$$S = \sum_{i=0}^{w-1} x_i c^{w-i-1}, Q = \sum_{i=0}^{w-1} x_i^2 c^{w-i-1}$$

Für ein effizienteres Update werden dazu für jeden Block B_k folgende Werte S_k und Q_k ermittelt:

$$S_k = \sum_{i=0}^{w-1} x_{ki} c^{w-i-1}, Q_k = \sum_{i=0}^{w-1} x_{ki}^2 c^{w-i-1}.$$

Wenn das Zeitfenster durch Verschieben eines neuen Blocks aktualisiert wird, können S und Q wie folgt angepasst werden:

$$\begin{aligned}S &\leftarrow S \cdot c^v - S_k \cdot c^w + S_{m+1}\\ Q &\leftarrow Q \cdot c^v - Q_k \cdot c^w + Q_{m+1}\end{aligned}$$

Damit ergibt sich für den Mittelwert und die Standardabweichung:

$$\begin{aligned}\overline{x} &= \frac{1-c}{1-c^w} S\\ s^2 &= \frac{1-c}{1-c^w} Q - \overline{x}^2.\end{aligned}$$

4.4.4 Diskrete Fourier Transformation

Die Dimensionalität des Zeitfensters ist w, was durchaus ein sehr großer Wert sein kann. Wenn man sich einen Datenstrom vorstellt, der jede Sekunde einen Wert liefert, und man dessen Entwicklung über einen Tag betrachten will, ergibt sich ein Datenvektor für das Zeitfenster mit der Dimensionalität $w = 86400$. Die Berechnung des Abstandes zwischen zwei Zeitfenstern ist somit sehr aufwändig. Die Abstandsfunktion innerhalb des Cluster-Algorithmus wird sehr häufig aufgerufen, so dass eine Optimierung dieser Funktion angebracht ist. Eine gute Möglichkeit dabei ist, die Dimensionalität mittels Komprimierung zu verringern. Eine gängige und effiziente Methode ist die Komprimierung der Daten mit der Diskreten Fourier Transformation (DFT). Wie später noch genauer beschrieben wird, ist die Fouriertransformation eine gute Basis für eine approximierte euklidische Abstandsberechnung und erlaubt zusätzlich noch eine Eliminierung von Rauschen innerhalb des Datenverlaufs. Die DFT einer Sequenz $X = (x_0, \ldots, x_{w-1})$ ist definiert durch die DFT-Koeffizienten

$$\mathrm{DFT_f(X)} = \frac{1}{\sqrt{\mathrm{w}}} \sum_{\mathrm{j=0}}^{\mathrm{w-1}} \mathrm{x_j e}^{\frac{-\mathrm{i}2\pi\mathrm{fj}}{\mathrm{w}}}, \;\; 0 \leq \mathrm{f} < \mathrm{w},$$

wobei $i = \sqrt{-1}$ ist.

Nun soll die DFT des normalisierten Zeitfensters X^N aus den Ausgangsdaten X und dem Gewichtungsvektor V berechnet werden. Die einzelnen normalisierten und gewichteten Werte lassen sich berechnen durch

$$c^{w-i-1} x_i^N = c^{w-i-1} \frac{x_i - \overline{x}}{s}.$$

Die DFT ist linear, es gilt also für alle $\alpha, \beta \geq 0$ und beliebige Vektoren X, Y, dass

$$\mathrm{DFT}(\alpha \mathrm{X} + \beta \mathrm{Y}) = \alpha \mathrm{DFT(X)} + \beta \mathrm{DFT(Y)}$$

Für die Berechnung des gewichteten und normalisierten Zeitfensters ergibt sich damit ($V \odot X$ ist das punktweise Produkt von V und X):

$$\begin{aligned}
\mathrm{DFT}(\mathrm{V} \odot \mathrm{X}) &= \mathrm{DFT}\left(\mathrm{V} \odot \frac{\mathrm{X} - \overline{\mathrm{x}}}{\mathrm{s}}\right) \\
&= \mathrm{DFT}\left(\frac{\mathrm{V} \odot \mathrm{X} - \mathrm{V}\overline{\mathrm{x}}}{\mathrm{s}}\right) \\
&= \frac{\mathrm{DFT}(\mathrm{V} \odot \mathrm{X}) - \mathrm{DFT(V)}\overline{\mathrm{x}}}{s}
\end{aligned}$$

Da der Gewichtsvektor V konstant ist, kann der Vektor $DFT(V)$ im Voraus berechnet werden. Eine inkrementelle Berechnung ist nur für $\mathrm{DFT}(\mathrm{V} \odot \mathrm{X})$ notwendig. Bei einem Update verfällt der Block $B_1 = (x_{10}, \ldots, x_{1,v-1})$ und der Block

$B_{m+1} = (x_{m+1,0}, \ldots, x_{m+1,v-1})$ kommt neu hinzu. Mit $X^{old} = B_1|\cdots|B_m$ und $X^{new} = B_2|\cdots|B_{m+1}$ seien die Vektoren des alten und des neuen Zeitfensters bezeichnet. Ohne Gewichtung lässt sich die DFT dann wie folgt inkrementell aktualisieren [ZS02]:

$$\mathrm{DFT_f} \leftarrow \mathrm{e}^{\frac{\mathrm{i2\pi fv}}{\mathrm{w}}}\mathrm{DFT_f} + \frac{1}{\sqrt{\mathrm{w}}}\left(\sum_{\mathrm{j=0}}^{\mathrm{v-1}} \mathrm{e}^{\frac{\mathrm{i2\pi f(v-j)}}{\mathrm{w}}}\mathrm{x_{m+1,j}} - \sum_{\mathrm{j=0}}^{\mathrm{v-1}} \mathrm{e}^{\frac{\mathrm{i2\pi f(v-j)}}{\mathrm{w}}}\mathrm{x_{1,j}}\right).$$

Bei Hinzunahme der Gewichtung muss auch das Gewicht jedes Elements aktualisiert werden. Dies geschieht durch Multiplikation jeden Elements x_i mit dem Faktor c^v. Die Elemente des neuen Blocks B_{m+1} bekommen die Gewichte $c^{v-j-1}, 0 \leq j < v$. Wegen der Linearität der DFT gilt $\mathrm{DFT_f(c^v X^{old})} = \mathrm{c^v DFT_f(X^{old})}$. Die Koeffizienten können somit nun wie folgt aktualisiert werden:

$$\mathrm{DFT_f} \leftarrow \mathrm{e}^{\frac{\mathrm{i2\pi fv}}{\mathrm{w}}}\mathrm{c^v DFT_f} + \frac{1}{\sqrt{\mathrm{w}}}\left(\sum_{\mathrm{j=0}}^{\mathrm{v-1}} \mathrm{e}^{\frac{\mathrm{i2\pi f(v-j)}}{\mathrm{w}}}\mathrm{c^{v-j-1}x_{m+1,j}} - \sum_{\mathrm{j=0}}^{\mathrm{v-1}} \mathrm{e}^{\frac{\mathrm{i2\pi f(v-j)}}{\mathrm{w}}}\mathrm{c^{w+v-j-1}x_{1,j}}\right).$$

Wenn man nun noch für jeden Block B_k die Werte

$$\beta_k^f = \sum_{j=0}^{v-1} e^{\frac{i2\pi f(v-j)}{w}} c^{v-j-1} x_{k,j}$$

einfügt, vereinfacht sich die DFT-Update-Formel zu

$$\mathrm{DFT_f} \leftarrow \mathrm{e}^{\frac{\mathrm{i2\pi fv}}{\mathrm{w}}}\mathrm{c^v DFT_f} + \frac{1}{\sqrt{\mathrm{w}}}\left(\beta^{\mathrm{f}}_{\mathrm{m+1}} - \mathrm{c^w}\beta^{\mathrm{f}}_{\mathrm{1}}\right).$$

Beim Update eines neuen Blockes müssen also zunächst die Summen S und Q für den neuen Block berechnet werden. Aus diesen und den Summen des nun aus dem Sliding Window zu entfernenden Blocks können die Summen und damit Mittelwert und Standardabweichung auf einfache Weise aktualisiert werden. Desweiteren müssen für den neuen Block die β-Koeffizienten berechnet werden. Aus diesen und den β-Koeffizienten des zu entfernenden Blocks können nun mit Hilfe des Mittelwerts und der Standardabweichung die DFT-Koeffizienten mit der DFT-Update-Formel effizient aktualisiert werden. Für das Update werden also für jeden Block B_k nur die Summen S_k und Q_k sowie die β-Koeffizienten benötigt. Die eigentlichen Daten sind nicht mehr notwendig und müssen nicht gespeichert werden, was einen erheblichen Speicherplatzgewinn zur Folge hat.

Nun können die DFT Koeffizienten effizient und inkrementell berechnet werden, aber damit alleine haben wir noch keinen zeitlichen Gewinn für die Distanzberechnung des euklidischen Abstands. Dieser ist für zwei Vektoren X, Y definiert als:

$$||X - Y|| = \left(\sum_{i=0}^{w-1}(x_i - y_i)^2\right).$$

Desweiteren ist die Eigenschaft hilfreich, dass die DFT den euklidischen Abstand erhält, es gilt also für zwei Vektoren X, Y:

$$||X - Y|| = ||\mathrm{DFT(X)} - \mathrm{DFT(Y)}||.$$

Für die DFT-Koeffizienten gilt außerdem, dass DFT_0 dem Mittelwert (also 0, da normiert) und die ersten folgenden Koeffizienten den niedrigeren Frequenzanteilen entsprechen, die auch den grundlegenden Verlauf bestimmen und damit die meiste Information enthalten. Höhere Frequenzen treten dagegen weniger stark auf und beinhalten häufig auch Rauschen, weshalb die Information der entsprechenden Koeffizienten weniger oder sogar fehlerhafte Informationen enthalten. Desweiteren sind die DFT-Koeffizienten für reelle Zahlen antisymmetrisch und es gilt $\mathrm{DFT_f} = \overline{\mathrm{DFT_{w-f}}}, 0 < \mathrm{f} < \mathrm{w}$, wobei $\overline{\mathrm{DFT_{w-f}}}$ die komplex konjugierte von $\mathrm{DFT_{w-f}}$ ist. Wenn man nun die weniger wichtigen Koeffizienten in der Mitte weg lässt (also 0 setzt), dann entspricht dies einem Tiefpassfilter, womit eine Glättung des Funktionsverlaufs erreicht wird. Unter dem Gesichtspunkt, dass dadurch Rauschen unterdrückt wird, hat dies sogar einen positiven Nutzen. Auf jeden Fall lassen sich aber durch Weglassen dieser Koeffizienten viele Dimensionen und damit Speicherplatz und Rechenzeit einsparen. Es werden also nur die ersten $u \ll w$ Koeffizienten gespeichert und genutzt. Genau genommen sind dies $DFT_1, DFT_2, \ldots, DFT_u$. Den euklidischen Abstand kann man somit approximativ berechnen durch

$$||X - Y|| = \sqrt{2 \sum_{f=1}^{u} (DFT_f(X) - DFT_f(Y)) \overline{DFT_f(X) - DFT_f(Y)}}$$

Aufgrund der Verwendung von nur u Koeffizienten wird die Abstandsberechnung deutlich effizienter und zugleich sinkt der Speicherbedarf. Die Zeitkomplexität ist asymptotisch mit $O(nvu)$ und der Speicherbedarf durch $O(nmu+nv)$ begrenzt. Nachdem eine effiziente Abstandsfunktion vorhanden ist, kann nun zum eigentlichen Clustern übergegangen werden.

4.4.5 Fuzzy-Clustering von Datenströmen

Das Clustern von statischen Daten mit K-Means und Fuzzy-C-Means wurde bereits am Anfang dieses Kapitels vorgestellt. Als Vorbereitung auf die Online-Umgebung wurde dann der Fall einer adaptiven Anpassung der Clusteranzahl vorgestellt, die sich besonders für sich inkrementell entwickelnde Clusterstrukturen eignen. Nachdem mit dem letzten Abschnitt nun auch eine effiziente Berechnung der Abstandsfunktion möglich ist, gilt es nun, diese Bausteine zu einem kompletten Clusteralgorithmus für Datenströme zusammenzusetzen.

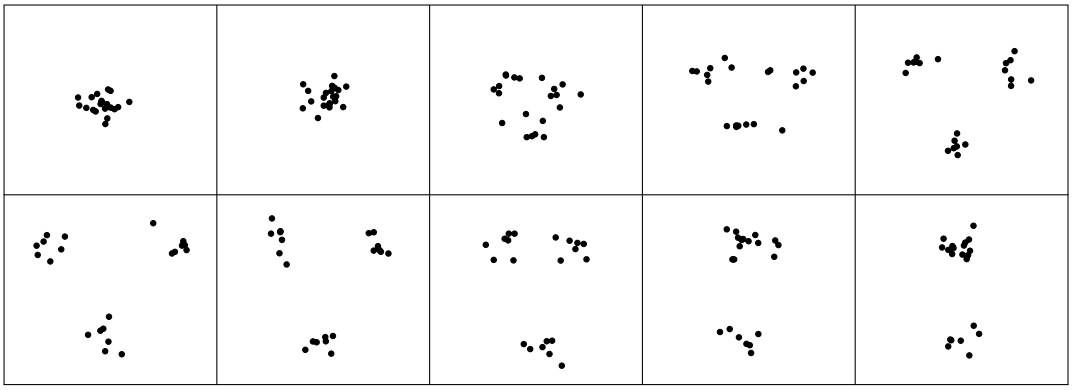

Abbildung 4.7: Kontinuierlich verändernde Datenmenge
Aus einem Cluster entwickeln sich zunächst drei Teilcluster, von denen schließlich zwei wieder verschmelzen (von oben links nach unten rechts).

K-Means erscheint konzeptionell geeignet für ein Online-Szenario. Da K-Means von Natur aus ein iteratives Verfahren ist, lässt es sich auf einfache Weise an eine adaptive Umgebung anpassen. Die Fuzzy-Variante ist aufgrund der Zugehörigkeitsgrade, die auch kleinere Änderungen der Datenströme widerspiegeln können, besonders geeignet. Schließlich verändern sich die Datenströme kontinuierlich, was sich in kontinuierlich verändernden Zugehörigkeitsgrade wiederspiegelt. Beim K-Means-Verfahren wären dagegen nur plötzliche Änderungen der Clusterzugehörigkeit erkennbar. Zur Illustration hierzu dient Abbildung 4.7, in der sich eine Datenpunktmenge kontinuierlich verändert.

Der Pseudocode der Online-Variante von Fuzzy-C-Means ist in Algorithmus 4.5 dargestellt. Der Standard-Fuzzy-C-Means-Algorithmus wird jeweils auf den aktuellen Zeitfenstern der Datenströme angewendet bis ein neuer Block erscheint. Dann wird ein Update-Shift ausgeführt, um die Zeitfenster der Datenströme und die Distanzfunktion zu aktualisieren. Mit den neuen Daten werden dann die Clusterzentren aktualisiert, um anschließend mit dem Standard-Fuzzy-C-Means-Algorithmus fortzufahren. Die letzte Clusterstruktur wird also direkt als Initialisierung zum Clustern der aktualisierten Daten genutzt. Da eine kontinuierliche Entwicklung angenommen wird, ist dies sicherlich eine gute Initialisierung. Die Clusterstruktur kann sich normalerweise schon deshalb nicht allzu sehr verändert haben, da nur ein Block des Zeitfensters ausgetauscht wurde.

Die Clusteranzahl wird vor jedem Update aktualisiert. Aufgrund der Annahme einer kontinuierlichen Entwicklung und aus Effizienzgründen wird die Clusteranzahl nur um maximal 1 verändert. Die Auswahl der Clusteranzahl geschieht gemäß dem in Abschnitt 4.3 beschriebenen Verfahren. Es werden verschiedene $K + 1$ und $K - 1$ Partitionen berechnet und die der vorgestellten Gütefunktion beste gewählt. Das Problem, dass aufgrund lokaler Minima im Gütemaß die richtige Clusteranzahl nicht immer gefunden wird, ist im Online-Szenario unproblematischer. Nach jedem Update verändern sich die Daten. Auch wenn sich die optimale Clusterstruktur nur gering ändert, ändert sich auch der Verlauf

Algorithmus 4.5 : Datenströme iterativ Clustern

Input : Strom von Trainingsdaten
Output : Strom von Clusterstrukturen

Initialisiere die K Clusterzentren zufällig;
while *Ströme liefern noch Daten* **do**
  repeat
    Berechne für jeden Strom die Zugehörigkeiten zu allen Cluster;
    Berechne die Mittelpunkte der neuen Cluster;
  until *neuer Block bereit*;
  Ermittle beste Clusteranzahl aus {K-1, K, K+1 };
  Output : aktuelle Clusterstruktur
  Aktualisiere Datenströme, Distanzen und Clusterzentren;
end

der Gütefunktion. Lokale Minima können verschwinden oder sich verschieben und die Möglichkeit, die global optimale Struktur zu finden, ist bei jedem Update neu gegeben.

4.4.6 Visualisierung und Weiterverarbeitung

Die Präsentation von Stream-Clusterergebnissen ist deutlich aufwändiger als von statischen Clusterergebnissen, da sich die Clusterstrukturen regelmäßig verändern. Meist ist kaum genügend Zeit vorhanden, die Ergebnisse von Hand auszuwerten und darauf rechtzeitig zu reagieren, weshalb sich im Online-Setting eine automatisierte und computerunterstützte Analyse der Clusterstrukturen anbietet. Trotzdem kann es natürlich auch interessant sein, zumindest zusätzlich die Entwicklung der Clusterstrukturen direkt zu verfolgen, wofür hier zwei verschiedene Möglichkeiten zur Visualisierung vorgestellt werden. Während die erste Variante versucht, nur die aktuelle Clusterstruktur möglichst detailliert und übersichtlich darzustellen, liegt der Fokus bei der zweiten Variante auf der Darstellung der zeitlichen Entwicklung der Zugehörigkeiten der einzelnen Ströme. Je nach Anwendung sind natürlich weitere Visualisierungsvarianten möglich und von Interesse.

Darstellung der aktuellen Clusterstruktur Die Darstellung der aktuellen Clusterstruktur geschieht über eine Matrix. Die Zeilen entsprechen den Datenströmen und die Spalten den Clustern. Vor jeder Zeile ist der gewichtete Verlauf des entsprechenden Datenstroms abgebildet. Ebenso ist über jeder Spalte der mittlere Verlauf des Clusters dargestellt. Die Zellen der Matrix sind den Zugehörigkeitsgraden entsprechend eingefärbt. Dabei steht ein weißes Feld dafür, dass der Datenstrom der Zeile zu dem Cluster der Spalte eine Zugehörigkeit von 0 hat. Ein schwarzes Feld steht entsprechend für den Zu-

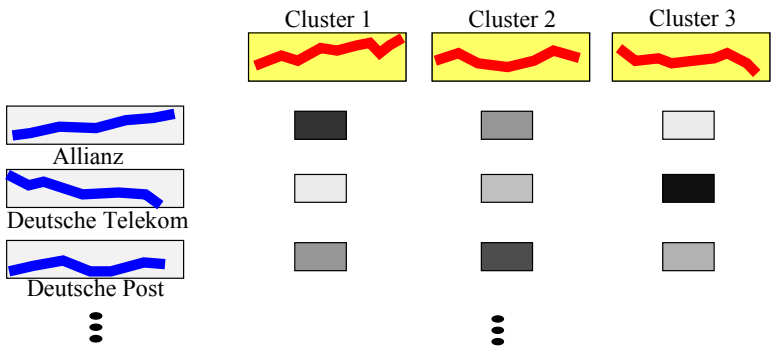

Abbildung 4.8: Beispiel: Darstellung der aktuellen Clusterstruktur

gehörigkeitsgrad 1 und dazwischenliegende Werte sind durch verschiedene Grauwerte dargestellt. Diese Darstellung passt sich bei jedem Update der aktuellen Clusterstruktur an. Da die Darstellung bei sehr vielen Datenströmen unübersichtlich werden kann, besteht auch die Möglichkeit, nur eine Teilmenge der Datenströme für die Darstellung auszuwählen. Ein Beispiel ist in Abbildung 4.8 dargestellt.

Darstellung des zeitlichen Strukturverlaufs Zur Darstellung des Strukturverlaufs kann der Übersichtlichkeit halber nicht die komplette Zugehörigkeitsmatrix für alle Ströme und Cluster über einen Zeitraum dargestellt werden. Deshalb beschränkt sich die Darstellung darauf, zu jedem Zeitpunkt für jeden Datenstrom nur das Cluster mit der größten Zugehörigkeit anzugeben. Die Darstellung erfolgt wieder durch eine Matrix. Die Zeilen entsprechen ebenso jeweils einem Datenstrom. Die Spalten allerdings repräsentieren jeweils einen Zeitpunkt. Die Felder stellen die Zugehörigkeit zu dem Cluster dar, zu dem der Strom die höchste Zugehörigkeit hat. Dabei sind unterschiedliche Cluster durch unterschiedliche Farben dargestellt. Der Grad der Zugehörigkeit ist dabei durch die Stärke des Farbtons wiedergegeben. Eine weitere Möglichkeit bestünde darin, die Farben der einzelnen Cluster je nach Zugehörigkeitsgrad zu mischen. Dies ist allerdings nur für sehr geringe Clusteranzahlen (3-6) möglich, und die gemischten grau-braunen Farben sind oft nur schwer zu interpretieren. Bei jedem Update kommt eine neue Spalte mit der aktuellen Struktur hinzu und die älteste Spalte verschwindet. Ein Beispiel ist in Abbildung 4.9 dargestellt.

4.4.7 Implementation

Die Grundlage für die Implementation dieses Datenstrom-Verfahrens bildet das PIPES Paket der Java Bibliothek XXL [BBD+01, KS04], das von der Datenbank-Arbeitsgruppe um Professor Seeger an der Philipps-Universität Marburg entwickelt wird. Im Zentrum

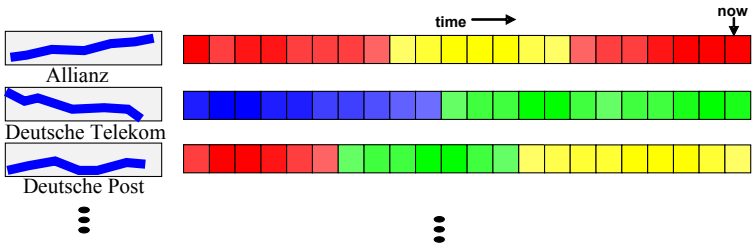

Abbildung 4.9: Beispiel: Darstellung des zeitlichen Strukturverlaufs

dieses Pakets steht die Bereitstellung einer Infrastruktur zur Verarbeitung von Datenströmen. Hierbei gibt es Grundgerüste für die Implementierung von Quellen, Senken und Operatoren von Datenströmen, die sich über Methoden zum An- und Abmelden beliebig kombinieren lassen. So lässt sich auf einfache Weise ein komplexer Anfragegraph auf unterschiedlichste Datenströme erzeugen. Zusätzlich stellt das Paket unterschiedliche Werkzeuge zur Steuerung und Verwaltung von einzelnen Prozessoren bis hin zum Scheduling einer Menge von Operatoren bereit.

Zunächst wurde in dieser Arbeit ein Paket mit den notwendigen Clusterverfahren implementiert. Existierende Implementierungen waren ungeeignet, da teilweise tiefgreifende Änderungen in Standard-Fuzzy-C-Means-Implementierungen notwendig gewesen wären. Wichtig hierbei war eine flexible Architektur, die es erlaubt, mit beliebigen Gütefunktionen, Initialisierungs- sowie Updatestrategien, Distanzmaßen und Objekten zu arbeiten. Eine zentrale Rolle hierbei spielt die Klasse FUZZYKMEANSSTATE, die zur Repräsentation einer kompletten Clusterstruktur mit Zugehörigkeitsgraden, Clusterzentren, Datenobjekten und verwendeten Funktionen (Distanzfunktion, Gütefunktion usw.) dient. Dadurch wird es möglich, dass der Datenstromoperator als Ergebnis einen Strom von Clusterstrukturen liefert. Das komplette Clustering-Paket ist sowohl für K-Means sowie Fuzzy-C-Means ausgelegt und beinhaltet auch zwei Erweiterungen für den Umgang mit Rauschen und Ausreißern. Desweiteren wurde in dieses Paket die in Abschnitt 4.3.2 beschriebene Anpassung der Clusteranzahl für statische Datensätze integriert.

Für das Datenstromverfahren wurden mehrere voneinander unabhängige Datenstromoperatoren implementiert. Zunächst war ein Operator notwendig, der zahlreiche Quellströme synchronisiert. Als nächstes folgte ein Operator, der die synchronisierten Ströme normalisiert, gewichtet und mit DFT komprimiert. Für die Evaluierung wurde dieser Operator zusätzlich ohne DFT-Komprimierung implementiert. Als letztes folgt der eigentliche Clusteroperator, der die vorverarbeiteten Ströme in Cluster aufteilt und die Clusterstrukturen als Datenstrom zur Verfügung stellt. Optional hat das Clusterverfahren Zugriff auf die Synchronisation, um zu erkennen, ob neue Daten vorhanden sind. Zur optimierten Speicherauslastung ist es möglich, Speicherplatz für die komprimierten Ströme über ein Objektpool zu verwalten. Der Aufbau dieser Operatoren ist in Abbildung 4.10 dargestellt.

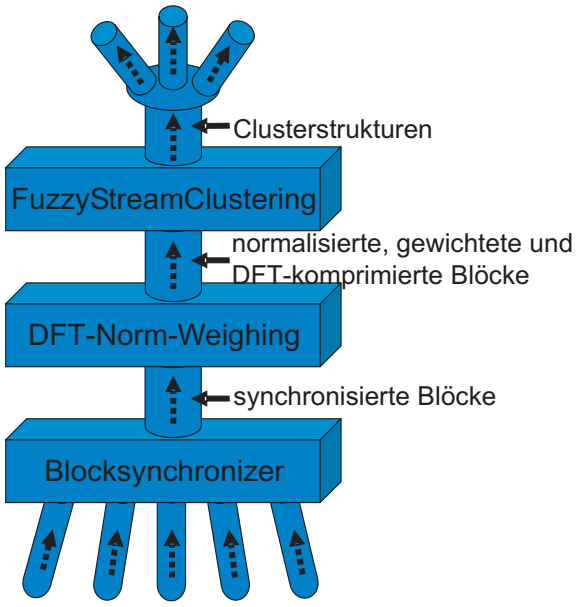

Abbildung 4.10: Operatorbaum für eine FCM-Stream Anfrage

Für die Evaluierung war die Implementation von zahlreichen weiteren Datenstromoperatoren notwendig. Dazu gehört die gleichzeitige Generierung von zahlreichen Datenströmen als Quellen, Speichern und Laden von Datenströmen als Senke und Quelle, Visualisierungen von Datenströmen und Clusterstrukturen als Senken sowie ein Operator zum Vergleich von Clusterstrukturen mithilfe eines Abstandsmaßes für Fuzzy-Clusterstrukturen.

4.5 Evaluation

Zur Evaluation des vorgestellten Clusterverfahrens auf Datenströmen wird die Anpassung der Clusteranzahl und der Clusterzugehörigkeitsgerade sowie die Performanz und Qualität mit Hilfe von synthetisch erzeugten Datenströmen untersucht. Zuvor wird das dazu verwendete relative Abstandsmaß für Fuzzy-Partitionen vorgestellt und der Generator für die synthetischen Datenströme beschrieben. Als letztes wird das Verfahren auf Real-World-Daten angewendet.

4.5.1 Relatives Abstandsmaß für Fuzzy-Partitionen

Ein wichtiger Bestandteil der Evaluation dieses Verfahrens wird es sein, den Einfluss der Komprimierung von Datenströmen auf einem Fenster auf das Clusterergebnis zu unter-

suchen. Dazu ist es notwendig, Clusterergebnisse mit und ohne Komprimierung vergleichen zu können. Der Vergleich von Fuzzy-Clusterstrukturen ist allerdings nicht trivial. Die absoluten Werte von Gütefunktionen sagen nichts über die Ähnlichkeit zweier Clusterstrukturen aus. Hierfür sind relative Evaluationsmaße notwendig. Nach [Bor05] gibt es dabei drei grundsätzlich verschiedene Vorgehensweisen. Die erste Möglichkeit ist, für jedes Cluster das ähnlichste Cluster in der anderen Clusterstruktur zu suchen und die einzelnen Clusterähnlichkeiten zu aggregieren. Der große Vorteil dieser Variante liegt in der Effizienz mit $O(nk)$, wobei n der Anzahl der Objekte und k der Clusteranzahl entspricht. Leider ist aber keine bijektive Abbildung der Cluster garantiert, womit das Maß auch nicht symmetrisch sein muss. Eine weitere Möglichkeit bestimmt zunächst eine bestmögliche bijektive Abbildung der Cluster beider Partitionen und bewertet dann diese Zuordnung. Vorteil hierbei ist die garantierte bijektive Abbildung, allerdings ist dies mit einem erhöhten Aufwand von $O(nk^3)$ verbunden. Außerdem müssen beide Partitionen die gleiche Clusteranzahl besitzen. Bei der dritten Variante betrachtet man zunächst alle Paare von Objekten und überprüft, ob diese jeweils denselben Clustern zugeordnet sind oder nicht, um eine Koinzidenz-Matrix zu erzeugen. Nun vergleicht man die Matrizen der einzelnen Clusterergebnisse. Diese Variante ist deutlich flexibler und lässt genauere Vergleichsmöglichkeiten zu als die anderen Varianten. Allerdings ist der Aufwand zur Berechnung der Koinzidenzmatrix gerade bei größeren Datenmengen mit $O(n^2k)$ relativ hoch.

Für die Wahl des Maßes muss berücksichtigt werden, dass im hier betrachteten Fall Clusterpartitionen mit verschiedener Clusteranzahl möglich sind. Desweiteren sollte das Maß auch die Fuzzy-Zugehörigkeitsgrade nicht vernachlässigen. Maße der zweiten Variante scheiden direkt durch die Bedingung der gleichen Clusteranzahl aus und für die Maße der dritten Kategorie ist der Aufwand zu hoch. Allerdings sind die gängigen Maße der ersten Varianten auch für gleiche Clusteranzahlen oder nur für Nicht-Fuzzy-Partitionen vorgesehen und nur unter Einbußen erweiterbar, so dass kein geeignetes Maß gefunden wurde. Darum wird hier ein neues Fuzzy-Vergleichsmaß verwendet, das auf der ersten Variante beruht und im Folgenden vorgestellt wird.

Bei der Beschreibung des neu entwickelten Abstandsmaßes auf Fuzzy-Partitionen werden einige Operatoren aus der Fuzzy-Logik benutzt, auf die ich hier nicht detailierter eingehe. Verwendung finden hierbei Fuzzy-Varianten der logischen Negation, Implikation und Disjunktion (T-Conorm). Die hier betrachteten und verwendeten Operatoren werden mit Formeln angegeben, für darüber hinausgehende Informationen sei auf Fuzzy-Logik-Literatur verwiesen (z.B. [Thi95, KY95, KGK95]).

Zunächst ein paar Überlegungen für ein Ähnlichkeitsmaß bzw. Abstandsmaß auf zwei Fuzzy-Partitionen A, B: Ein Cluster $c^A \in A$ ist einem Cluster $c^B \in B$ ähnlich, wenn Objekte, die eine hohe Zugehörigkeit zu c^A in Partition A, haben auch eine solche zu c^B in der Partition B haben, also die Regel $c^A(x) \rightarrow c^B(x)$ im Sinne einer (mehrwertigen) Implikation erfüllt ist. Da die Regel auch von Daten unterstützt wird, die nichts mit dem Cluster c^A zu tun haben ($c^A(x)$ sehr klein), werden stattdessen die Daten gezählt, die der

Regel widersprechen. Um das passende Cluster $c^B \in B$ zu einem Cluster c^A zu bestimmen, werden für jedes Cluster aus B die negativen Beispiele gezählt und das Cluster mit den wenigsten negativen Beispielen gewählt. In Fuzzy-Logik ausgedrückt widerspricht ein Element x einer Regel mit dem Grad $\neg(c^A(x) \rightarrow c^B(x))$, wobei $\rightarrow$ für eine Fuzzy-Implikation (z.B. Lukasiewicz-Implikation) und $\neg$ für eine Fuzzy-Negation stehen. Als Negation wird in der Regel $\neg(x) = 1 - x$ benutzt. Für den Fehler einer Zuordnung $c^A(x) \rightarrow c^B(x)$ ergibt sich also:

$$d(c^A, c^B) = \sum_x \neg(c^A(x) \rightarrow c^B(x)) = \sum_x 1 - (c^A(x) \rightarrow c^B(x)).$$

Damit gilt:

$$d(c^A, c^B) = 0 \text{ falls } \forall x. c^A(x) \leq c^B(x).$$

Ein Cluster c_A wird dem Cluster c_B zugeordnet, für den der geringste Fehlerwert auftritt. Diese Fehlerwerte werden für alle Cluster aus A aufsummiert und zur Normierung durch die Anzahl der Daten n geteilt:

$$d(A, B) = \frac{1}{n} \sum_{c^A \in A} \min_{c^B \in B} d(c^A, c^B)$$

Aus Symmetriegründen wird $d(A, B)$ mit $d(B, A)$ zum kompletten Abstandsmaß $D(A, B)$ verknüpft:

$$D(A, B) = d(A, B) \oplus d(B, A),$$

wobei $\oplus$ für eine Fuzzy-Disjunktion (Fuzzy-T-Conorm) steht.

Wie in der theoretischen Betrachtung noch gezeigt wird, muss für die Fuzzy-Implikation $a \rightarrow b \geq 1 - a$ gelten, damit $d(A, B) \in [0, 1]$ garantiert werden kann. Dies ist für die Lukasiewicz, Zadeh, Kleene-Dienes und Reichenbach Implikation gegeben. Wenn man die Formel erweitert zu

$$d(c^A, c^B) = \sum_x min(a, 1 - (c^A(x) \rightarrow c^B(x)))$$

sind auch weitere Implikationen wie Gödel und Goguen verwendbar.

In den Experimenten wird hierbei die Lukasiewicz-Implikation ($a \rightarrow b = \min(1, 1 - a + b)$) und die algebraische Fuzzy-T-Conorm ($a \oplus b = a + b - ab$) angewendet.

Theoretische Betrachtungen Damit die Anwendung der Fuzzy-T-Conorm überhaupt möglich ist, muss gezeigt werden, dass $d(A, B) \in [0, 1]$ immer gilt. Durch

$$f_{AB}(c^A) := argmin_{c^B \in B} d(c^A, c^B)$$

wird eine Abbildung von A nach B definiert, und es gilt:

$$\begin{aligned} d(A,B) &= \frac{1}{n}\sum_{c^A\in A}\min_{c^B\in B} d(c^A,c^B) \\ &= \frac{1}{n}\sum_{c^A\in A} d(c^A, f_{AB}(c^A)) \\ &= \frac{1}{n}\sum_{c^A\in A}\sum_{x}(1-(c^A(x)\rightarrow f_{AB}(c^A)(x))) \\ &= \frac{1}{n}\sum_{x}\sum_{c^A\in A}(1-(c^A(x)\rightarrow f_{AB}(c^A)(x))) \end{aligned}$$

Wenn nun für die Implikation $a \rightarrow b \geq 1-a$ gilt, folgt

$$\begin{aligned} d(A,B) &= \frac{1}{n}\sum_{x}\sum_{c^A\in A}(1-(c^A(x)\rightarrow f_{AB}(c^A)(x))) \\ &\leq \frac{1}{n}\sum_{x}\sum_{c^A\in A} c^A(x) \\ &= \frac{1}{n}\sum_{x} 1 \\ &= 1 \end{aligned}$$

Desweiteren gelten folgende Eigenschaften:

Selbstidentität: Da immer $c^A(x) \rightarrow c^A(x) = 1$ gilt, folgt $d(A,A) = 0$.

Positivität: Für die Positivität ist es notwendig, dass eine Clusterstruktur nicht zwei Cluster mit identischen Zugehörigkeitsgraden zu allen Daten enthält. Dies ist aber sicherlich keine relevante Einschränkung, da bei zwei identischen Clustern immer ein Cluster unnötig ist. Für zwei derartige Clusterstrukturen A, B mit $A \neq B$ gilt, dass es für jede Abbildung $f : A \rightarrow B$ ein Cluster c_A und eine Instanz x gibt mit $c_A(x) \neq f(c_A)(x)$. Da die Summe aller Zugehörigkeiten von x jeweils 1 ist, existiert damit auch ein Cluster c_A, so dass $c_A(x) < f(c_A)(x)$. Damit gilt $c^A(x) \rightarrow f(c_A)(x) < 1$ und $d(A,B) > 0$.

Symmetrie: Folgt direkt aus $D(A,B) = d(A,B) \oplus d(B,A)$ und Symmetrie von $\oplus$.

Obwohl die Dreiecksungleichung in praktischen Anwendungen bisher nie verletzt wurde, ist ein formaler Nachweis dieser Eigenschaft bisher nicht gelungen.

Empirische Betrachtungen Ein Abstandsmaß auf Fuzzy-Partitionen soll den Unterschied zwischen zwei Clusterergebnissen berechnen, unabhängig davon, wie gut die Fuzzy-Partitionen sind und welche besser ist. Damit unterscheidet sich ein Abstandsmaß deutlich von den Gütemaßen, die man notfalls als Unähnlichkeit einer Partition zu der optimalen Partition interpretieren könnte. Um dies auch in der Evaluation zu verdeutlichen, wurde ein einfacher Datensatz gewählt, bei dem alle zu vergleichenden Partitionen gleich gut oder schlecht sind. Der Datensatz besteht dabei aus einem Ring von 1000 Datenpunkten. Ein zweidimensionales Beispiel ist in Abbildung 4.11 gezeigt.

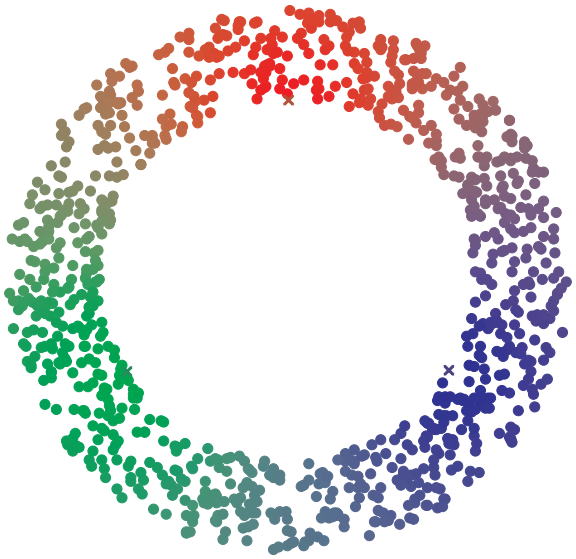

Abbildung 4.11: Ringdatensatz mit 3 Cluster

Auf diesem Datensatz werden Fuzzy-Clusterpartitionen mit jeweils drei Clustern erzeugt. Die Cluster teilen dabei den Ring jeweils in drei gleich große Bereiche auf, wie sie in Abbildung 4.11 zu sehen ist. Unabhängig von der Lage dieser Cluster sind alle derartigen Partitionen vergleichbar gute oder schlechte Clusterstrukturen. Zur Evaluation wird nun eine derartige Ausgangspartition erzeugt. Die Vergleichspartition wird mit der gleichen Partition initialisiert, so dass der Abstand zunächst 0 sein sollte. Die Cluster der Vergleichspartition werden nun in Uhrzeigerrichtung gedreht. Dabei sollte der Abstand bis zu einer Rotation von $\frac{2\pi}{6}$ zunehmen und dann wieder abnehmen. Bei einer Drehung von $\frac{2\pi}{3}$ liegen die Cluster versetzt übereinander, womit der Abstand wieder 0 sein sollte, da die Reihenfolge der Cluster für den Abstand keine Rolle spielen darf. Diese Rotation wird in Abbildung 4.12 nochmals veranschaulicht. Die Kurve des Abstandsmaßes mit zunehmender Rotation in Abbildung 4.13 zeigt den gewünschten Verlauf. An der Spitze zeigt Sie einen Knick bei $\frac{2\pi}{6} \approx 1.05$, der dadurch zustande kommt, dass hier die Zuordnung zwischen den Clustern wechselt.

Im zweiten Experiment wird die Clusteranzahl auf Grundlage des gleichen Ringda-

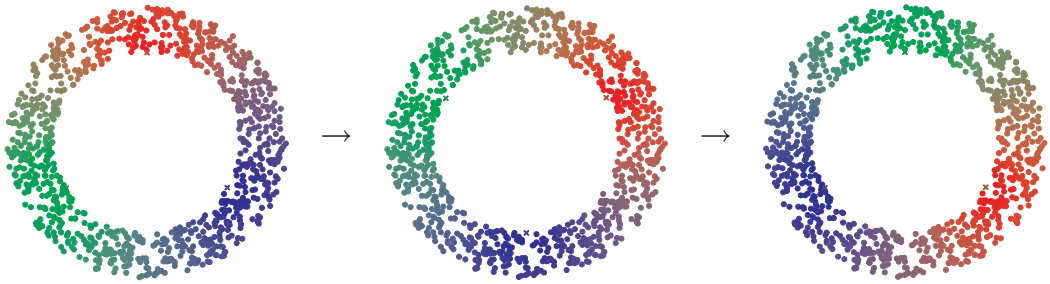

Abbildung 4.12: Clusterrotation

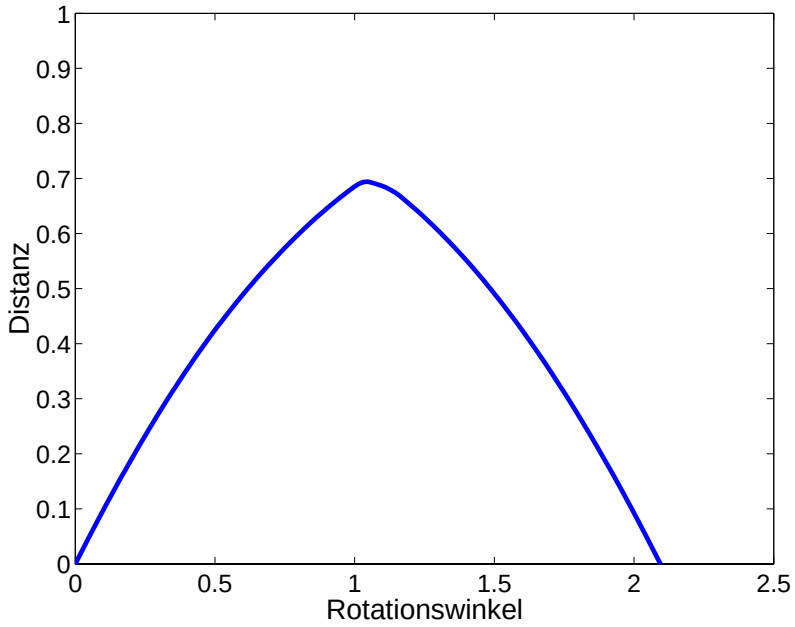

Abbildung 4.13: Verlauf des Distanzmaßes bei Rotation

tensatzen variiert. Zunächst wird eine Fuzzy-Clusterpartition mit zwei symmetrischen Cluster erzeugt. Anschließend wird die Clusteranzahl in jedem Schritt verdoppelt, indem zwischen zwei Cluster jeweils ein neues Cluster erzeugt wird. Abbildung 4.14 zeigt die Datensätze für 2, 4 und 8 Cluster. Betrachtet wird nun die Distanz zu der Ausgangspartition mit zwei Cluster. Dieser Wert sollte sich mit zunehmender Clusteranzahl vergrößern. Die Ergebniskurve in Abbildung 4.15 zeigt das entsprechende Verhalten. Damit zeigt das Distanzmaß zumindest für diese Experimente ein gewünschtes Verhalten.

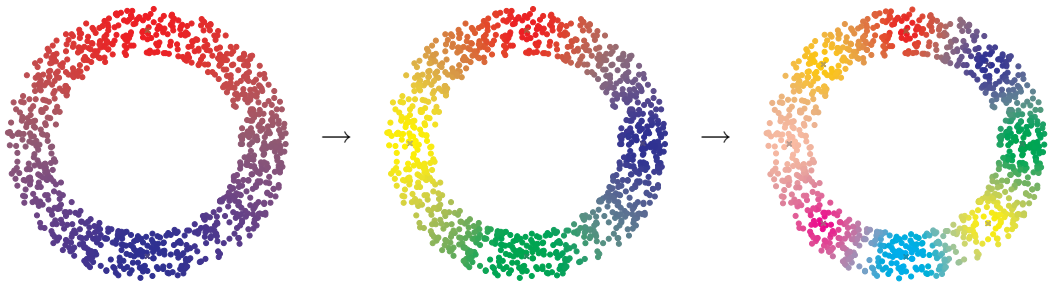

Abbildung 4.14: Verdoppelung der Clusteranzahl (2, 4 und 8 Cluster).

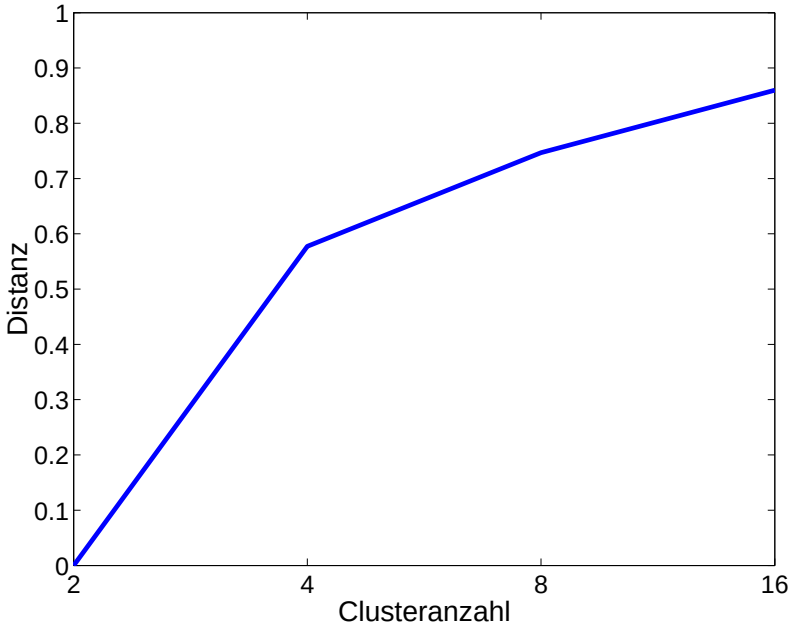

Abbildung 4.15: Distanzmaß bei Verdopplung der Clusteranzahl.

4.5.2 Synthetische Datenströme

Die ersten Experimente für das Clusterverfahren basieren auf synthetischen Datenströmen. Dabei liegt jedem Cluster von Datenströmen eine Referenzfunktion zugrunde. Diese wird für jeden Strom jeweils zufällig horizontal und vertikal verzerrt. Eine auf einem Random-Walk basierendes Prototyp-Funktion p eines Clusters wird hierbei wie folgt berechnet:

$$p(t+\delta t) = p(t) + p'(t+\delta t) \tag{4.7}$$

$$p'(t+\delta t) = p'(t) + u(t), \tag{4.8}$$

wobei $t = 0, \delta t, 2\delta t, 3\delta t, \ldots$ und $u(t)$ Zufallsvariablen sind, gleichverteilt in einem Intervall $[-p_a, p_a]$. Hierbei ist $p_a > 0$ ein Parameter für die Glätte von p. Je kleiner p_a ist,

desto glatter p. Zusätzlich wird die erste Ableitung p' auch durch ein Intervall $[-p_b, p_b]$ begrenzt, um die Steigung der Funktion kontrollieren zu können. Falls das Intervall überschritten wird, wird der Wert auf die nächste Intervallgrenze $-p_b$ oder p_b zurückgesetzt. Die horizontale und vertikale Verzerrung für einen Datenstrom x werden über Funktionen h und g wie folgt erzeugt:

$$x(t) \quad = p(t + h(t)) + g(t), \tag{4.9}$$

wobei h und g auf gleiche Weise wie p erzeugt werden. Die Wertebereiche von h und g sind auf ein Intervall $[-h_c, h_c]$ bzw. $[-g_c, g_c]$ begrenzt, um die maximale Verzerrung kontrollieren zu können. Falls der Wertebereich überschritten wird, wird der Funktionswert auf die nächste Intervallgrenze zurückgesetzt und gleichzeitig wird die zugehörige Ableitung auf 0 gesetzt. Für die Verzerrungsfunktionen g und h werden in allen Experimenten die in Tabelle 4.3 angegebenen Parameterwerte benutzt.

In einigen Experimenten werden als Prototyp-Funktionen auch andere Funktionen verwendet, um eine garantierte Anzahl von Cluster sicherzustellen. Die Verzerrung wird allerdings auch dann immer durch die hier beschriebenen Random-Walk-Funktionen realisiert.

4.5.3 Anpassung der Clusteranzahl

In diesem Experiment wird die Fähigkeit des Verfahrens zur Anpassung der Clusteranzahl in einer Datenstrom-Umgebung überprüft. Hierfür wird die Anzahl der Cluster während der Anwendung verändert, wobei ein Wechsel langsam verläuft. Zunächst werden nur zwei Cluster erzeugt. Regelmäßig wird dann auf vier Cluster gewechselt, um nach einiger Zeit wieder auf zwei Cluster zurückzukehren.

Realisiert wird dies, indem die 100 Datenströme in 4 Cluster zu je 25 Strömen eingeteilt werden. Als Prototyp-Funktion für das erste Cluster wird $p_1(t) = \sin(t)$ und für das zweite Cluster $p_2(t) = 1 - \sin(t)$ verwendet. Der Prototyp des dritten Clusters wird definiert durch $p_3(t) = \lambda p_1(t) + (1 - \lambda)\sin(t + \pi/2)$, wobei der Parameter $\lambda \in [0, 1]$ ist. Ebenso gilt für das vierte Cluster $p_4(t) = \lambda p_2(t) + (1 - \lambda)(1 - \sin(t + \pi/2))$. Für $\lambda = 1$ ist $p_3 = p_1$ und $p_4 = p_2$, es existieren also nur zwei Cluster. Für $\lambda = 0$ sind p_3 und p_4

Funktion h	Funktion g
$h_a = 0.5$	$g_a = 0.5$
$h_b = 5$	$g_b = 1.5$
$h_c = 30$	$g_c = 15$

Tabelle 4.3: Parameter der Verzerrungsfunktionen

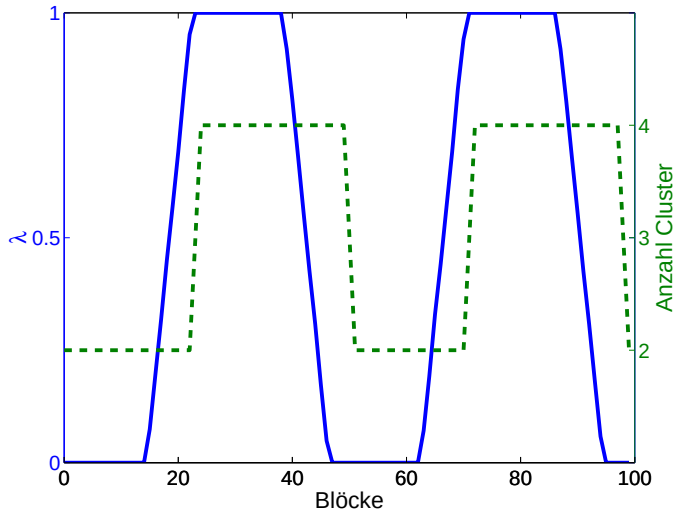

Abbildung 4.16: Wechsel der Clusteranzahl
Dargestellt sind der Parameter λ (blau durchgehend) und die Clusteranzahl (grün gestrichelt).

unabhängig von p_1 und p_2, so dass vier unabhängige Cluster vorliegen. Der Parameter λ wechselt zwischen 0 und 1 kontinuierlich innerhalb von 8 Blöcken. Ein Block besteht aus 512 Datenpunkten und das Sliding Window besteht aus 4 Blöcken. Der Parameter der exponentiellen Gewichtung wird so gesetzt, dass der letzte Datenpunkt im Sliding Window ein Gewicht von 0.4 hat.

Die Abbildung 4.16 zeigt neben den Werten von λ auch die vom Verfahren bestimmten Clusteranzahlen über eine Zeit von 100 Blöcken. Wie zu sehen ist, passt das Verfahren die Anzahl der Cluster korrekt an. Dies geschieht mit einer kleinen Verzögerung, da sich im Sliding Window auch noch ältere Daten befinden. Allerdings ist aus den Zugehörigkeitsgraden schon frühzeitig erkennbar, dass die Ströme der Cluster heterogener werden.

4.5.4 Anpassung der Clusterzugehörigkeit

Für dieses Experiment werden sehr ähnliche Datenströme wie im vorhergehenden Experiment verwendet. Hierbei sollen einige Datenströme zwischen zwei Cluster hin und her wechseln. Als Prototypen für die zwei Cluster werden wieder die Funktionen $p_1(t) = \sin(t)$ und $p_2(t) = 1 - \sin(t)$ verwendet. Die zwei Datenströme, die zwischen den Clustern wechseln sollen, erhalten den Prototyp $p(t) = \lambda p_1(t) + (1-\lambda)p_2(t)$, wobei $\lambda \in [0, 1]$ ist.

Abbildung 4.17 zeigt den Parameter λ zusammen mit dem durchschnittlichen Zugehörigkeitsgrad der beiden Ströme zum zweiten Cluster. Wie zu erkennen ist, passt sich

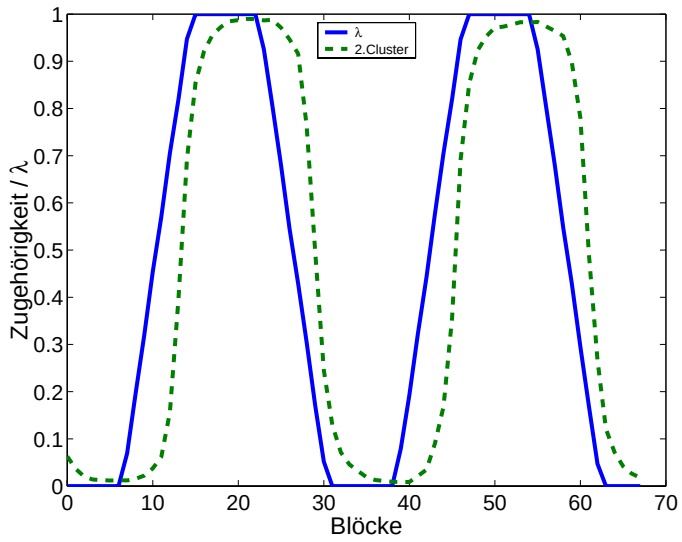

Abbildung 4.17: Wechsel zwischen Clustern I
Dargestellt sind der Parameter λ (blau durchgehend) und die mittlere Zugehörigkeit der zwei wechselnden Ströme zum 2. Cluster (grün gestrichelt).

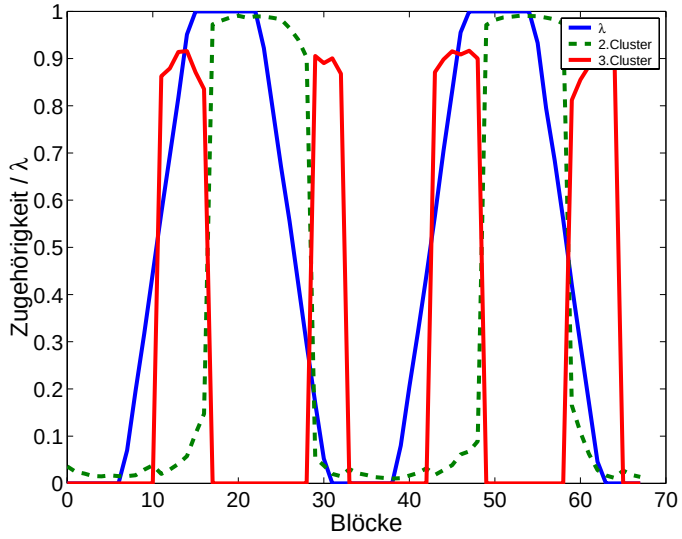

Abbildung 4.18: Wechsel zwischen Clustern II
Dargestellt sind der Parameter λ (blau durchgehend) und die mittlere Zugehörigkeit der 5 wechselnden Ströme zum 2. Cluster (grün gestrichelt) und 3. Cluster(rot durchgehend).

der Zugehörigkeitsgrad wieder mit einer kleinen Verzögerung korrekt an.

Das Experiment wurde wiederholt, diesmal mit fünf Datenströmen, die zwischen den Clustern wechseln. Die Ergebnisse in Abbildung 4.18 sind sehr ähnlich, mit der Ausnahme, dass die fünf Datenströme während eines Wechsels kurzfristig ein eigenes drittes Cluster bilden. Dieses Cluster entsteht, wenn die Datenströme relativ weit vom ersten Cluster entfernt sind, und verschwindet, wenn sie nah genug an das zweite Cluster kommen, was ebenso einem sinnvollen Verhalten entspricht.

4.5.5 Performanz und Qualität

Die Laufzeit-Performanz des Online-Clusteralgorithmus hängt direkt von der Stärke der Komprimierung, also der Anzahl verwendeter DFT-Koeffizienten ab. Deshalb wird der Zusammenhang zwischen der Anzahl der DFT-Koeffizienten und der Laufzeit untersucht. Eine stärkere Komprimierung bedeutet allerdings nicht nur eine niedrigere Laufzeit, sondern auch einen Genauigkeitsverlust bei der Abstandsberechnung, was direkten Einfluss auf die Clusterstrukturen hat. Deshalb ist es ebenso notwendig, diesen Zusammenhang zwischen Kompression und Qualität des Ergebnisses zu überprüfen.

Die im Experiment verwendeten Datenströme wurden wie am Anfang beschrieben erzeugt. Ihre Länge beträgt 500 Blöcke mit je 128 Werten und ein Zeitfenster besteht aus 16 Blöcken mit insgesamt 2048 Werten. Als Referenz-Clusterstrukturen dienen die 10 Verläufe von Clusterstrukturen, die durch 10-fache Anwendung dieser Datenströme ohne DFT-Komprimierung erzeugt wurden. Zunächst muss die Varianz der Clusterstrukturen ohne Kompression bestimmt werden. Dazu wurde das Clusterverfahren erneut mit den Datenströme ohne DFT-Kompression angewendet. Die erzeugten Clusterstrukturen wurden dann in jedem Zeitpunkt mit den Referenz-Clusterstrukturen verglichen. Anschließend wurde der durchschnittliche Abstand zu jedem der Referenz-Clusterstrukturen berechnet. Dies wurde wiederum 10-fach wiederholt, so dass insgesamt 100 Paare von Clusterstruktur-Verläufen verglichen wurden. Ebenso wurde für verschiedene DFT-Kompressionsstärken verfahren, die auch mit den Referenz-Clusterstrukturen verglichen wurden.

Die Ergebnisse sind in Tabelle 4.4 zu sehen und in der Abbildung 4.19 graphisch dargestellt. Wie man sieht, gibt es eine deutliche Veränderung zwischen 25 und 50 DFT-Koeffizienten. Für 50 und mehr DFT-Koeffizienten ist der Abstand nicht unterscheidbar zu der Varianz ohne Kompression. Erst mit weniger DFT-Koeffizienten unterscheiden sich die Clusterstrukturen deutlich von denen ohne Kompression.

Die Laufzeiten sind in Abbildung 4.20 dargestellt. Wie zu erwarten ist das Clustering mit wenigen DFT-Koeffizienten deutlich schneller als ohne DFT. Die Laufzeit steigt mit der Anzahl der verwendeten DFT-Koeffizienten linear an. Allerdings ist die Laufzeit bei

DFT-Koeffizienten	mittlere Distanz	(Standardabweichung)
ohne DFT	0.1102	(0.0102)
1024	0.1080	(0.0091)
750	0.1065	(0.0092)
500	0.1072	(0.0084)
250	0.1077	(0.0089)
100	0.1108	(0.0092)
50	0.1123	(0.0094)
25	0.1853	(0.0092)
10	0.2994	(0.0084)
5	0.3697	(0.0093)
2	0.3936	(0.0072)
1	0.3774	(0.0051)

Tabelle 4.4: durchschnittliche Distanz zu Ergebnissen ohne DFT-Kompression

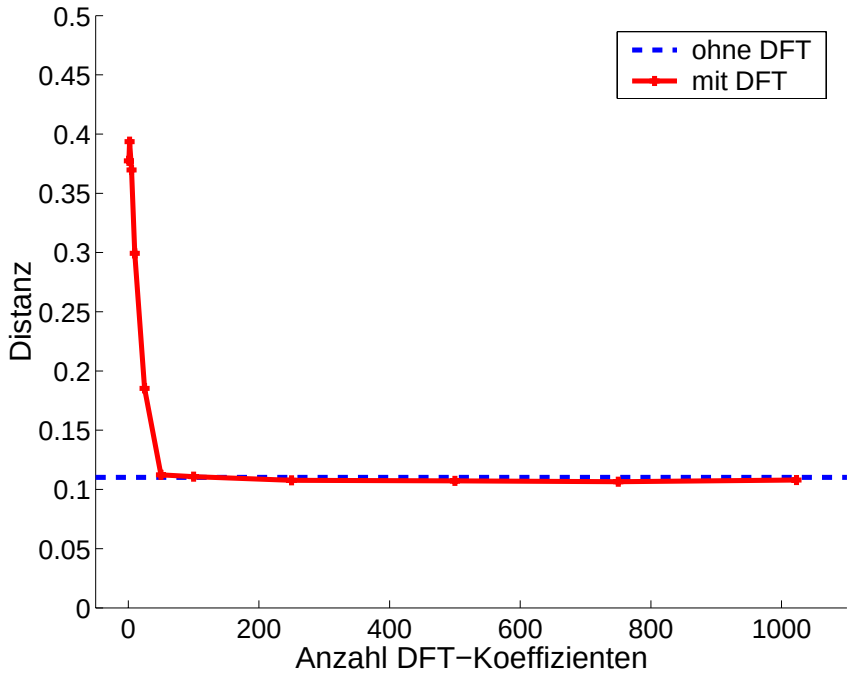

Abbildung 4.19: Qualitätsvergleich für unterschiedliche DFT-Kompressionsstärken. Die blau gestrichelte Linie stellt hierbei die mittlere Distanz zwischen 2 Clusterergebnissen ohne DFT dar.

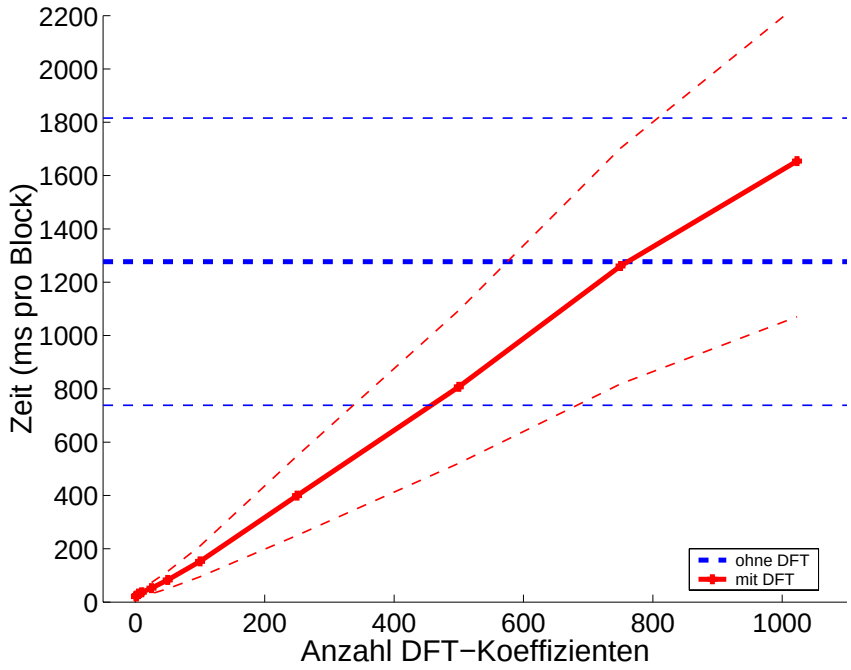

Abbildung 4.20: DFT-Laufzeitvergleich
Gegenüberstellung der Laufzeiten mit DFT-Kompression (rot durchgehend) zu ohne (blau gestrichelt), wobei der Bereich der Standardabweichung ($\pm std$) jeweils dünn gestrichelt gekennzeichnet ist.

Verwendung aller DFT-Koeffizienten (also ohne Komprimierung) höher als ohne DFT. Die effizienteste Clustervariante ohne erkennbaren Qualitätsverlust wäre also die Verwendung von 50 DFT-Koeffizienten. Man hat hierbei eine Kompression von ca. 1:20 und eine Laufzeit, die ca. 15 mal kleiner ist als ohne DFT bei vergleichbar guten Ergebnissen.

4.5.6 Real-World-Anwendung

Um die Anwendbarkeit auf Real-World-Daten zu zeigen,wurde das Verfahren auf Aktienkurse der Frankfurter Börse angewendet. Die Aktienkurse entsprechen hierbei den 110 Kursen des deutschen H-DAX Index. In dem Zeitraum vom 26. Mai bis zum 24. August 2004 wurden die Kurse innerhalb der Handelszeit alle 30 Sekunden aufgezeichnet. Zusammen mit dem Index selbst ergeben sich 111 Datenströme. Das Zeitfenster geht über 480 Aufzeichnungen, was einer Dauer von 240 Minuten, also 4 Stunden, entspricht, und wurde in 16 Blöcke eingeteilt. Ein Block beinhaltet damit 30 Zeitpunkte, was 15 Minuten entspricht. Die Gewichtung wurde so gewählt, dass das älteste Element des Zeitfensters noch ein Gewicht von 0.4 hat. Außerdem wurde für die Analyse der Aktienströme die Normalisierung geändert. Eine Aktie, die über das Zeitfenster hinweg kontinuierlich von

40 Euro auf 39,50 Euro fällt, hätte bei der Standardnormalisierung den gleichen Verlauf wie eine Aktie, die von 80 Euro auf 20 Euro fällt. Dieses Verhalten ist bei Aktienkursen sicherlich nicht erwünscht. Wirklich interessant ist dagegen der relative Verlauf zum momentanen Wert. Deshalb wurde bei der Normalisierung nicht durch die gewichtete Standardabweichung geteilt, sondern durch den aktuellen Aktienkurs, was ansonsten keinen Einfluss auf das Verfahren hat. Der Mittelwert wurde weiterhin auf 0 normalisiert.

Auswertung Zunächst einmal ist sehr auffällig, dass es die meiste Zeit nur zwei Cluster gibt. Dies mag zum einen daran liegen, dass die Anwendung von Qualitätsfunktionen (insbesondere bei Xie-Beni und verwandten Indizes) zu einer niedrigen Clusteranzahl neigt. Auf der anderen Seite liegt es sicherlich auch daran, dass Aktien an einer Börse meist einen relativ ähnlichen Verlauf haben, und deshalb selten mehr als zwei unterschiedliche Verläufe zu erkennen sind.

Als nächstes wurden die Clusterstrukturen über die gesamte Zeit betrachtet. Zu jedem Datenstrom wurden die vier Datenströme mit den insgesamt ähnlichsten Zugehörigkeiten ermittelt, die für einen Teil der Aktien in Tabelle 4.5 dargestellt sind. Die ähnlichsten Kurse zum HDax-Index selbst sind wie zu erwarten die Aktien, die am stärksten gehandelt wurden. Dies sind hier Daimler Chrysler, Münchner Rück, BASF und Deutsche Bank. Gleichzeitig haben zahlreiche Aktien den HDax unter den ähnlichsten Kursen, was ebenso sinnvoll erscheint. Außerdem erkennt man, dass die ähnlichsten Kurse oft Unternehmen aus der gleichen oder ähnlichen Branchen sind oder sonstige enge Beziehungen bestehen. So gehörten Daimler-Chrysler und BMW zu den ähnlichsten Kursen von Volkswagen, zu Bayer verlaufen Stada (Arzneimittel), BASF (Chemie) und BB Biotech sehr ähnlich, und Infineon verhält sich ähnlich zu Epcos (elektronische Bauelemente) und Siemens (ehemaliger Mutterkonzern und damals größter Aktionär).

Der eigentliche Sinn dieses Verfahrens ist allerdings nicht eine Gesamtauswertung im Nachhinein, sondern die Online-Analyse der aktuell erzeugten Clusterstrukturen. Die in Abschnitt 4.4.6 vorgestellten Visualisierungen sind jedoch zu umfangreich, um hier eine solche Analyse darzustellen. Deshalb wurde hierfür eine noch kompaktere Darstellung der aktuellen Clusterstruktur gewählt. Dazu werden die normierten Verläufe der Clusterzentren innerhalb einer Grafik dargestellt, wobei jedes Cluster eine andere Farbe bekommt. Um die Summe der Zugehörigkeiten zu visualisieren, wird die Dicke der Linie eines Clusters linear aus der Summe der Zugehörigkeit berechnet. Je dicker ein Clusterverlauf dargestellt ist, desto mehr Ströme gehören zu diesem Cluster. Bei der Interpretation der Ergebnisse sollte man allerdings beachten, dass dies normierte und vor allem gewichtete Verläufe sind.

Als Beispiel ist in Abbildung 4.21 einer der seltenen Momente mit mehreren Cluster zu sehen. Der Ausschnitt umfasst die Clusterstrukturen vom 24.06.2004 von 11 Uhr bis 13.30 Uhr im 30-Minuten-Abstand. Die linke Seite ist hierbei jeweils der aktuelle

Aktie	Aktien mit ähnlichsten Verläufen
HDAX	DAIMLERCHRYSLER,MÜNCHENER RÜCK,BASF,DEUTSCHE BANK
ADIDAS-SALOMON	FRESENIUS VZ.,FRESENIUS MEDICAL CARE,STADA NA,AMB GENERALI
ALLIANZ	MÜNCHENER RÜCK,HDAX,DEUTSCHE BANK,BMW
ALTANA	METRO,AMB GENERALI,HENKEL VZ.,CELESIO
AT & S AUSTRIA	TELES,EVOTEC OAI AG AKTIEN O.N,HOCHTIEF AG,PROSIEBENSAT.1 MEDIA
BASF	HDAX,RWE,E.ON,BAYER
BAYER	HDAX,STADA NA,BASF,BB BIOTECH
BB BIOTECH	STADA NA,KRONES VZ.,BAYER,ROFIN-SINAR
BMW	HDAX,VOLKSWAGEN,ALLIANZ,E.ON
COMMERZBANK	HYPOVEREINSBANK,HENKEL VZ.,STADA NA,DEUTSCHE POST
DAIMLERCHRYSLER	HDAX,VOLKSWAGEN,RWE,BAYER
DEGUSSA	WELLA VZ.,NORDDT. AFFINERIE,RHEINMETALL VZ.,VOSSLOH
DEUTSCHE BANK	MÜNCHENER RÜCK,HDAX,ALLIANZ,VOLKSWAGEN
DEUTSCHE BÖRSE	MEDION,KRONES VZ.,IVG IMMOBILIEN,CELESIO
DEUTSCHE POST	NORDDT. AFFINERIE,WEB.DE,TELES,VOSSLOH
DEUTSCHE TELEKOM	HDAX,RWE,DAIMLERCHRYSLER,COMMERZBANK
DRÄGERWERK VZ.	SÜSS MICROTEC,SÜDZUCKER,LEONI AG NAMENS-AKTIEN O,SCM MICROSYSTEMS
E.ON	HDAX,BASF,RWE,BMW
EPCOS	KONTRON,AIXTRON,PROSIEBENSAT.1 MEDIA,JENOPTIK
FRAPORT	KARSTADT QUELLE,FRESENIUS VZ.,SCHWARZ PHARMA,CELANESE
FREENET.DE	MOBILCOM,EVOTEC OAI AG AKTIEN O.N,QSC,MICRONAS
FRESENIUS MEDICAL CARE	FRESENIUS VZ.,FRAPORT,ADIDAS-SALOMON,E.ON
FRESENIUS VZ.	FRESENIUS MEDICAL CARE,FRAPORT,ADIDAS-SALOMON,AMB GENERALI
GPC BIOTECH	DIALOG,ROFIN-SINAR,WEB.DE,FJH
HYPOVEREINSBANK	COMMERZBANK,HENKEL VZ.,IDS SCHEER,LUFTHANSA
INFINEON	EPCOS,SIEMENS,SAP,QSC
JENOPTIK	SCM MICROSYSTEMS,WEB.DE,QSC,DIALOG
KARSTADT QUELLE	FRAPORT,EADS,DRÄGERWERK VZ.,AAREAL BANK AG AKTIEN O.
LINDE	VOSSLOH,AAREAL BANK AG AKTIEN O.,FJH,FIELMANN
LUFTHANSA	EVOTEC OAI AG AKTIEN O.N,HENKEL VZ.,MAN,WEB.DE
MAN	LUFTHANSA,JENOPTIK,WEB.DE,TUI AG
MERCK KGAA	WCM BETEILIGUNGS AG,ZAPF CREATION,MEDION,NORDDT. AFFINERIE
METRO	ALTANA,DOUGLAS,STADA NA,CELESIO
MG TECHNOLOGIES	KONTRON,EPCOS,TELES,MICRONAS
MOBILCOM	WEB.DE,DIALOG,QSC,FREENET.DE
MÜNCHENER RÜCK	ALLIANZ,HDAX,DEUTSCHE BANK,DAIMLERCHRYSLER
PROSIEBENSAT.1 MEDIA	SINGULUS,THIEL,WCM BETEILIGUNGS AG,AT & S AUSTRIA
PUMA	BOSS VZ.,KONTRON,AAREAL BANK AG AKTIEN O.,TUI AG
RHEINMETALL VZ.	KRONES VZ.,DEGUSSA,WCM BETEILIGUNGS AG,BILFINGER BERGER
RWE	DAIMLERCHRYSLER,BASF,HDAX,E.ON
SAP	INFINEON,BB BIOTECH,RWE,EPCOS
SCHERING	CELANESE,SCHWARZ PHARMA,IWKA,AT & S AUSTRIA
SCHWARZ PHARMA	NORDDT. AFFINERIE,CELANESE,VOSSLOH,SGL CARBON
SCM MICROSYSTEMS	JENOPTIK,ROFIN-SINAR,SÜSS MICROTEC,WEB.DE
SIEMENS	EPCOS,INFINEON,THYSSENKRUPP,HDAX
STADA NA	GPC BIOTECH,FJH,BB BIOTECH,BAYER
SÜDZUCKER	DRÄGERWERK VZ.,NORDDT. AFFINERIE,BOSS VZ.,FIELMANN
T-ONLINE	KARSTADT QUELLE,QSC,SINGULUS,FRAPORT
THYSSENKRUPP	EPCOS,AIXTRON,SIEMENS,HANNOVER RÜCK
TUI AG	PROSIEBENSAT.1 MEDIA,SINGULUS,EPCOS,MERCK KGAA
UNITED INTERNET	KONTRON,SGL CARBON,COMDIRECT BANK,WEB.DE
VOLKSWAGEN	DAIMLERCHRYSLER,BMW,HDAX,THYSSENKRUPP
VOSSLOH	NORDDT. AFFINERIE,SALZGITTER,SCHWARZ PHARMA,AWD HOLDING
WEB.DE	ROFIN-SINAR,QSC,DIALOG,JENOPTIK

Tabelle 4.5: Aktien mit ähnlichsten Cluster-Zugehörigkeiten:
Zu den Aktien in der linken Spalte, stehen in der rechten Spalte die vier Aktien mit den über den gesamten Zeitraum ähnlichsten Cluster-Zugehörigkeiten.

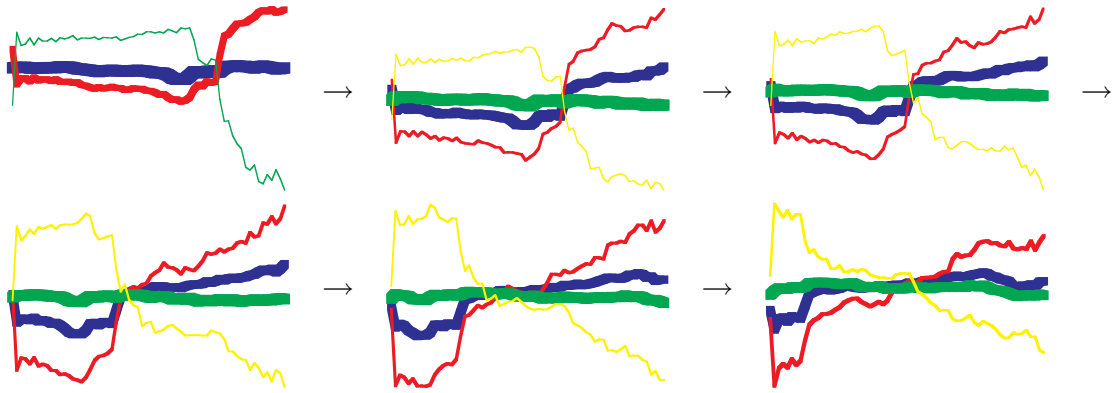

Abbildung 4.21: Clusterentwicklung der Aktienkurse I
im Zeitraum vom 24.06.2004 von 11 Uhr bis 13.30 Uhr

Moment. Die Artefakte am linken Rand sind Folgen der DFT, die versucht, Anfangs- und Endwert der Kurven anzugleichen. Da diese Artefakte für ähnliche Verläufe ähnlich ausfallen, hat er allerdings, wie in den vorangegangenen Tests gezeigt, kaum entscheidenden Einfluss auf die Clusterergebnisse. Zu diesem Zeitpunkt sind mehrere verschiedene Trends zu erkennen. Einmal gibt es ein Cluster von stark ansteigenden Kursen, ein anderes fällt genau zeitgleich deutlich ab. Die größten Cluster verhalten sich allerdings relativ ruhig. Hierbei kann man gut den Vorteil der unterschiedlichen Dicken der Clusterkurven beobachten, die für unterschiedlich große Cluster stehen. Weiter ist zu erkennen, dass die Kurse des Clusters mit dem Abfall weiterhin kontinuierlich fallen, und die Kurse des sprunghaft angestiegenen Clusters weiter ansteigen. In der letzten Grafik ist zu erkennen, dass die Aktien aller Cluster leicht abfallen, wobei das Cluster mit dem vorherigen Anstieg sich weiterhin am stabilsten hält.

Als zweite Analyse sind die Clusterstrukturen vom 04.07.2004 zwischen 11.30 Uhr und 14 Uhr in Abbildung 4.22 dargestellt. Wie man sieht, muss zwischen 11.30 Uhr und 12 Uhr einen Abfall fast aller Kurse stattgefunden haben. Hierbei war der Abfall der Aktien im roten Cluster stärker als im blauen. Die Aktien des blauen Clusters haben hierbei einen kleinen aber deutlicher als im roten Cluster zu erkennenden Anstieg vor dem Abfall hinter sich. Anschließend erholen sich die Aktien des blauen Clusters wieder leicht, während die Aktien des roten Clusters weiter leicht abfallen. Eine Stunde nach dem ersten starken Abfall tritt erneut ein deutlicher Abfall in allen Kursen auf und auch hier sind die Aktien des roten Clusters stärker betroffen. In der anschließenden Stunde ist eine Erholung aller Cluster zu erkennen, allerdings erholen sich hierbei wiederum die Aktien des blauen Clusters stärker als dies beim roten Cluster der Fall ist. Die Kurse des blauen Clusters übersteigen sogar wieder die Kurse vor dem letzten starken Abfall und scheinen weiter zu steigen.

Inwieweit solche Erkenntnisse und Analysen hilfreich für den Anleger sind und unter

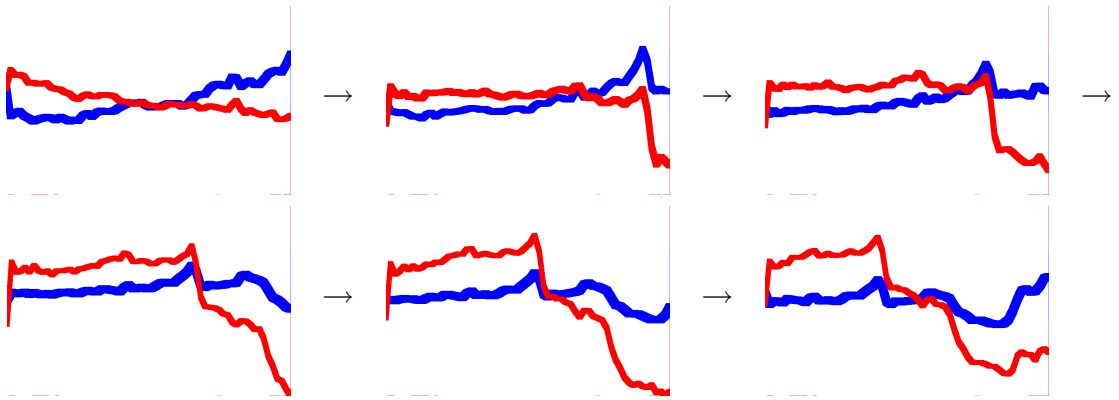

Abbildung 4.22: Clusterentwicklung der Aktienkurse II
im Zeitraum vom 04.07.2004 von 11.30 Uhr bis 14.00 Uhr

welchen Umständen welche Entscheidung zu treffen wäre, kann natürlich nur von Börsenspezialisten genauer bestimmt werden. Sicherlich sind hierfür auch weitere Tools notwendig, um konkretere Aussagen zu einzelnen Aktien und ganzen Gruppen von Aktien machen zu können. Es muss zunächst möglich sein, zu den einzelnen Clustern die zugehörigen Aktien anzuzeigen. Desweiteren wäre es bestimmt auch interessant, direkt erkennen zu können, in welchem Umfang Aktien zwischen den Clustern wechseln. Anhand der Kurse von stabileren Clustern können sicherlich bessere Aussagen und Entscheidungen getroffen werden als bei Clustern, deren Zusammensetzung sich ständig stark ändert. Eine automatische Weiterverarbeitung mit dem Know-How eines Börsenspezialisten wäre sicherlich angebracht und könnte durchaus nützlich sein.

Sicherlich ist bei der Verwendung von Aktien mit einem Update von nur allen 15 Minuten nicht unbedingt ein derart effizientes Verfahren notwendig, trotzdem zeigt es die Anwendbarkeit auf echte Datenströme. Die Ergebnisse erscheinen nicht nur wie willkürlich produzierte Clusterstrukturen, sondern sind sinnvoll erklärbar oder interpretierbar. Ebenso erscheint die kontinuierliche Entwicklung der Clusterstrukturen bei der Analyse sehr hilfreich. Diese Information ginge verloren, wenn man zu jedem Zeitpunkt eine eigenständige Clusteranalyse durchführen würde.

4.6 Variationen

Die hier vorgestellte Lösung ist eine möglichst allgemein gehaltene effiziente Fuzzy-Cluster-Variante für Datenströme. Wie die folgenden Beispiele zeigen, lässt sie sich für neue und erweiterte Problemstellungen erweitern und spezialisieren sowie mit anderen Komponenten kombinieren. Varianten mit zusätzlich erlaubtem Timeshift und anderen Kompressionsmöglichkeiten werden im Folgenden vorgestellt.

4.6.1 Timeshift

Eine mögliche Erweiterung bestünde darin, auch zeitlich versetzte Ströme als ähnlich zu erkennen. Durch Übertragung, Aufzeichnungen oder andere Einflüsse könnten Datenströme eine zeitversetzte ähnliche Entwicklung aufweisen. Derartige Ströme sollten ebenfalls als ähnlich gelten und, wenn möglich, zum selben Cluster gehören. Ein derartiges Szenario lässt sich ebenfalls im vorgestellten Ansatz integrieren. Zunächst muss festgelegt werden, wie groß diese zeitliche Verschiebung maximal sein darf. Wenn diese nun den Zeitraum von s Blöcken umfasst, müssen für jeden der Datenströme die DFT-Koeffizienten der letzten s Zeitfenster $w_t, w_{t-1}, \ldots, w_{t-s+1}$ gespeichert werden. Nach einem Update der Datenströme und der aktuellen Clusterzentren muss noch vor der K-Means / Fuzzy-C-Means Anwendung für jeden Datenstrom das Referenz-Zeitfenster gewählt werden, welches betrachtet werden soll. Dazu wählt man das Zeitfenster w, das am nächsten zu einem Clusterzentrum liegt:

$$w = \mathrm{argmin}_{\mathrm{w} \in \{\mathrm{w_t}, \mathrm{w_{t-1}}, \ldots \mathrm{w_{t-s+1}}\}} \min_{\mathrm{A}} ||\mathrm{w} - \mathrm{c_A}||.$$

Ist das Zeitfenster für jeden Datenstrom festgelegt, kann wie zuvor das Clustering fortgesetzt werden. Die Wahl des Zeitfensters wird also als erster Schritt in der Fuzzy-Cluster-Iteration ausgeführt. Bei einem notwendigen Update durch einen neuen Block wird jedes Zeitfenster durch das folgende ausgetauscht und wie bisher die Clusterzentren neu berechnet. Dadurch wird die Gefahr verringert, dass sich bei einem Update nichts an der Clusterstruktur ändert, weil einfach für jeden Datenstrom das vorhergehende Zeitfenster beibehalten wird. In Abbildung 4.23 wird die Vorgehensweise nochmals visuell dargestellt. Die maximale Verschiebung eines erkennbaren Timeshifts beträgt damit $s \cdot v$ Zeiteinheiten.

4.6.2 Alternative Distanz- und Gewichtsfunktionen, Kompressions- und Cluster-Verfahren

Die verwendete DFT-Kompression ist mit der Blockverarbeitung und der exponentiellen Gewichtung vollständig verflochten. Dadurch lassen sich das Kompressionsverfahren und die Gewichtsfunktion nicht direkt und beliebig austauschen. Eine einfache Möglichkeit, flexibler zu werden ist, die Kompression für jeden Block getrennt anzuwenden und jedem Block ein Gewicht, das für alle Elemente im Block gilt, zuzuweisen. Das Gewicht lässt sich dann bei jedem Update beliebig verändern. Der Vorteil hierbei ist, dass jedes beliebige Kompressionsverfahren und Distanzmaß angewendet werden kann. Ebenso kann damit jede beliebige Gewichtung der Blöcke verwendet werden. Nachteil ist, dass jeder Block separat komprimiert wird, wodurch nur relativ große Blöcke verwendet werden sollten. Außerdem gilt für jedes Element eines Blocks das gleiche Gewicht, was wiederum für

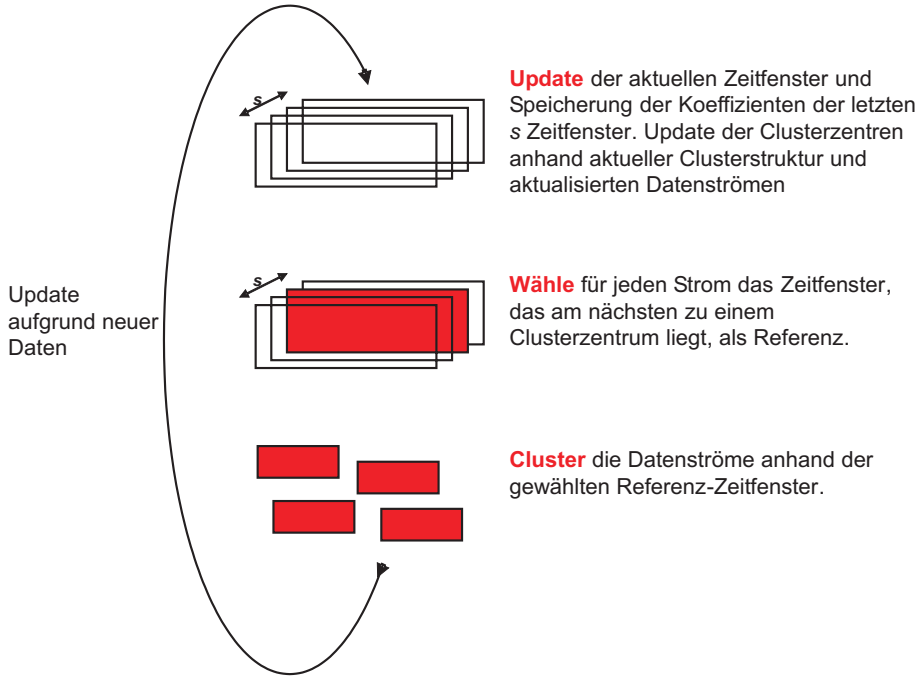

Abbildung 4.23: Timeshifts berücksichtigen: Visualisierung der Timeshift Variante

kleinere Blöcke spricht, um die Sprünge in der Gewichtung gering zu halten. Es muss hierfür also ein geeigneter Kompromiss gefunden werden. Alternative Cluster-Verfahren sind ebenso möglich, da die Distanzberechnung komplett unabhängig von dem Cluster-Verfahren ist. Allerdings muss ein Cluster-Verfahren, wie das vorgestellte Verfahren, die Clusterstruktur und die Clusteranzahl dynamisch anpassen können.

4.7 Resümee

Dieses Kapitel präsentierte ein Online-Clusterverfahren für Datenströme. Dazu war zunächst die Entwicklung einer Fuzzy-C-Means-Variante mit adaptiver Clusteranzahl notwendig und im Zusammenhang damit eine Optimierung der Xie-Beni-Gütefunktion. Die optimierte Gütefunktion sowie die adaptive Fuzzy-C-Means-Variante wurden anschließend evaluiert.

Desweiteren wurde eine selbst entwickelte effiziente adaptive DFT-Kompression mit Normalisierung und Gewichtung der Datenströme vorgestellt. Dabei werden die Datenströme aus Effizienzgründen in Blöcken verarbeitet. Größere Blöcke ermöglichen eine schnellere Berechnung, bei kleineren Blöcken wird die Clusterstruktur häufiger aktualisiert. Durch die Kompression ist eine deutlich effizientere Abstandsberechnung möglich.

Damit konnte nun der eigentliche Clusteralgorithmus auf Datenströmen vorgestellt werden, der den adaptiven Fuzzy-C-Means-Algortihmus verwendet und durch die effizienten Distanzberechnungen sehr effektiv anzuwenden ist. Die Hauptidee hierbei ist, die letzte Clusterstruktur vor einem Update als anschließende Initialisierung für das Clustern der aktualisierten Ströme zu verwenden.

Für die Evaluierung war es notwendig, ein geeignetes Vergleichsmaß von Fuzzy-Clusterstrukturen mit unterschiedlichen Clusteranzahlen zu entwickeln, da bisher existierende Ansätze nicht geeignet waren. Für dieses Maß wurden einfache und grundlegend wichtige Eigenschaften theoretisch und empirisch überprüft.

Bei der anschließenden Evaluierung des Online-Clusterverfahrens wurde hauptsächlich auf synthetisch erzeugte, kontrollierbare Datenströme zurückgegriffen. Dadurch war eine Überprüfung der korrekten Anpassung der Zugehörigkeitsgrade und Clusteranzahlen möglich. Anschließend wurde der Einfluß der DFT-Komprimierung auf das Ergebnis untersucht. Auch bei deutlichen Komprimierungsraten von 1:20 wurde kaum ein Unterschied zum Clustering ohne Komprimierung festgestellt. Erst bei noch stärkerer Komprimierung waren Veränderungen der Clusterstrukturen deutlich zu erkennen. Eine letzte Anwendung auf echte Aktienströme zeigte, dass das Verfahren auch auf Real-World-Daten sinnvolle Ergebnisse liefert.

Für die Entwicklung des Verfahrens mussten zunächst zahlreiche Detailprobleme gelöst werden, die nur indirekt mit dem Verfahren selbst zu tun haben. Dies betrifft unter anderem die geeignete Gütefunktion, das Vergleichsmaß für Fuzzy-Clusterstrukturen und die adaptive DFT-Komprimierung mit Normalisierung und Gewichtung, die ebenfalls evaluiert oder hergeleitet werden mussten. Ein Grund hierfür ist die Tatsache, dass das hier vorgestellte Verfahren der erste Lösungsansatz für eine derartiges Problemstellung ist. Dieser lässt sich deshalb sicherlich wiederum noch in einigen Punkten erweitern, anpassen und optimieren, wie die vorgestellten Variationen bereits gezeigt haben. Insgesamt ist ein brauchbares Clusterverfahren für Datenströme entstanden, das flexibel und erweiterbar anzuwenden ist und die Möglichkeit zu optimierten Variationen offen lässt.

Kapitel 5

Instanzbasierte Klassifikation auf Datenströmen

Nachdem im letzten Kapitel ein Verfahren für unüberwachtes Lernen auf Datenströmen vorgestellt wurde, beschäftigt sich dieses Kapitel nun mit einem Ansatz zum überwachten Lernen. Im Gegensatz zum unüberwachten Lernen, bei dem das Verfahren Objekte selbständig in Cluster einteilen muss, sind die Klassen beim überwachten Lernen anhand von Trainingsdaten bereits vorgegeben. Es wird also zunächst gelernt, um anschließend das gelernte Wissen anzuwenden. Die Klassifikation mit Datenströmen macht die Aufgabenstellung dadurch deutlich komplizierter, dass gleichzeitig gelernt und klassifiziert werden muss. Lern- und Klassifikationsphase finden also parallel statt. Dieses Kapitel geht zunächst auf die Klassifikation mit statischen Daten ein. Nach einer ausführlichen Betrachtung der unterschiedlichen Anforderungen für den adaptiven Fall mit Datenströmen wird der eigene Ansatz beschrieben und anschließend evaluiert.

5.1 Adaptives Klassifizieren

Wie in der Übersicht im Abschnitt 3.4 bereits erwähnt, besteht die Klassifikation auf statischen Daten aus zwei voneinander getrennten Phasen. Die Lernphase dient zur Extraktion eines Modells aus den dargebotenen gelabelten Trainingsdaten. Dieses gelernte Modell dient anschließend in der Klassifikationsphase dazu, ungelabelten Daten eine Klasse zuzuordnen. Dabei existieren zahlreiche verschiedene Vorgehensweisen. Bekannte Ansätze sind Entscheidungsbäume, Regellerner, Bayes Varianten, instanzbasiertes Lernen u.v.m. Sie unterscheiden sich stark in ihrer Art und dem verbundenen Aufwand zu lernen und zu klassifizieren.

Ein adaptiver Lerner auf Datenströmen besitzt als Input zwei Datenströme (siehe Abildung 5.1). Ein Datenstrom liefert gelabelte Trainingsdaten (Beispiele), die zum Lernen des Modells dienen. Der andere Input-Datenstrom liefert die Anfragen an den Klassifizierer in Form von ungelabelten Daten. Diese gilt es, mit Hilfe des gelernten Modells zu klassifizieren. Die klassifizierten Anfragedaten bilden dabei den Ausgabestrom des Klassifizierers. Beide Input-Datenströme können jederzeit Daten liefern. Lern- und Klassifikationsphase sind somit nicht, wie im statischen Fall, von einander getrennt, sondern laufen gleichzeitig ab. Jederzeit können neue gelabelte Trainingsdaten sowie ungelabelte Daten zum Klassifizieren eintreffen, so dass der Klassifizierer jederzeit in Lage sein muss, zu lernen und zu klassifizieren.

5.1.1 Praktische Szenarien

Umgebungen, in denen sich Zusammenhänge oder Eigenschaften regelmäßig unvorhersehbar oder zumindest auf unbekannte Weise verändern können, sind ideal für die Anwendung von adaptiven Lernen. Der Ursprung der Dynamik kann dabei sowohl von zu-

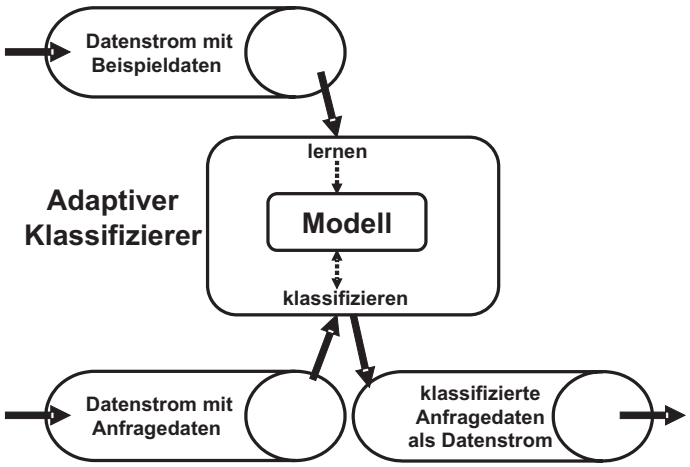

Abbildung 5.1: Prozess des adaptiven Klassifizierers

fälligen äußeren Einflüssen bestimmt werden, als auch durch menschliche Eingriffe. Beispiele findet man unter anderem bei natürlichen Zusammenhängen, wie sie z.B. bei der Wettervorhersage auftreten, die sich über die Zeit auf vorher nicht bekannte Weise ändern können. Durch Abnutzung von Bauteilen können sich Eigenschaften von Maschinen jederzeit ändern, wodurch ein Lernen und Optimieren der Einstellung notwendig werden kann. Adaptives Lernen ist auch in Umgebungen gefragt, die von menschlichem Verhalten beeinflusst werden. Das Verhalten bestimmter Menschen kann dabei von Interesse sein, wie bei strategischen Spielen oder bei der Bedienung von Programmen. Genauso verändert sich auch das Verhalten einer Gesellschaft, wie es beim Kaufverhalten, bei der politischen Meinungsbildung oder bei der Verkehrsmittel- und Streckenwahl der Fall ist. Sich ständig verändernde Zusammenhänge treten also in sehr vielen und unterschiedlichen Bereichen auf. Der Einfluss der Veränderungen auf das Lernen von Zusammenhängen ist dabei entscheidend, wodurch Lerner, die keine Veränderungen berücksichtigen, überfordert sind.

5.1.2 Klassifikationsrate beim adaptiven Lernen

Klassifizierer werden typischerweise anhand von Klassifikationsraten (Wahrscheinlichkeit einer korrekten Klassifikation) verglichen. Nun stellt sich die Frage, wie man Klassifikationsraten für adaptive Lerner berechnen kann. Da der datengenerierende Prozess sich über die Zeit verändern kann und das Modell auch ständig weiter trainiert wird, ändern sich auch die Klassifikationsraten über die Zeit. Desweiteren haben die Trainingsdaten eine feste Reihenfolge, so dass Standardverfahren wie Kreuzvalidierung oder Leave-One-Out nicht angewendet werden können. Auch der Ansatz, die Daten in Test- und Trainingsdaten aufzusplitten, ist nicht möglich. Zunächst hat man aber das grundsätzliche Problem,

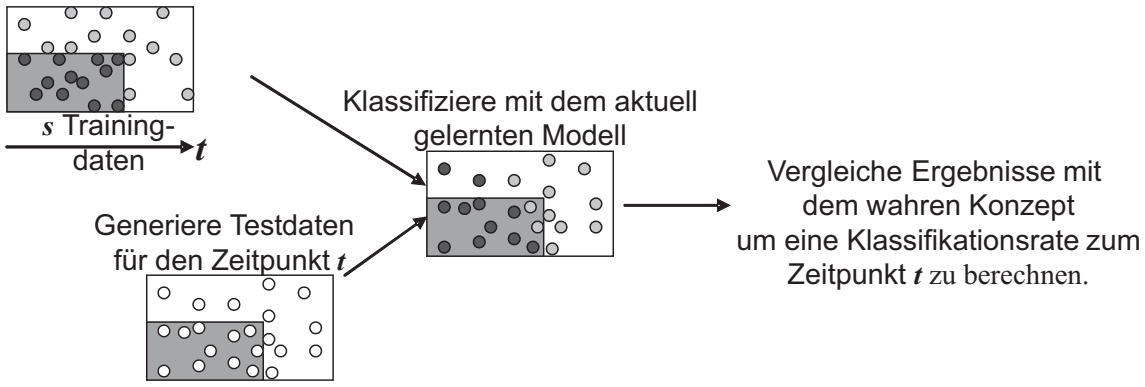

Abbildung 5.2: Absolute Klassifikationsrate

dass Klassifikationsraten in diesem Fall eigentlich Funktionen über die Zeit sind. Funktionen kann man nicht so einfach vergleichen, vor allem gibt es keine totale Ordnung. Deshalb ist es sinnvoll, einen Mittelwert für die Funktionen der Klassifikationsrate zu ermitteln, um diese dann zu vergleichen.

In der Praxis sind zwei verschiedene Varianten zur Berechnung von Klassifikationsraten für adaptive Lerner entstanden. Bei der *absoluten Klassifikationsrate* wird regelmäßig eine aktuelle Klassifikationsrate berechnet. Dazu wird nach einer festen Anzahl von Trainingsdaten eine *statische* Testmenge für einen festen Zeitpunkt generiert. Diese wird mit dem aktuell trainierten Modell klassifiziert und mit dem wahren Konzept verglichen (siehe Abbildung 5.2). So lässt sich eine Klassifikationsrate für diesen Zeitpunkt ermitteln. Die ermittelten Klassifikationsraten werden am Ende gemittelt, um einen Vergleichswert zu erhalten. Der Abstand zwischen zwei Messungen sollte nicht zu groß sein. Bei manchen Verfahren, wie bei FACIL [FTARR06], werden die Messungen nur nach jeweils 1000 Trainingsinstanzen vollzogen, so dass der Einbruch in der Rate nach einem Concept Shift gar nicht zum Tragen kommt. Der Vorteil an diesem Ansatz ist, dass die Berechnung auf einer beliebig großen Testmenge beruht und damit ausreichend genau bestimmt werden kann. Desweiteren lassen sich verschiedene Verfahren auch für feste Zeitpunkte vergleichen. Leider ist das Ergebnis von dem Abstand zwischen den Messungen und von der Anzahl verwendeter Daten abhängig. Der große Nachteil ist aber, dass man das wahre Konzept kennen muss, was nur für synthetische Daten der Fall ist. Daher kann diese Variante nicht für Real-World-Daten angewendet werden. Für diesen Zweck gibt es noch eine zweite Variante zur Berechnung der Klassifikationsrate.

Der Trainingsdatenstrom wird für die *Klassifikationsrate der Traingsdaten* auch als Testdatenstrom verwendet. Dazu wird jedes Beispiel, bevor es zum Trainieren genutzt wird, zum Testen verwendet. Mit dem aktuellen Modell wird es klassifiziert und mit der angegebenen Klasse verglichen. Damit lässt sich eine Klassifikationsrate über die gesamte Zeit berechnen (siehe Abbildung 5.3). Da zu jedem Zeitpunkt nur eine Testinstanz zur

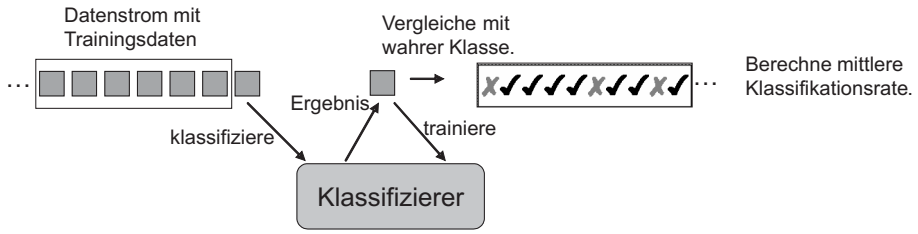

Abbildung 5.3: Kontinuierliche Klassifikationsrate

Verfügung steht, lässt sich natürlich keine Klassifikationsrate zu einem festen Zeitpunkt angeben wie in der vorhergehenden Variante. Die einzige Möglichkeit, eine zeitlich begrenzte Rate zu berechnen, besteht darin, den Mittelwert über ein Zeitintervall (*moving average* genannt) zu ermitteln. Der Vorteil dieses Verfahrens ist, dass es auch für Real-World-Datenströme geeignet ist, und ein Klassifizierer selbst die Möglichkeit hat, seine Rate zu berechnen und dies zur Erkennung von Concept Drift zu benutzen. Leider sind genauere Klassifikationsraten aber nur über einen größeren Zeitraum zu bestimmen. Desweiteren wird bei dem Verfahren angenommen, dass die Klasse einer neuen Trainingsinstanz noch dem bisherigen Modell entspricht, was bei Concept Drift nicht mehr der Fall ist. Außerdem wird die Klasse der Trainingsinstanz als wahr angenommen, was bei Rauschen nicht der Fall ist. Folglich fällt bei vorhandenem Rauschen die berechnete Klassifikationsrate fast um den Anteil von Rauschen niedriger aus. Trotzdem gibt es bei unbekanntem wahren Konzept keine Möglichkeit, die Klassifikationsrate besser zu bestimmen.

5.1.3 Concept Drift

Ein weiterer großer Unterschied zur Klassifikation auf statischen Daten ist das im Überblick (Abschnitt 3.4.1) schon erwähnte Phänomen des Concept Drift. Datenströme besitzen eine zeitabhängige Komponente. Wenn sich der datengenerierende Prozess über die Zeit verändert, bezeichnet man das als Concept Drift. Da es verschiedene Arten von derartigen Veränderungen gibt, unterscheidet man auch zwischen mehreren Arten von Concept Drift. Eine grundsätzlich Unterscheidung besteht zunächst zwischen virtuellem Concept Drift und echtem Concept Drift.

- Bei *echtem Concept Drift* ändert sich die wahre Klasse eines Datenpunktes. Die Klasse, die einmal für ein Objekt korrekt war, kann zu einem anderen Zeitpunkt falsch sein. Hierbei kann man noch zwischen *kontinuierlichem Concept Drift* und *Concept Shift* unterscheiden. Bei kontinuierlichem Concept Drift findet eine kontinuierliche Änderung des wahren Konzeptes statt (Abbildung 5.4). Die Grenzen zwischen den Klassen verschieben sich über die Zeit. Bei Concept Shift hingegen

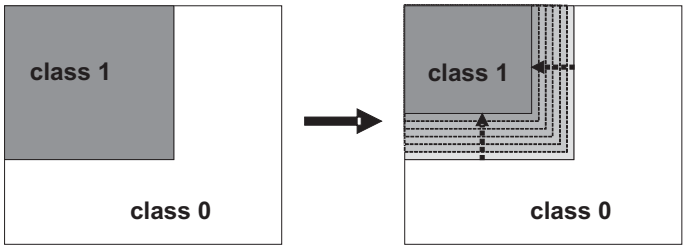

Abbildung 5.4: Beispiel für echten kontinuierlichen Concept Drift

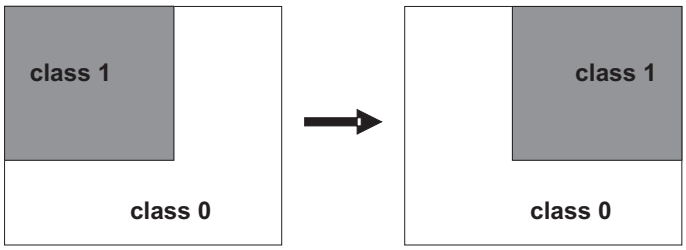

Abbildung 5.5: Beispiel für echten Concept Shift

ändert sich das wahre Konzept plötzlich (Abbildung 5.5). In der Regel existiert dafür in einem Zeitraum vor und nach diesem Shift kein weiterer Concept Drift. Ein Concept Shift muss nicht den gesamten Datenraum betreffen, sondern kann auch nur in einem Teilbereich auftreten, so dass die Veränderung nur auf einen Teil der Daten Auswirkungen hat.

- *Virtueller Concept Drift* verändert die Klassifikation der Datenpunkte nicht. Die Änderungen betreffen nur die Verteilung der Daten. Auch hier ist meist eine Anpassung des Modells notwendig. Ein Spezialfall ist *Sampling Shift*, bei dem sich die Verteilung der Trainingsdaten unabhängig von den Anfragedaten verändert. Dies bedeutet, dass Anfragen für den gesamten Datenraum auch dann zu beantworten sind, wenn es über einen Zeitraum nur Beispiele aus einem kleinen Bereich des Datenraums gibt. Die Herausforderung hierbei ist, das ältere Wissen über die anderen Bereiche nicht zu vergessen.

5.1.4 Aufwandsanforderungen

Im statischen Fall ist beim Aufwand meist nur die Klassifikationsphase entscheidend, da die Lernphase komplett im Voraus ablaufen kann. Deshalb machen Verfahren Sinn, die ein Modell erzeugen, mit dem sich neue Daten effizient klassifizieren lassen. Dies bieten z.B. Ansätze wie Entscheidungsbäume und Regellerner, bei denen die Klassifikation über die Auswertung einfacher Regeln abläuft.

Die Tatsache, dass das Modell beim adaptiven Lernen ständig weiter trainiert werden muss, verlangt eine hohe Flexibilität, die im Lernen auf statischen Daten nicht notwendig ist. Desweiteren spielen Aufwand für Lernen und Klassifizieren gemeinsam eine Rolle. Je nachdem, wie das Verhältnis zwischen Trainingsdaten und Anfragedaten ist, können dabei unterschiedliche Ansätze interessant sein. Wird häufig gelernt, ist ein Verfahren mit kurzer Lernphase interessant, wenn hingegen deutlich häufiger klassifiziert werden muss, sollte dies möglichst effizient unterstützt werden.

Somit scheinen Ansätze wie Entscheidungsbäume, die im Lernen auf statischen Daten sehr beliebt sind, für adaptives Lernen zunächst eher ungeeignet. Wie im Überblick (Abschnitt 3.4) aber schon erwähnt, gibt es trotzdem zahlreiche verbreitete adaptive Ansätze auf der Basis von Entscheidungsbäumen. Die Flexibilität wird in diesen Ansätzen dadurch erreicht, dass zahlreiche Entscheidungsbäume bzw. Teilbäume gleichzeitig gelernt werden und daraus ein aktuell bester Baum bestimmt wird. Der große Nachteil dieser Ansätze ist allerdings der zusätzliche Aufwand durch das Lernen vieler (Teil-)Entscheidungsbäume. Der Aufwand für die Klassifikation bleibt hingegen sehr gering. Die bisher vorgestellten Regellerner (Abschnitt 3.4.4) sind zwar flexibler als Entscheidungsbäume, aber trotzdem lässt sich das Modell nur durch Hinzunahme und Löschen verändern. Einzelne Regeln lassen sich nicht anpassen. Desweiteren existieren nur Verfahren für diskrete Daten. Numerische Attribute lassen sich zwar durch Diskretisierung auch benutzen, allerdings ist Diskretisierung bei Datenströmen deutlich schwieriger, da weder minimale und maximale Werte der Attribute vorher bekannt sind und die Grenzen festgelegt werden müssten, bevor die Daten bekannt sind.

Wenn ein Verfahren mit effizienter Lernphase gewünscht wird, ist ein flexibles und einfaches Modell notwendig, wie es bei instanzbasierten Verfahren der Fall ist. Das Modell besteht dabei nur aus einer Menge von Trainingsdaten. Die hohe Flexibilität ist dadurch gegeben, dass Einfügen und Löschen von Daten in einer Menge extrem einfach und schnell zu realisieren sind. Der Aufwand für eine Klassifikation ist dafür allerdings deutlich größer als bei modellbasierten Ansätzen. Deshalb macht die Verwendung eines derartigen Ansatzen immer dann besonders Sinn, wenn das Verhältnis von Trainingsdaten zu Anfragedaten relativ groß ist, also relativ häufig gelernt werden muss. Verschiedene instanzbasierte Verfahren für adaptive Lerner wurden bereits im Überblick (Abschnitt 3.4.4) vorgestellt. Wie erwähnt, verwenden die meisten Verfahren eine pauschale Strategie zum Löschen von Daten, die gar nicht oder nur bedingt vom tatsächlich auftretenden Concept Drift abhängt.

Dieses Kapitel stellt ein neues, universell verwendbares, instanzbasiertes Lernverfahren für Datenströme vor, das abhängig vom auftretenden Concept Drift Daten lernt und vergisst und dabei die Flexibilität eines instanzbasierten Ansatzes ausnutzt. Das anschließende Kapitel stellt ein instanznahes Regelverfahren vor. Durch die Regeln ist eine effizientere Klassifikation möglich. Um die Flexibilität zu erhöhen, werden als Regeln verallgemeinerte Instanzen benutzt, wodurch man eine Kombination zwischen instanzbasierten

und modellbasierten Verfahren erhält.

5.2 Instanzbasiertes Lernen

Hinter dem Begriff instanzbasiertes Lernen steht eine ganze Familie von Algorithmen aus dem maschinellen Lernen. Wie der Begriff im Gegensatz zum modellbasierten Lernen schon vermuten lässt, wird das Wissen auf der Basis einer Menge von Instanzen, *Fallbasis* genannt, repräsentiert. Da es nicht notwendig ist, ein Modell aus den Instanzen zu extrahieren, ist das Lernen deutlich effizienter durchzuführen. Eine eventuelle Auswahl der zu speichernden Instanzen und eine effiziente Speicherung sind dabei die wesentlichen Aufgaben. Das eigentliche Wissen wird erst mit der Kenntnis der zu klassifizierenden Daten lokal ausgewertet, weshalb man auch von einem *Lazy-Learning-Verfahren* spricht. Wissen wird nur dann extrahiert, wenn es tatsächlich benötigt wird. Dafür erhält man allerdings deutlich höhere Speicherkosten, als wenn nur ein gelerntes Modell gespeichert werden muss, sowie deutlich höheren Aufwand für jede einzelne Anfrage.

Ein typisches instanzbasiertes Verfahren ist „k-nearest neighbor“ (k-NN). Sei $D = A_1 \times \ldots \times A_b$ der Raum der Instanzen als kartesisches Produkt der Wertebereiche der b Attribute. Weiter sei $d : X \times X \mapsto \mathbb{R}^+$ ein Abstandsmaß über D. Für eine Instanz $x \in D$ sei $c_x \in C$ das zugehörige Label. In der Klassifikation ist C eine endliche Menge von m Klassen. Sei X nun die aktuell gespeicherte Menge von Instanzen aus der Menge der bisher gesehenen Trainingsinstanzen und x_0 eine neue Instanz, deren Klasse geschätzt werden soll. Zur Klassifikation nutzt k-NN die Menge N_{x_0} der $k \geq 1$ nächsten Nachbarn von x_0 aus X. Die Schätzung $c_{x_0}^{est}$ von c_{x_0} wird mittels „majority vote“ wie folgt ermittelt:

$$c_{x_0}^{est} = \arg\max_{c \in C} \mathrm{card}\{x \in N_{x_0} | c_x = c\}.$$

Die geschätzte Klasse ist also diejenige, die in der Umgebung N_{x_0} am häufigsten vorkommt. Mittlerweile gibt es auch zahlreiche Modifikationen und Erweiterungen für das k-NN Verfahren wie die gewichtete Aggregation:

$$c_{x_0}^{est} = \arg\max_{c \in C} \sum_{x \in N_{x_0} : c_x = c} w_x,$$

wobei w_x das Gewicht der Instanz x ist. Das Gewicht w_x ist normalerweise eine mit steigendem Abstand $d(x, x_0)$ monoton fallende Funktion.

5.3 Der Algorithmus IBL-DS

Dieser Abschnitt stellt ein neu entwickeltes instanzbasiertes Verfahren zur Klassifikation auf Datenströmen vor. Das Verfahren versucht, das Update besser auf konkret existie-

renden Concept Drift abzustimmen. Zunächst werden erst einmal die Möglichkeiten, die es für ein Update der Fallbasis gibt, diskutiert und die letztendlich gewählte Richtung begründet.

5.3.1 Vorüberlegungen

Dieser Abschnitt versucht, die schon in Abschnitt 3.4.4 vorgestellten Ansätze in Hinblick auf die Entwicklung eines neuen Ansatzes zu vergleichen und zu bewerten. Die einfachsten Verfahren dabei sind die pauschalen Verfahren. Diese versuchen erst gar nicht konkret existierenden Concept Drift zu erkennen, sondern wenden pauschale Löschstrategien an, in der Hoffnung, die gespeicherte Datenmenge damit aktuell zu halten. Dabei existieren unterschiedliche Ansätze wie sie unter anderem in [Sal97] beschrieben werden. Die einfachsten Ansätze löschen Daten allein aufgrund deren Alters wie bei Sliding-Window-Verfahren oder zeitlich gewichtetem Vergessen. Dies ist insoweit sinnvoll, als dass ältere Trainingsdaten sicherlich eher nicht mehr dem aktuellen Konzept entsprechen als neuere. Trotzdem hat man das Problem entscheiden zu müssen, wie genau man diese Altersgrenze, die durch die Größe des Sliding Windows oder der Gewichtskonstanten bestimmt wird, festlegt. Je nach Datenstrom können extrem verschiedene Werte sinnvoll sein.

Desweiteren können Daten aufgrund ihrer Nachbarschaft gewichtet werden, wodurch neben dem Alter auch die Lage eine Rolle spielt. Vorteil dieses Verfahrens im Gegensatz zu dem rein altersabhängigen Vergessen ist, dass in Bereichen des Datenraums, für die es keine neuen Trainingsdaten gibt, auch keine älteren Daten gelöscht werden. Dies macht Sinn, denn solange es keine neueren Daten in der direkter Nachbarschaft eines Objektes gibt, sind auch keine aktuelleren Aussagen für den Bereich vorhanden. Ob dieses Objekt veraltet ist, kann somit nicht festgestellt werden. Dadurch wird das Problem des virtuellen Concept Drift und Sampling Shift indirekt gelöst. Aber auch hier stellt sich die Frage, wie groß die Nachbarschaft sein soll und welche Nachbarn wie stark abgewichtet werden sollen. Hierdurch wird wieder indirekt festgelegt, wie alt ein Objekt in welcher Nachbarschaft sein muss, um gelöscht zu werden. Diese Ansätze sind deshalb pauschale Strategien, da das Löschen alleine von vorab festgelegten Faktoren abhängt, ohne Berücksichtigung des momentan vorhanden Concept Drift. Desweiteren haben sie Probleme beim Umgang mit Concept Shift, bei dem sich ein Großteil des wahren Konzeptes abrupt ändert. Die pauschalen Strategien vergessen trotzdem gleichmäßig langsam ihr bisheriges Wissen, wodurch das veraltete Wissen noch relativ lange Einfluss auf die Klassifikation aktueller Daten hat. Bei kontinuierlichem Concept Drift dagegen können auch pauschale Verfahren bei geeigneter Wahl der Parameter sehr gute Ergebnisse liefern.

Was den obigen Ansätzen fehlt, ist die direkte Erkennung von vorhandenem Concept Drift. Ein weiteres unschönes aber praktisch relevantes Phänomen ist Rauschen, also fehlerhaft markierte Daten. Gerade bei der Erkennung von Concept Drift spielt Rauschen

eine entscheidende Rolle. Wie am Anfang schon erwähnt (Abschnitt 2.4), fordern Domingos und Hulten [DH03], dass ein Verfahren robust gegenüber Rauschen und sensibel gegenüber Concept Drift sein soll. Wenn ein neues Trainingsobjekt erscheint, das nicht zum bisherigen gelernten Modell passt, kann sowohl Rauschen als auch Concept Drift der Grund sein. Allerdings sind die Folgen extrem unterschiedlich. Rauschen sollte möglichst direkt gelöscht werden. Neue Daten infolge von Concept Drift dürfen dagegen nicht gelöscht werden. Es sollten, wenn möglich, sogar die nun nicht mehr Konzept konformen Daten gelöscht werden. Also ist es wichtig, zwischen Concept Drift und Rauschen unterscheiden zu können. Allerdings ist das auf Grundlage der bisherigen Daten und dem bisher gelernten Modell wie eben beschrieben nicht möglich. Ein Unterschied ist erst durch weitere direkt benachbarte Trainingsdaten möglich. Falls diese sich wieder nach dem vorher gelernten Modell richten, handelt es sich um Rauschen und können gelöscht werden. Bei Concept Drift dagegen werden auch die weiteren Trainingsdaten dem neuen Konzept folgen und damit dem alten gelernten Modell widersprechen. Wenn dies der Fall ist, sollten die als veraltet erkannten Daten gelöscht werden. Beim Eintreffen neuer Daten kann nicht sofort auf Rauschen oder Concept Drift entschieden werden. Man ist nur in der Lage, „älteres“ Rauschen zu erkennen und anschließend zu löschen. Und auch bei Concept Drift kann die Strategie nur darauf hinaus laufen, ältere Daten, die nicht mehr dem aktuellen wahren Konzept entsprechen, zu erkennen und zu löschen. Bei jüngerem Rauschen kann man nur auf die Mithilfe des zugrunde liegenden Klassifikationsverfahrens hoffen. So ist es sinnvoll, k-NN mit einem höheren Wert für k zu benutzen, um robuster gegenüber Rauschen zu sein. Ob ein Objekt gelöscht werden kann hängt also ab von Alter, Lage und Konsistenz.

Ein komplett anderer Ansatz versucht den Concept Drift direkt mittels statistischen Methoden zu erkennen [KBDG04, CGM03, GMCR04]. Als Grundlage dient die aktuelle Klassifikationsrate. Solange sich das wahre Konzept nicht ändert, wird das gelernte Modell wie bei inkrementellen Lernern ständig verfeinert, was eine sich verbessernde oder zumindest gleichbleibende Klassifikationsrate zur Folge hat. Tritt aber nun Concept Drift auf, dann verschlechtert sich die Klassifikationsrate, was sich statistisch erkennen lässt. Hat man nun Concept Drift erkannt, ist die Frage, wie man damit umgeht. Die einfachste Variante ist, alle oder zumindest einen Großteil der Daten zu löschen und mit neuen Daten das Lernen wieder zu beginnen. Dieses Verfahren funktioniert besonders bei Concept Shift sehr gut, da bei der Erkennung altes Wissen im großen Umfang vergessen wird. Problematischer ist die Erkennung von Concept Drift, der nur einen kleinen Teil des Datenraumes betrifft. Da der gesamte Datenraum Einfluss auf die Klassifikationsrate hat, ändert sich diese nur langsam. Und wenn dieser Concept Drift dann endlich erkannt wurde, wird auch ein großer Teil des nicht veralteten Wissens gelöscht. Den Datenraum aufzuteilen und in jedem Bereich Concept Drift über die Klassifikationsrate zu erkennen, funktioniert auch nur bedingt. Denn je kleiner der Bereich wird, desto seltener gibt es Beispiele für diesen Concept Drift, womit die Erkennung deutlich länger dauert. Die Idee ist nun, die vorhergehend besprochenen pauschalen Ansätze mit der diskutierten lokalen Erkennung

von Concept Drift zu optimieren, um Concept Drift und lokal begrenzten Concept Shift zu erkennen, und dies dann mit einer statistischen Erkennung von globalen Concept Shift zu kombinieren.

Ein weiterer Punkt, den man bei einem Update berücksichtigen sollte, ist Redundanz. Um immer ein möglichst flexibles Modell zu haben, macht es Sinn, auch unnötige, redundante Daten zu entfernen. Je weniger Daten gespeichert werden, desto einfacher und schneller ist eine Anpassung möglich. Dies gilt besonders für Concept Shift, bei dem meist ein Großteil des Modells in kürzester Zeit entfernt und neu gelernt werden muss. Was im statischen Fall relativ einfach ist, stellt sich im adaptiven Fall deutlich problematischer dar. Das Hauptproblem ist, dass ein Objekt nicht nur im aktuellen Moment unnötig sein soll, es darf auch in Zukunft nicht mehr wichtig werden, damit es entfernt werden kann. Und genau dieser zweite Aspekt ist nicht sicher vorherzusehen. Schließlich gilt für jedes Objekt, das irgendwann einmal die Grenze zu einer anderen Klasse direkt an das Objekt grenzen kann und dieses Objekt somit entscheidend wird. Das Löschen von zur Zeit unnötigen Daten birgt immer die Gefahr, Daten zu löschen, die später einmal wichtig werden. Wie im Beispiel für Redundanz bei Concept Drift/Shift in Abbildung 5.6 dargestellt ist, können zunächst unwichtige Daten bei entsprechendem Concept Drift hilfreich zur Klassifikation werden. Auf der anderen Seite wird durch das Löschen von unnötigen Daten das Modell kleiner und damit flexibler, womit schneller auf Concept Drift und vor allem auf Concept Shift reagiert werden kann. Hierbei gilt es wieder, einen guten Ausgleich zu finden.

5.3.2 Update der Fallbasis

In diesem Abschnitt wird der neue Algorithmus vorgestellt. Wie im letzten Abschnitt erwähnt, besteht das Verfahren aus zwei getrennten Teilen. Einmal gibt es einen statistischen Ansatz, der für die schnelle Erkennung von Concept Shift zuständig ist. Ein zweiter heuristischer Ansatz versucht, auf Basis von Alter, Lage und Konsistenz durch Concept Drift veraltete Daten und Rauschen zu erkennen und zu löschen.

Heuristische Erkennung von Concept Drift und Rauschen Zunächst ist instanzbasiertes Lernen von Natur aus inkrementell, da ein Update nur aus der Speicherung der neuen Instanz besteht. Beim adaptiven Lernen kommt noch die Behandlung von Concept Drift hinzu. Grundsätzlich hat man nur die Möglichkeiten, Daten in die Fallbasis einzufügen oder zu löschen, wobei das vorgestellte Verfahren die Größe und Zusammensetzung der Fallbasis selbständig optimiert. Wenn ein neue Instanz x_0 erscheint, wird sie zur Fallbasis hinzugefügt. Eine neue Instanz nicht zu speichern, würde nur im Falle von Rauschen Sinn machen. Dies ist aber, wie oben schon erklärt, nicht erkennbar bzw. von neu auftretendem Concept Drift unterscheidbar. Dafür wird allerdings überprüft, ob ande-

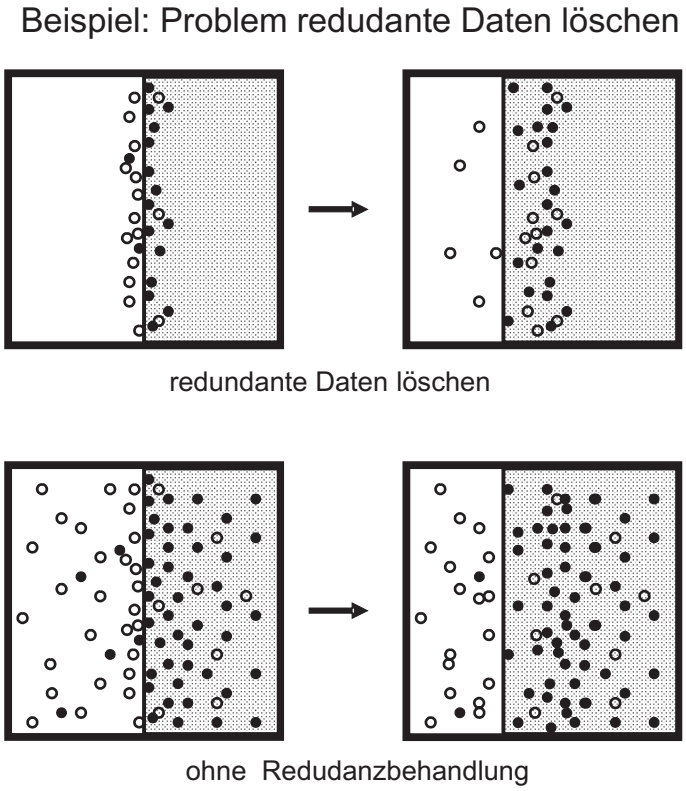

Abbildung 5.6: Problem von Redundanz und Concept Drift

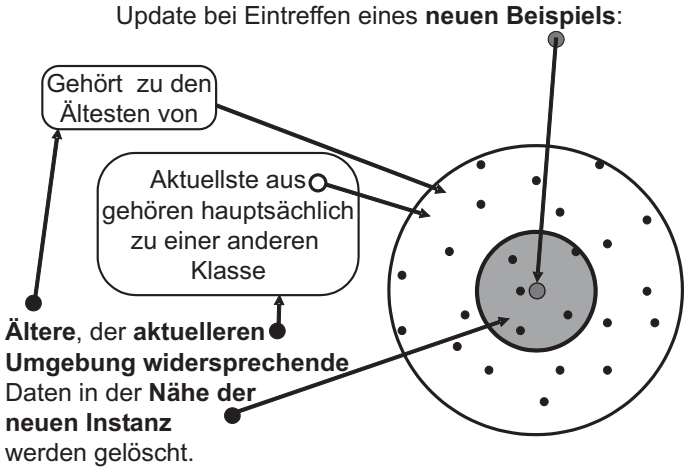

Abbildung 5.7: Lokales Update zur Erkennung von Concept Drift

re gespeicherte Instanzen durch das Einfügen der neuen Instanz gelöscht werden können und sollen. Dafür wird zunächst eine Menge C in der Nachbarschaft von x_0 als Kandidatenmenge ermittelt. Diese Nachbarschaft der ähnlichsten Daten ist durch die nächsten k_{sim} Nachbarn von x_0 definiert, wobei meist $k_{sim} = k$ im Falle von k-NN gewählt wird. Aus dieser Menge sollen nun ältere Daten, die mit dem aktuellen Modell nicht konsistent sind, gelöscht werden.

Zur Ermittlung des relativen Alters und insbesondere der Konsistenz, dient eine weitere Umgebung der k_{test} nächsten Instanzen von x_0. Diese Menge wird auch *Testumgebung* genannt, wobei $k_{test} > k_{sim}$ ist. Das aktuelle Modell wird von der Menge N den k_{new} aktuellsten Instanzen einschließlich x_0 repräsentiert, wobei wegen der Einfachheit hier $k_{new} = k_{sim}$ gewählt wird. Die Instanzen dieser Menge können nicht gelöscht werden, auch wenn sie sich in der Kandidatenmenge befinden. Für eine Concept-Drift-Erkennung sollte das aktuelle Modell zunächst möglichst eindeutig sein und der Klasse c_{x_0} der neuen Instanz x_0 entsprechen. Als eindeutig gilt das aktuelle Modell dabei, wenn

$$p_1 - p_2 > \frac{1}{m+1}$$

gilt, wobei p_1 und p_2 die relative Häufigkeiten der häufigsten und zweit häufigsten Klassen der Daten in N sind und m die Anzahl der Klassen ist. Ist dies erfüllt, werden alle Daten aus C gelöscht, die nicht in N sind, also nicht zu den aktuellsten gehören und die einer anderen Klasse als die in N dominierende Klasse zugeordnet sind. Dieser Teil des Updates ist in Abbildung 5.7 nochmals grafisch veranschaulicht.

Falls keine Daten gelöscht werden und durch das Hinzufügen der neuen Instanz die maximale Größe der Fallbasis überschritten wird, wird die älteste Instanz aus C unabhängig von der Klasse gelöscht. Dadurch kann eine obere Schranke für die Größe der Fallbasis und damit für den notwendigen Speicherplatz garantiert werden.

Symbol	Bedeutung
m	Anzahl der Klassen
l	Anzahl der Attribute
x_0	neues Beispiel zum Trainieren
C	Zum Löschen in Frage kommende Kandidatenmenge um x_0,
k_{sim}	Größe der Nachbarschaft von C
k_{test}	Größe der Testumgebung um x_0
N	Aktuellste Instanzen der Testumgebung
k_{new}	Größe von N ohne x_0
p_1	Relative Häufigkeit der häufigsten Klasse in N
p_2	Relative Häufigkeit der zweit häufigsten Klasse in N
h	Anzahl Instanzen, die zur Berechnung der Fehlerrate benutzt werden
h_{last}	Anzahl Instanzen zur Berechnung der aktuellen Fehlerrate bei Concept Shift
p	Fehlerrate der letzten h Trainingsbeispiele
s	Standardabweichung von p
p_{min}	Niedrigste beobachtete Fehlerrate p
s_{min}	Zu p_min gehörende Standardabweichung
p_{last}	Fehlerrate der letzten h_{last} Trainingsbeispiele
p_{dif}	Geschätzte Veränderung der Fehlerrate durch Concept Shift
q	Untergrenze der Fehlerrate für die Erkennung von Concept Shift
α	Signifikanzniveau bei der statistischen Concept-Shift-Erkennung.

Tabelle 5.1: IBL-DS Notationen

Statistische Erkennung von Concept Shift Der bisher beschriebene Teil nimmt eine langsame und stetige Anpassung der Fallbasis vor. Dies ist für eine stetigen Concept Drift sinnvoll, wenn sich aber das Modell schlagartig ändert (Concept Shift), passt sich die Fallbasis auch nur langsam und damit verzögert an. Für diesen Fall dient eine zusätzliche globale statistische Analyse der Klassifikationsrate. Dazu wird der Klassifikationsfehler p mit zugehöriger Standardabweichung $s = \sqrt{\frac{p(1-p)}{100}}$ über die letzten 100 Trainingsbeispiele betrachtet. Ein einfacher Fall tritt ein, wenn $p > 1 - \frac{1}{m}$ gilt, die Trefferquote also schlechter ist, als wenn immer die häufigste Klasse gewählt werden würde. Zusätzlich wird Concept Shift bei einer signifikanten Verschlechterung der Fehlerrate p gemeldet. Dazu wird die geringste Fehlerrate seit dem letzten Update mit zugehöriger Standardabweichung (p_{min} und s_{min}) als Referenz gehalten. Statistische Signifikanz wird durch einen einseitigen Standard t-Test mit einem Signifikanzniveau (Wahrscheinlichkeit einer irrtümlichen Erkennung) von α entschieden, wobei die Bedingung

$$p + s > p_{min} + z_{1-\alpha} s_{min}$$

erfüllt sein muss. Da sich die Fehlerrate bei Concept Shift eindeutig verschlechtert und eine versehentliche Erkennung möglichst vermieden werden soll, wird für das Signifikanzniveau mit $\alpha = 0.001$ ein relativ hoher Wert gewählt. Allerdings gibt es noch ein Problem bei sehr kleinen Fehlerraten p. Hierbei ist die Standardabweichung s ebenfalls sehr klein. Dadurch ist trotz niedrigem Signifikanzniveau eine fehlerhafte Erkennung leicht möglich. Bei $p_{min} = 0, s_{min} = 0$ würde sogar jeder Fehler bei jedem Signifikanzniveau $\alpha > 0$ zu einem Concept Shift führen. Um dies zu vermeiden, sollte der Fehler p zur Concept-Shift-Erkennung einen Mindestwert q überschreiten (als Defaultwert wird $q = 0.25$ gewählt). Dies ist unproblematisch, da bei geringerem Concept Shift auch die Concept-Drift-Erkennung für eine Adaption ausreicht.

Die Reaktion auf einen Concept Shift hängt vom Ausmaß ab, wozu die Änderung der Fehlerrate p_{dif} dient. Um diese abzuschätzen, wird p_{min} mit der Fehlerrate der letzten 20 Trainingsbeispiele p_{last} verglichen, womit gilt

$$p_{dif} = p_{last} - p_{min}.$$

Falls $p_{last} > 1 - \frac{1}{m}$ gilt, sollt komplett neu gelernt werden, weshalb dann $p_{dif} = 1$ gesetzt wird. Anschließend wird ein Anteil von p_{dif} der aktuellen Fallbasis gelöscht, wobei allerdings mindestens k_{test} Instanzen erhalten bleiben. Dabei sollen die zu löschenden Daten möglichst alt und über den gesamten Datenraum gleichmäßig verteilt gelöscht werden. Bei der Auswahl der Daten aus der Fallbasis hilft die gewählte Datenstruktur M-Tree, worauf im nächsten Abschnitt 5.3.3 genauer eingegangen wird.

Behandlung von Redundanz Zum Update gehört als letztes die Erkennung und Behandlung von redundanten Instanzen. Dies ist allerdings, wie im letzten Abschnitt schon

diskutiert, sehr problematisch. Darum wird bei der Erkennung von Redundanz sehr vorsichtig vorgegangen. Damit Daten als redundant gelöscht werden können, muss sichergestellt sein, dass sie zur Klassifikation nicht benötigt werden und dass auch bei auftretenden Rauschen oder Concept Drift genügend Instanzen in der Umgebung vorhanden sind. Damit Redundanz vorliegt muss deshalb die gesamte aktuelle Umgebung N einer Klasse angehören und auch in der Testumgebung k_{test} darf es maximal nur einen Fehler geben. Zusätzlich wird verlangt, dass die Dichte der Umgebung nicht überdurchschnittlich groß ist, wobei als Vergleichsmaß der Abstand der entferntesten Instanz benutzt wird.

Das Pseudocode des gesamten Updates befindet sich in Algorithmus 5.1.

5.3.3 Technische Details

Parameter Hier werden alle Parameter des Verfahrens, die auch in der Beschreibung erwähnt wurden, nochmal kurz beschrieben:

- k_{new}: Größe der lokalen Umgebung, in der bei einem Update bestehende Daten gelöscht werden können. Wie in der Beschreibung diskutiert, sind die Werte 3, 5 und 7 sinnvollste Einstellungen.

- k_{test}: Größe der bei einem Update betrachteten lokalen Umgebung, die für die Entscheidung, ob Daten gelöscht werden können, herangezogen wird. Der Wert des Parameters hängt auch mit dem Wert für k_{new} zusammen und sollte deutlich größer sein. Ein relativ kleiner Wert steht für höhere Flexibilität, ein höherer Wert für mehr Robustheit. Allerdings zeigen die Ergebnisse der Evaluation in Abschnitt 5.4.5, dass auch größere Änderungen dieses Parameters zu relativ geringen Qualitätsunterschieden der Ergebnisse führen.

- α: Signifikanzniveau der statistische Erkennung von Concept Shift. Da ständig auf Concept Shift getestet wird und Fehlalarme schwerwiegende Folgen haben, sollte das Signifikanzniveau sehr niedrig liegen (<0.01).

- q: Mindestfehler für die Erkennung von Concept Shift. Da bei der Erkennung von Concept Shift global Daten gelöscht werden, sollte das Ausmaß des Concept Shift eine gewisse Größe übersteigen.

- h: Anzahl der verwendeten letzten Trainingsdaten zur Schätzung der Klassifikationsrate. Da dieser Wert als Grundlage der statistischen Concept-Shift-Erkennung dient, sollte er Zwecks Vermeidung einer Fehlentscheidung auf der einen Seite möglichst groß sein, aber auch klein genug für eine möglichst schnelle Erkennung. Deshalb kommen nur Werte im Bereich von 50 bis 150 in Frage.

Algorithmus 5.1 : IBL-DS Update

Input : Fallbasis I, neue Instanz x_0
Output : aktualisierte Fallbasis I

Klassifiziere x_0 und aktualisiere Fehlerrate p und Standardabweichung s der letzten h Instanzen ;
aktualisiere $\overline{d}$, den durchschnittlichen Abstand des k_{test} nächsten Nachbarn der letzten h Instanzen ;
repeat
  $C \leftarrow k_{sim}$ nächsten Nachbarn von x_0 in I ;
  $T \leftarrow k_{test}$ nächsten Nachbarn von x_0 in I ;
  $d_T \leftarrow$ Abstand des k_{test} nächsten Nachbarn ;
  $N \leftarrow \{x_0\} \cup k_{new}$aktuellsten Instanzen aus T;
  $c \leftarrow$ häufigste Klasse in N;
  $p_1 \leftarrow$ relativer Anteil der häufigsten Klasse in N;
  $p_2 \leftarrow$ relativer Anteil der zweit häufigsten Klasse in N;
  if $p_1 = 1 \wedge |\{x \in T | c_x = c\}| \geq (k_{test} - 1) \wedge d_T \leq \overline{d}$ **then**
    $I \leftarrow I \setminus \{x \in C | x \notin N\}$
  else
    if $c_{x_0} = c \wedge p_1 - p_2 > \frac{1}{m+1}$ **then**
      $I \leftarrow I \setminus \{x \in C | c_x \neq c \wedge x \notin N\}$
    end
  end
  if $|I| =$*maxSize* **then**
    $I \leftarrow I \setminus \{$älteste Instanz in $C\}$
  end
until I *unverändert*;
$I \leftarrow I \cup \{x_0\}$;
if $p < p_{min}$ **then**
  $p_{min} \leftarrow p, s_{min} \leftarrow s$
else
  if $p > q \wedge p + s > p_{min} + z_{1-\alpha} s_{min}$ **then**
    $p_{last} =$ Fehlerrate der letzten h_{last} Instanzen **if** $p_{last} > \frac{1}{m+1}$ **then**
      $p_{dif} \leftarrow 1$
    else
      $p_{dif} \leftarrow p_{min} - p_{last}$
    end
    Lösche $min(|I| p_{diff}, |I| - k_{test})$ alte Instanzen gleichverteilt aus I.;
    Reset p_{min} und s_{min} mit p_{last};
  end
end

Parameter	Defaultwert
k_{new}	5
k_{test}	50
α	0.001
q	0.25
h	100
h_{last}	20

Tabelle 5.2: Defaultwerte der Parameter

- h_{last}: Anzahl der verwendeten letzten Trainingsdaten zur Schätzung der Klassifikationsrate, nachdem Concept Shift erkannt wurde. Dieser Wert muss möglichst klein sein, damit möglichst nur Trainingsdaten nach dem Concept Shift berücksichtigt werden, allerdings sind Werte kleiner als 20 kaum robust genung.

Die Tabelle 5.2 zeigt die verwendeten Defaulteinstellungen. Diese Parameter aufgrund der verwendeten Datenströme zu optimieren hätte nur einen geringen Nutzen. Zunächst ist unklar, inwieweit die Ströme, auch wenn sie sehr unterschiedliche Eigenschaften haben, in ihrer Art und Zusammenstellung möglichen Anwendungsszenarien entsprechen. Außerdem sind die Parameter konstruktionsbedingt der Art, dass sie in einem gewissen Intervall relativ robust zu einem guten Verfahren führen. Diese Robustheit der Parameter zu überprüfen ist viel sinnvoller (vgl. Abschnitt 5.4.5).

Implementierung Das Verfahren wurde in Java und mit Hilfe der Data-Mining-Bibliothek WEKA [WF05] implementiert. Diese Bibliothek enthält verschiedene Verfahren des überwachten und unüberwachten Lernens und zahlreiche Tools zur Vor- und Nachverarbeitung sowie zur visuellen Darstellung. Für den Algorithmus wurde das Interface *UpdateableClassifier* erweitert, dass schon für inkrementelle Verfahren vorgesehen ist. Neue Trainingsdaten können hierbei über eine *update*-Funktion an den Klassifikator zum Update seines Modells übergeben werden. Mittels der *classifyInstance*-Funktion können jederzeit Instanzen mit dem aktuell gelernten Modell klassifiziert werden. Beide Funktionen lassen sich dabei jederzeit in beliebiger Reihenfolge aufrufen.

Für die Implementierung der Datenströme wurden ebenso Klassen für statische Datensätze aus WEKA erweitert. Die Trainings- und Testdaten werden dabei nach einem vorgegebenen Modell generiert. Um sicherzustellen, dass alle Verfahren mit exakt den gleichen Datenströmen und Daten aufgerufen werden, lassen sich die Datenstrom-Generatoren mit einem vorgegebenen *seed*-Wert für den Zufallsgenerator initialisieren.

M-Tree Typischerweise wird zur Optimierung des Nearest-Neighbor-Verfahren ein kd-Tree benutzt [Ben75]. Dieser hat allerdings den Nachteil, dass nur numerische Attribute

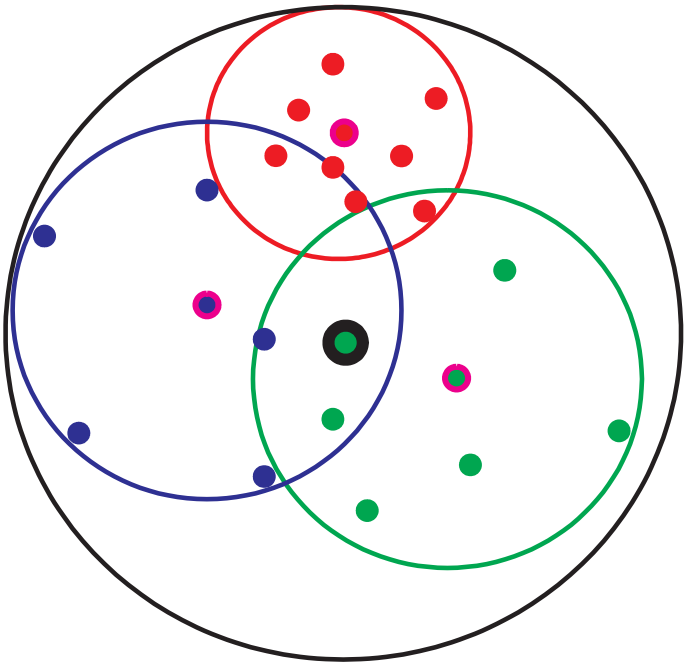

Abbildung 5.8: Einfaches Beispiel eines M-Tree

und nur bestimmte Metriken verwendet werden können. Deshalb wird hier zur Speicherung der Fallbasis ein M-Tree [CPRZ97] benutzt. Dieser benötigt nur eine Metrik auf den Daten, für die die Dreiecksungleichung gilt, und ist ansonsten für beliebige Arten von Daten verwendbar. Die Blätter eines M-Tree sind Instanzen. Innere Knoten kombinieren Unterknoten zu größeren Bereichen und die Wurzel repräsentiert die gesamte Datenmenge. Jeder Knoten n wird durch eine Instanz c_n als Zentrum, einen Radius r_n und eine Liste von Unterknoten s_n repräsentiert. Alle Daten, die in den Unterknoten gespeichert sind, haben einen Abstand zu dem Zentrum, der maximal diesem Radius entspricht. Ein Blatt hat einen Radius von 0, da es genau einer Instanz entspricht. Die Anzahl der Instanzen oder Unterknoten ist durch ein Intervall [*minCapacity*, *maxCapacity*] begrenzt. In dem Verfahren wird dabei das Intervall $[6, 15]$ benutzt, da es in den Experimenten zu den besten Ergebnissen führte. Ein einfaches Beispiel eines M-Tree ist in Abbildung 5.8 dargestellt.

Die ursprünglichen Anforderungen des M-Tree bei der Entwicklung sind allerdings etwas anders als in dieser Anwendung benötigt. Der M-Tree wurde für die externe Speicherung einer sehr hohen Zahl von Daten entwickelt. Hierbei wird allerdings eine Hauptspeicher-Datenstruktur benötigt für effizientes Einfügen, Löschen und Berechnung der nächsten Nachbarn. Außerdem soll die Datenstruktur schon im Bereich von 1000 Daten eine effizientere Berechnung der nächsten Nachbarn ermöglichen als eine einfache Liste. Dazu musste der M-Tree deutlich schlanker und effizienter implementiert werden als die ursprüngliche Version der Entwickler. Als Grundlage hierfür diente die

M-Tree-Implementierung aus der Java-Bibliothek XXL (siehe Abschnitt 4.4.7), die der Originalvariante entspricht. Das Ziel wurde vor allem durch eine schlankere Modellierung erreicht, die mit zwei Klassen für Baum und Knoten auskommt, sowie einer neuen effizienteren Splitfunktion, die einen Split in $O(n)$ Zeit durchführen kann (anstatt $O(n^2)$, wobei n die Anzahl Unterknoten im zu splittenden Knoten ist). Die Verwendung der von den Entwicklern vorgeschlagenen Optimierungen (z.B. die Verwendung von Matrizen mit vorberechneten Distanzen) sind durch die hohe Fluktuation von Daten in dieser Anwendung leider nicht hilfreich.

Die nächsten Nachbarn in einem M-Tree lassen sich mit einem durchschnittlichen Aufwand von $O(n \log n)$ effizient berechnen. Diese werden iterativ bestimmt, so dass es im Voraus nicht notwendig ist, die Zahl der benötigten nächsten Nachbarn zu kennen. Für die Berechnung wird ein Min-Heap H verwendet, der eine Menge von Knoten speichert. Der Heap-Wert eines Knoten, den es zu minimieren gilt, ist der Abstand zu der Instanz, dessen nächste Nachbarn gesucht werden. Der Abstand eines Knoten n und einer Instanz i ist dabei definiert durch $\max(0, ||i - c_n|| - r_n)$ und ist genau dann 0, wenn sich die Instanz innerhalb des Radius um das Zentrum des Knotens befindet. Der Heap wird vor der Berechnung des ersten nächsten Nachbarn mit dem Wurzelknoten des M-Tree initialisiert. Die Berechnung eines nächsten Nachbarn ist in Algorithmus 5.2 dargestellt.

Algorithmus 5.2 : Nächste-Nachbar-Berechnung im M-Tree

```
Input : Heap H
Output : nächster Nachbar n

while not(isEmpty(H)) do
    n=POP(H);
    if isLeaf(n) then
        return n
    else
        H.insertAll(s_n)
    end
end
```

Einfache Tests mit 1000 zufällig erzeugten Daten (10 numerische Dimensionen mit euklidischer Distanz) zeigte, dass die Zahl der notwendigen Distanzberechnungen zur Bestimmung der fünf nächsten Nachbarn durch den neuen Split nicht signifikant von der ursprünglichen Splitvariante abweicht. Zugleich ist der zeitliche Aufwand für die Berechnung der fünf nächsten Nachbarn mit dem M-Tree geringer als bei Verwendung einer einfachen Liste, wobei die Anzahl der Distanzberechnungen sogar deutlich geringer ist.

Distanzfunktion Als Distanzfunktion wurde eine Variante von SVDM benutzt, wobei es sich um eine vereinfachte Version des VDM-Distanzmaßes (Value Difference Me-

tric [SW86]) handelt, die sehr erfolgreich in dem regelbasierten Klassifikationsalgorithmus RISE [Dom95,Dom96,Dom97] zum Einsatz kam. Der Vorteil hierbei ist, dass sowohl numerische wie auch nominale Attribute kombiniert werden können. Diese Funktion wurde hier nun zusätzlich für die Behandlung von fehlenden Werten erweitert.

Für einen Datensatz mit l Attributen $A_1, \ldots, A_l$ sei eine Instanz x als l-dimensionaler Vektor wie folgt definiert:

$$x = (x_1, x_2, \ldots, x_l) \in D_1 \times D_2 \times \ldots \times D_l,$$

wobei D_i der Wertebereich des Attributs A_i ist. Der Wertebereich eines numerischen Attributs A_i wird dabei mit $D_i = [0, 1]$ als normalisiert angenommen. Ein Attributswert a_i wird dabei durch die Abbildung

$$a_i \mapsto \frac{a_i - \min_i}{(\max_i - \min_i)}$$

normalisert, wobei $\max_i$ und $\min_i$ die bisher größten und kleinsten aufgetretenen Werte des Attributs A_i sind. Der Abstand zweier normalisierter Attributswerte a_i und a'_i ist durch den einfachen euklidischen Abstand definiert:

$$\delta_i(a_i, a'_i) = |a_i - a'_i|$$

Falls der Attributswert für mindestens eine der Instanzen fehlt, ist der Abstand mit $\delta_i(a_i, a'_i) = 0$ festgelegt.

Für ein diskretes Attribut A_j berechnet sich die Distanz zwischen zwei Ausprägungen x_j und x'_j mit folgender Funktion:

$$\delta_j(a_j, a'_j) = \sum_{k=1}^{m} |P(c_x = k|x_j = a_j) - P(c_x = k|x_j = a'_j)|,$$

wobei m die Anzahl der Klassen ist. Die Wahrscheinlichkeit $P(c_x = k|x_j = a_j)$ wird durch die relative Häufigkeit von Instanzen der Klasse k unter allen Instanzen, die für das Attribut A_j den Wert $x_j = a_j$ annehmen, geschätzt. Falls der Attributswert a'_j unbekannt ist, wird dafür die Klassenverteilung über Instanzen verwendet (analog bei unbekanntem a_j):

$$\delta_j(a_j, ?) = \sum_{k=1}^{m} |P(c_x = k|x_j = a_j) - P(c_x = k)|.$$

Für den Sonderfall, dass beide Attributswerte unbekannt sind, kommt folgender gemittelter Abstand zur Verwendung:

$$\delta_j(?, ?) = \frac{1}{|A_i|} \sum_{a \in A_i} \delta_j(a, ?).$$

Die komplette Distanz zwischen zwei Instanzen x und y ist dann gegeben durch:

$$d(x, y) = \frac{1}{l} \sum_{i=1}^{l} \delta_j(x_j, y_j)^2.$$

Da die Distanzfunktion von der Fallbasis abhängt, verändert sie sich im Laufe der Zeit. Dabei entsteht ein Konflikt mit der Indexstruktur M-Tree, da sich damit auch die Radien der einzelnen Knoten verändern. Für eine Aktualisierung der Distanzfunktion müssten die Radien aller Knoten neu berechnet werden. Deshalb wird die Distanzfunktion nur dann aktualisiert, falls sich der maximale Abstand eines numerischen Attributs oder der Abstand zwischen zwei Ausprägungen eines nominalen Attributs um mindestens 0.1 ändert. Zusätzlich wird diese Bedingung nur nach jeweils 100 neuen Trainingsdaten überprüft. Auf eine Neuberechnung der Splits wird allerdings verzichtet, da sich diese, wenn überhaupt, nur gering ändern würden. Desweiteren passen sich die Splits durch Einfügen und Löschen mit der Zeit von selbst an.

5.4 Evaluation

Wie bereits in [BH07] diskutiert wurde, ist die experimentelle Validierung von adaptiven Lernern aus mehreren Gründen problematisch. Zunächst ist die Evaluation von Algorithmen auf Datenströmen offensichtlich schwieriger als bei statischen Daten. Die Klassifikationsrate variiert über die Zeit und entspricht damit einer Funktion, die sich nicht auf direkte Weise mit der eines anderen Verfahrens vergleichen lässt. Außerdem spielen zusätzliche Effekte, wie Concept Drift und Datenraten von Trainings- und Anfrageströmen, eine wichtige Rolle. Leider existieren zur Zeit auch noch keine Real-World- und Benchmark-Datenströme, um verschiedene Verfahren auf einheitlichen Daten zu vergleichen.

Aus diesen Gründen werden hier hauptsächlich synthetische Daten verwendet, die deutliche Vorteile bei der Evaluation haben. Experimente lassen sich deutlich kontrollierter durchführen, da das wahre Konzept jederzeit bekannt ist. Es ist möglich, genau zu messen und zu vergleichen, wann und wie ein Verfahren auf Concept Drift reagiert. Die Klassifikationsrate ist zu jedem Zeitpunkt berechenbar und damit werden verschiedene Verfahren jederzeit vergleichbar. Man hat zusätzlich die Möglichkeit, zahlreiche unterschiedliche Arten von Concept Drift zu testen und dabei Eigenschaften wie die Datenrate variieren zu lassen. Im Folgenden werden ausführliche Tests mit synthetischen Daten durchgeführt und anschließend noch ein Experiment mit Real-World Daten vorgestellt.

Bei den Tests wird der vorgestellte Ansatz mit folgenden instanzbasierten Verfahren verglichen (für Details zu den Verfahren siehe Abschnitt 3.4.4):

- Standard Sliding Window Ansatz mit fester Fenstergrößen von 200, 400 und 800, die mit Win200, Win400 und Win800 bezeichnet werden.

- Local Weighed Forgetting mit den Parametern $\beta = 0.04$ und $\beta = 0.02$ (LWF04, LWF02).

- Time Weighed Forgetting mit den Gewichten $w = 0.996$ und $w = 0.998$ (TWF996, TWF998).

Die Klassifikation mittels k-NN wird in allen Verfahren mit $k = 5$ ausgeführt. Für den vorgestellten Algorithmus wird dabei ebenfalls $k_{sim} = 5$ gewählt und eine Maximalgröße der Fallbasis von 5000 Beispielen. Hierbei ist zu beachten, dass es nicht um die Optimierung einzelner Methoden, sondern um den Vergleich der Verfahren unter möglichst gleichen Bedingungen geht, weshalb die Größe des Parameters k immer konstant gehalten wurde.

5.4.1 Datensätze

Für die Evaluation wurden acht Datenstrom-Typen mit verschiedenen Eigenschaften verwendet: GAUSS, SINE2, DISTRIB, RANDOM, STAGGER, MIXED, HYPERPLANE, MEANS. Der STAGGER Datenstrom wurde in [SRG86] präsentiert und wurde später auch bei der Evaluation der FLORA-Systeme [WK92, WK93, WK96] hauptsächlich verwendet. Die Ströme GAUSS, SINE2, STAGGER, MIXED wurden in [KW95, GMCR04] verwendet und HYPERPLANE wurde in mehreren Experimenten angewendet [DH00, HSD01, WFYH03]. DISTRIB, RANDOM und MEANS sind bisher noch nicht verwendete Datenstrom-Typen mit neuen Eigenschaften. Die Datenstrom-Typen werden nun im Einzelnen beschrieben:

- GAUSS: Dies ist ein binäres Klassifikationsproblem im $\mathbb{R}^2$ mit zwei normalverteilten Klassen, eine mit Standardabweichung 1 um $(0, 0)$ und die zweite mit Standardabweichung 4 um $(2, 0)$ (Kovarianzen sind 0). Nach jeweils 2000 Instanzen werde die Klassen vertauscht und damit Concept Shift erzeugt.

  Das Besondere an diesem Datensatz ist, dass sich die Klassen überschneiden. Genau genommen können überall Daten beider Klassen auftreten, allerdings mit unterschiedlicher Wahrscheinlichkeit. Die Kunst des Lernverfahrens muss dabei sein, die Bereiche zu ermitteln, in denen jeweils eine Klasse wahrscheinlicher Auftritt als die andere. Hinzu kommt zusätzlich der Concept Shift, den es zu erkennen gilt. Durch die starke Überlappung sind keine sehr hohen Klassifikationsraten zu erwarten.

- SINE2: Die Daten sind gleichverteilt in $[0,1] \times [0,1]$. Die beiden Klassen werden durch die Funktion $y = 0.5 + 0.3 \cdot \sin(3\pi x)$ getrennt. Nach jeweils 2000 Instanzen wird die Klassifikation wieder vertauscht.

  Bei diesem Datensatz sind die einzelnen Klassen klar voneinander getrennt. Allerdings verläuft die Grenze sehr geschwungen. Die Schwierigkeit liegt hierbei darin, die Spitzen der Ausbuchtungen richtig zu klassifizieren und nach jedem Concept Shift schnell wieder richtig zu erkennen.

- DISTRIB: Die Daten sind gleichverteilt in $[0,1] \times [0,1]$. Eine Instanz (x,y) gehört zu Klasse 1 falls $(x,y) \in [0,0.5] \times [0,0.5]$ oder $(x,y) \in [0.5,1] \times [0.5,1]$ gilt, ansonsten gehört sie zu Klasse 0. Hierbei ändert sich das zugrunde liegende Konzept nicht, sondern die Verteilung der Trainingsdaten. Sie werden jeweils nur in einem Viertel des Datenraums generiert, wobei das Viertel nach jeweils 2000 Daten im Uhrzeigersinn wechselt.

- RANDOM: Wie vorher auch, sind die Daten in $[0,1] \times [0,1]$ gleichverteilt. Eine Instanz (x,y) gehört mit einer Wahrscheinlichkeit von p zur Klasse 1, unabhängig von x und y. Innerhalb eines Zeitintervalls mit 2000 Trainingsdaten steigt p linear von 0 nach 1. Anschließend wechselt p im gleichen Zeitraum wieder von 1 nach 0.

- MIXED: Die Instanzen sind durch zwei boolesche Variablen v, w und zwei numerische Attribute $x, y \in [0,1]$ beschrieben. Eine Instanz wird positiv klassifiziert, wenn sie wenigstens zwei der folgenden Bedingungen erfüllt: $v = \texttt{true}, \mathrm{w} = \texttt{true}, \mathrm{y} < 0.5 + 0.3\sin(3\pi\mathrm{x})$. Nach jeweils 2000 Daten wird die Klassifikation vertauscht.

- STAGGER: Instanzen werden durch drei symbolische Attribute beschrieben: size (mit den Ausprägungen: small, medium und large), color (red, green) und shape (circular, non-circular). Im ersten Kontext werden genau die Instanzen, die die Bedingung (size=small)$\wedge$(color=red) erfüllen, als positiv klassifiziert (Klasse 1). Im zweiten Kontext lautet die Konzeptbeschreibung (color=green)$\vee$(shape=circular) und das dritte Konzept wird durch die Bedingung (size=(medium$\vee$large)) beschrieben. Das Konzept wechselt dabei nach jeweils 2000 Instanzen und beginnt nach dem dritten Konzept wieder mit dem ersten.

- HYPERPLANE: Der Instanzenraum $[0,1]^d$ wird durch eine sich drehende Hyperebene in zwei Klassen getrennt. Die Hyperebende wird definiert durch:

$$\sum_{i=1} dw_i x_i = w_0.$$

  Bei Ankunft einer neuen Instanz werden die Gewichte $w_i, 1 \leq i \leq d$ wie folgt verändert:

$$w_i \leftarrow w_i + f_i \frac{d}{5000},$$

	Attribute	Klassen	Drift
GAUSS	2 numerische	2 (überlappend)	Concept Shift
SINE2	2 numerische	2	Concept Shift
DISTRIB	2 numerische	2	Sampling Shift
RANDOM	2 numerische	2	Drift der Verteilung
MIXED	2num.+2diskr.	2	Concept Shift
STAGGER	3 diskrete	2	Concept Shift
HYPER2/5	2/5 num.	2	Concept Drift
MEANS2/5	2/5 num.	5	Concept Drift
MEANS15	15 num.	2	Concept Drift

Tabelle 5.3: Vergleich der Datenströme

wobei $f_i \in \{-1, 1\}$ alle 2000 Elemente zufällig neu gewählt wird. Damit beide Klassen immer gleich groß bleiben, wird w_0 jeweils mit $w_0 = 0.5 \sum_{i=0}^{d} w_i$ neu berechnet. Am Anfang werden die w_i aus dem Intervall $[0, 1]$ zufällig und gleichverteilt initialisiert. Die Experimente wurden mit $d = 2$ und $d = 5$ durchgeführt.

- MEANS: Die n Klassen werden durch n Zentren in $[0, 1]^d$ definiert. Eine Instanz gehört dabei zur Klasse des nächsten Zentrums. Jedes Zentrum verschiebt sich um einen festen Drift für jede Dimension. Falls das Einheitsintervall für eine Dimension überschritten wird, wird der Drift für diese Dimension invertiert. Die Experimente wurden für $n = 5$ und $d \in \{2, 5\}$ durchgeführt. Für jede Dimension wurde der Drift mit einem zufälligen Wert aus dem Intervall $[-5{\cdot}10^{-5}, 5{\cdot}10^{-5}]$ initialisiert.

Die Tabelle 5.3 stellt einige Eigenschaften der Datenstrom-Typen gegenüber. Für jeden Datenstrom-Typ wurde ein Generator implementiert, dessen Zufallsgenerator mit einem vorgegebenen Seed initialisiert wird. Durch die Wahl des Seed ist damit ein gesamter Datenstrom eindeutig festgelegt. Bei der Generierung der einzelnen Datenströme wurde den Daten jeweils 5% zufälliges Rauschen hinzugefügt. Dies bedeutet, dass 5% der Trainingsdaten falsch klassifiziert sind, wobei die Wahl der Daten und die Wahl der Klasse zufällig und gleichverteilt sind. Mit jedem Datenstrom-Generator wurden 200 konkrete Datenströme mit einer Länge von 20000 Instanzen erzeugt und den Lernverfahren als Eingabe übergeben. Zur Initialisierung der Verfahren dienten jeweils die ersten 100 Instanzen.

5.4.2 Klassifikationsraten

Für alle Datensätze und alle Verfahren wurde die mittlere absolute Klassifikationsrate und die Klassifikationsrate der Trainingsdaten sowie deren Standardabweichungen berechnet. Die Ergebnisse sind in den Tabellen 5.4 und 5.5 dargestellt. Für die Datenströme GAUSS,

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	**.840**	.806	.838	.780	.755	.700	.758	.708
	(.0017)	(.0017)	(.0016)	(.0018)	(.0017)	(.0016)	(.0017)	(.0016)
SINE2	**.926**	.868	.901	.869	.842	.772	.843	.780
	(.0017)	(.0014)	(.0012)	(.0018)	(.0018)	(.0017)	(.0017)	(.0017)
DISTRIB	**.906**	.894	.877	.506	.511	.521	.512	.521
	(.0079)	(.0050)	(.0059)	(.0044)	(.0059)	(.0072)	(.0056)	(.0070)
RANDOM	.708	.706	**.718**	.663	.655	.625	.658	.629
	(.0019)	(.0017)	(.0013)	(.0016)	(.0016)	(.0018)	(.0016)	(.0018)
MIXED	**.922**	.863	.895	.872	.845	.775	.846	.783
	(.0019)	(.0016)	(.0033)	(.0017)	(.0016)	(.0016)	(.0016)	(.0016)
STAGGER	**.971**	.509	.509	.910	.888	.864	.897	.876
	(.0027)	(.0010)	(.0011)	(.0039)	(.0064)	(.0085)	(.0048)	(.0077)
HYPER2	.957	**.974**	.970	.924	.927	.924	.930	.927
	(.0056)	(.0092)	(.0049)	(.0042)	(.0079)	(.0149)	(.0077)	(.0145)
HYPER5	.865	.883	**.892**	.828	.835	.825	.836	.829
	(.0048)	(.0208)	(.0110)	(.0057)	(.0112)	(.0216)	(.0111)	(.0214)
MEAN2	.934	**.946**	.931	.894	.909	.910	.910	.912
	(.0044)	(.0043)	(.0047)	(.0042)	(.0041)	(.0062)	(.0041)	(.0060)
MEAN5	.794	**.830**	.799	.703	.737	.764	.738	.766
	(.0077)	(.0060)	(.0071)	(.0070)	(.0063)	(.0059)	(.0064)	(.0059)

Tabelle 5.4: Absolute Klassifikationsrate mit Standardabweichung

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	**.804**	.774	.803	.751	.728	.679	.731	.686
	(.0033)	(.0031)	(.0029)	(.0038)	(.0036)	(.0036)	(.0034)	(.0034)
SINE2	**.884**	.833	.862	.833	.809	.747	.810	.753
	(.0025)	(.0022)	(.0022)	(.0031)	(.0029)	(.0028)	(.0029)	(.0028)
DISTRIB	.931	.943	**.945**	.902	.901	.899	.903	.902
	(.0023)	(.0017)	(.0017)	(.0031)	(.0030)	(.0033)	(.0031)	(.0033)
RANDOM	.709	.706	**.718**	.663	.654	.625	.658	.629
	(.0038)	(.0033)	(.0035)	(.0040)	(.0036)	(.0036)	(.0036)	(.0035)
MIXED	**.878**	.826	.855	.834	.809	.746	.810	.753
	(.0030)	(.0023)	(.0039)	(.0030)	(.0030)	(.0028)	(.0029)	(.0028)
STAGGER	**.925**	.507	.507	.870	.850	.829	.858	.840
	(.0031)	(.0032)	(.0033)	(.0042)	(.0064)	(.0079)	(.0050)	(.0073)
HYPER2	.912	**.927**	.924	.882	.885	.882	.887	.885
	(.0057)	(.0086)	(.0051)	(.0049)	(.0078)	(.0138)	(.0076)	(.0135)
HYPER5	.828	.845	**.853**	.795	.801	.792	.803	.796
	(.0049)	(.0189)	(.0102)	(.0059)	(.0103)	(.0196)	(.0101)	(.0194)
MEAN2	.900	**.911**	.896	.862	.876	.877	.877	.879
	(.0050)	(.0046)	(.0051)	(.0048)	(.0049)	(.0069)	(.0050)	(.0067)
MEAN5	.766	**.800**	.771	.679	.712	.738	.712	.739
	(.0080)	(.0062)	(.0074)	(.0075)	(.0069)	(.0065)	(.0069)	(.0065)

Tabelle 5.5: Klassifikationsrate der Trainingsdaten mit Standardabweichung

SINE2, MIXED und STAGGER zeigt IBL-DS die beste Performanz für beide Maße, bei DISTRIB nur für das absolute Maß. Allerdings gilt auch für alle anderen Ergebnisse, dass IBL-DS gute Ergebnisse liefert, die nicht weit vom Optimum entfernt sind.

IBL-DS zeigt bei Concept Shift besonders gute Ergebnisse, die verwendete Strategie zur Erkennung von Concept Shift mit flexibler Größe der Fallbasis scheint also auch praktisch gut zu funktionieren. Einige andere Verfahren scheinen dagegen besonders anfällig für solche abrupten Änderungen zu sein, was die zum Teil schlechten Klassifikationsraten vermuten lassen. Dies liegt natürlich auch daran, dass die anderen Verfahren die Fallbasis unabhängig von dem tatsächlich vorhandenen Concept Drift anpassen.

Eine besondere Rolle spielt STAGGER, da der Datenraum nur 12 verschiedene Ausprägungen für die Daten beinhaltet, was in den wenigen diskreten Attributen begründet liegt. Dadurch ist es nicht sinnvoll, viele Daten zu speichern, obwohl sich das Konzept über relativ lange Zeit nicht ändert, um schneller bei Concept Shift reagieren zu können. Dies belegen auch die Ergebnisse der Vergleichsverfahren, da die Ergebnisse in den Fällen besser sind, bei denen die Fallbasis durch die Wahl der Parameter kleiner gehalten wird. LWF kommt mit diesem Datenstrom überhaupt nicht zurecht.

Der extreme Unterschied zwischen den beiden Maßen für die Klassifikationsrate beim DISTRIB-Datenstrom ist in der extremen Verteilung der Trainingsdaten begründet. Nach einem Shift dauert es 6000 Zeitschritte mit neuen Instanzen bis wieder eine Trainingsinstanz im selben Abschnitt des Datenraums erscheint. Alle fensterbasierten Verfahren haben bis dahin längst Trainingsdaten aus diesem Abschnitt aus der Fallbasis gelöscht. Dadurch können Anfragen aus diesem Bereich des Datenraums nicht sinnvoll beantwortet werden, was sich in der absoluten Klassifikationsrate widerspiegelt. Nur LWF und IBL-DS speichern über die komplette Zeit Daten aus allen Bereichen, wobei IBL-DS die bessere Performanz für die absolute Klassifikationsrate hat.

Bei den HYPER und MEANS Datenströmen tritt dagegen nur relativ gleich bleibender Concept Drift auf. Hierbei haben die anderen Verfahren den Vorteil, dass sie gerade von dieser Annahme ausgehen. Bei jedem der Verfahren gilt, dass sich die optimalen Parameter für einen Datenstrom kaum über die Zeit ändern. Außerdem kann es nicht versehentlich zu einem fälschlicherweise erkannten Concept Shift kommen. Deshalb sind mit guter Vorauswahl der Parameter hohe Klassifikationsraten zu erwarten.

Zum Vergleich zeigt Tabelle 5.6 die absoluten Klassifikationsraten bei 40mal schnellerem Concept Drift (technisch wurde nur jede 40. Instanz zum Lernen benutzt). Abgesehen von DISTRIB, ohne echten Concept Drift, und STAGGER, mit dem Sonderfall von nur 12 verschiedenen Ausprägungen und eine hohe Überlappung der unterschiedlichen Konzepte, dominiert IBL-DS alle anderen Verfahren. Dies ist auf die für den hohen Concept Drift nicht optimierten Parameter und damit der Größen der Instanzbasen zurückzuführen und zeigt den Hauptvorteil von IBL-DS gegenüber den anderen Verfahren, die Concept Drift ohne direkte Erkennung behandeln.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	**.609**	.515	.555	.502	.502	.502	.504	.504
SINE2	**.521**	.493	.488	.502	.502	.502	.503	.502
DISTRIB	.938	**.965**	.950	.909	.921	.929	.922	.932
RANDOM	**.568**	.501	.504	.500	.500	.500	.501	.501
STAGGER	**.697**	.477	.477	.567	.530	.568	.532	.604
MIXED	**.539**	.508	.513	.502	.502	.502	.504	.504
HYPER2	**.848**	.774	.812	.757	.716	.681	.720	.689
HYPER5	**.692**	.639	.668	.630	.599	.581	.603	.587
MEANS2	**.777**	.553	.683	.660	.507	.368	.518	.395
MEANS5	**.628**	.478	.600	.609	.505	.371	.514	.396

Tabelle 5.6: Absolute Klassifikationsrate bei 40mal schnellerem Concept Drift.

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
absolute	.784	.835	.823	.722	.738	.753	.740	.755
Rate	(.0092)	(.0068)	(.0074)	(.0138)	(.0127)	(.0117)	(.0126)	(.0116)
absolute Rate	.687	.615	.674	.694	.651	.582	.656	.597
bei 40fachem Drift	(.0096)	(.0157)	(.0147)	(.0056)	(.0126)	(.0186)	(.0127)	(.0182)

Tabelle 5.7: Klassifikationsraten für höher dimensionale Daten
Angewendet wurde der MEANS-Datenstrom mit 15 Dimensionen und 2 Klassen.

5.4.3 Größe der Fallbasis

Wie schon erwähnt, scheint die Strategie von IBL-DS zur Behandlung von Concept Shift sehr gut zu funktionieren. Wie schnell das Verfahren reagiert, wird in Abbildung 5.9 anhand eines Beispiels gezeigt. Die obere Abbildung zeigt die Entwicklung der absoluten Klassifikationsrate und die Größe der Fallbasis zwischen der 10000. und 20000. Trainingsinstanz eines Datenstroms vom Typ SINE2. Die untere Abbildung beleuchtet den Shift um die 12000. Trainingsinstanz genauer. Wie zu sehen ist, nimmt die Größe der Fallbasis nach jedem Concept Shift sehr schnell ab. Nach nur 20 neuen Instanzen wird wieder eine Klassifikationsrate erreicht, die deutlich über 50% liegt.

Die durchschnittliche Größe der jeweils verwendeten Fallbasis ist in Tabelle 5.8 gegenübergestellt. Diese zeigt, dass die verwendete Fallbasis von IBL-DS im Schnitt meist größer ist, als die der besten Referenzverfahren. Bei den Datenströmen mit Concept Shift fällt dies dabei deutlich kleiner aus. Bei einem Concept Shift wird zwar die gesamte Fallbasis geleert, dafür steigt die Größe der Fallbasis in Zeiträumen ohne Shift deutlich an. Im Mittel liegt die Größe der Fallbasis dann leicht über den besten Referenzverfahren, die fast konstante Größen über die ganze Zeit besitzen. Bei den Datenströmen mit konti-

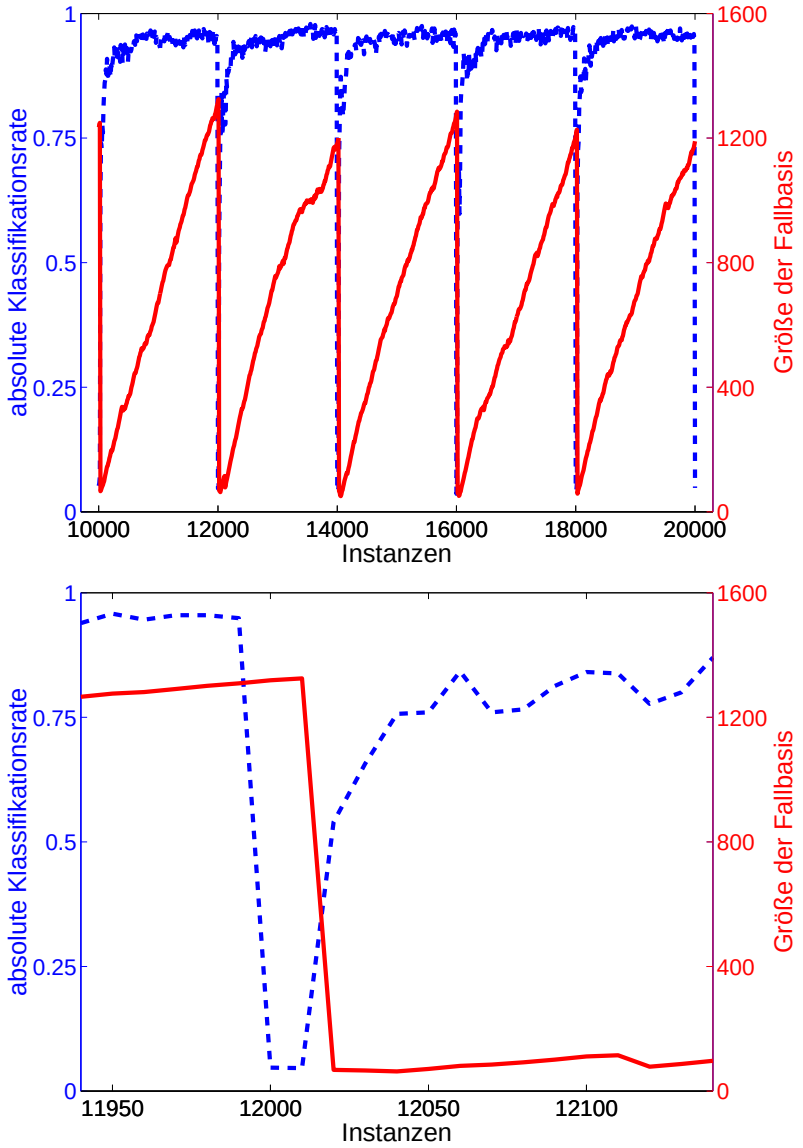

Abbildung 5.9: Vergleich von Klassifikationsrate und Größe der Fallbasis
Dieses Beispiel zeigt die absolute Klassifikationsrate (blau gestrichelt) und die Größe der Fallbasis (rot durchgehend) von IBL-DS beim SINE2-Datenstrom im Bereich der 10.000. und 20.000. (oben) bzw. um der 12.000. Trainingsinstanz (unten).

	IBL-DS	LWF02	LWF04	Win200	Win400	Win800	TWF996	TWF998
GAUSS	638	553	277	200	399	795	401	799
SINE2	659	550	275	200	399	795	401	799
DISTRIB	3513	548	274	200	399	795	401	799
RANDOM	508	550	275	200	399	795	401	799
MIXED	613	551	272	200	399	795	401	799
STAGGER	3405	3051	3051	200	399	795	401	799
HYPER2	3549	550	275	200	399	795	401	799
HYPER5	2745	886	433	200	399	795	401	799
MEAN2	2007	550	276	200	399	795	401	799
MEAN5	2433	888	432	200	399	795	401	799

Tabelle 5.8: Durchschnittliche Größe der Fallbasis

nuierlichem Concept Drift ist die Fallbasis bei IBL-DS deutlich größer als die der besten Referenzverfahren, obwohl die Klassifikationsrate leicht darunter liegt. Dies ist auch eine Konsequenz des kontinuierlichen Concept Drift, der sich viel schwieriger erkennen lässt als Concept Shift. Besonders auffällig sind auch die großen Fallbasen beim DISTRIB- und STAGGER-Datenstrom für IBL-DS. Beim DISTRIB-Datenstrom lässt sich das damit erklären, dass hier gar kein echter Drift vorhanden ist und so Daten hauptsächlich nur aufgrund von Redundanz gelöscht werden. Bei STAGGER ist dagegen die Tatsache verantwortlich, dass der auftretende Concept Shift jeweils nur einen Teil des Datenraums betrifft, und somit nie die gesamte Fallbasis gelöscht wird. Die Ergebnisse zeigen auch deutlich den Zusammenhang bei dem LWF-Verfahren zwischen Daten und Größe der Fallbasis. Hierbei ist die Verteilung der Distanzen zwischen den Objekten und ihren nächsten Nachbarn verantwortlich. Daher ist die Größe bei allen Datenströmen aus dem $\mathbb{R}^2$ fast identisch. Andere Distanzverteilungen treten in höheren Dimensionen auf, wie beim HYPER5- und MEAN5-Datenstrom und ebenso bei diskreten Attributen wie bei STAGGER. Insbesondere zeigt dies wiederum, dass die Größe der Fallbasis auch bei LWF komplett unabhängig vom tatsächlich auftretenden Concept Drift ist. Um so erstaunlicher sind die trotz allem relativ stabilen Klassifikationsraten bei fast allen Datenströmen.

5.4.4 Zeit für Updates und Klassifikation

Dieser Abschnitt vergleicht die Laufzeiten für Updates und Klassifikationen zwischen den Verfahren. Die Tabelle 5.9 zeigt die durchschnittlichen Zeiten für 1000 Updates von Trainingsdaten und die Tabelle 5.10 für 1000 Klassifikationen von neuen unklassifizierten Daten. Wie zu erwarten, erhöht sich der Aufwand durch das komplexere Updateverfahren von IBL-DS und da während eines Updates mehr als die Klassifikation der Trainingsinstanz nötig ist, sind natürlich auch Updates aufwändiger als Klassifikationen. Aber bei Update-Laufzeiten von unter 5 Sekunden für 1000 Daten sind immer noch sehr hohe Da-

	IBL-DS	LWF02	Win800	TWF998
GAUSS	.56	.17	.033	.035
SINE2	.36	.12	.023	.024
DISTRIB	.49	.12	.027	.028
RANDOM	.33	.12	.023	.024
MIXED	.63	.52	.050	.075
STAGGER	.70	.30	.032	.066
HYPER2	.50	.12	.023	.025
HYPER5	4.3	.78	.048	.055
MEANS2	.47	.12	.023	.025
MEANS5	3.2	.76	.048	.054

Tabelle 5.9: Laufzeit für 1000 Updates (in Sekunden)

	IBL-DS	LWF02	Win800	TWF998
GAUSS	.17	.13	.13	.14
SINE2	.090	.087	.063	.067
DISTRIB	.13	.087	.079	.081
RANDOM	.078	.088	.067	.066
MIXED	.16	.19	.078	.080
STAGGER	.19	.23	.072	.24
HYPER2	.13	.087	.063	.067
HYPER5	1.3	.61	.37	.40
MEANS2	.12	.089	.066	.067
MEANS5	.92	.59	.37	.40

Tabelle 5.10: Laufzeit für 1000 Klassifikationen (in Sekunden)

tenraten zu bewältigen.

5.4.5 Robustheit gegenüber Variation der Parameter

Eine sicherlich unschöne Eigenschaft von IBL-DS ist der stark heuristische Charakter verbunden mit einer Vielzahl von Parametern. Ein Grund hierfür liegt, wie später in der Diskussion noch erörtert wird, in der Wahl des instanzbasierten Verfahrens als Grundlage. Um diese Problematik der Parameterwahl zu verringern, wurden für alle Parameter bereits möglichst gute Default-Einstellungen vorgeschlagen. Bei der Anwendung des Verfahrens brauchen diese Parameter deshalb vom Benutzer nicht mehr beachtet zu werden. Wie das vorherige Experiment schon gezeigt hat, ist diese fixe Vorauswahl für unterschiedlichste Datenströme und Arten von Concept Drift geeignet. Dieser Teil der Evaluation soll zeigen, dass die Wahl der Parameter eher unkritisch und robust ist, geringere Änderun-

	Default	$k_{test} = 35$	$k_{test} = 70$	$k_{new} = 3$	$k_{new} = 10$	$\alpha = .05$	$\alpha = .0001$
GAUSS	.840	.839	.836	.764	.837	.840	.840
SINE2	.926	.927	.921	.863	.926	.924	.924
DISTRIB	.906	.904	.902	.882	.910	.905	.906
RANDOM	.708	.703	.711	.709	.704	.708	.708
MIXED	.922	.926	.912	.800	.920	.919	.919
STAGGER	.971	.963	.962	.969	.944	.971	.971
HYPER2	.957	.950	.956	.959	.947	.955	.955
HYPER5	.865	.865	.868	.861	.860	.867	.867
MEAN2	.934	.937	.940	.752	.926	.939	.939
MEAN5	.794	.793	.794	.609	.794	.795	.795

	$q = 0.1$	$q = 0.4$	$h = 50$	$h = 150$	$h_{last} = 10$	$h_{last} = 30$
GAUSS	.840	.832	.840	.835	.840	.840
SINE2	.924	.917	.925	.919	.925	.923
DISTRIB	.883	.906	.905	.905	.906	.906
RANDOM	.710	.702	.709	.711	.708	.709
MIXED	.920	.914	.922	.916	.919	.919
STAGGER	.967	.964	.960	.961	.959	.969
HYPER2	.956	.954	.955	.954	.955	.955
HYPER5	.875	.866	.873	.866	.867	.867
MEAN2	.940	.939	.940	.939	.940	.939
MEAN5	.772	.794	.758	.795	.794	.894

Tabelle 5.11: Absolute Klassifikationsrate für IBL-DS mit verschiedenen Parametern

gen der einzelnen Werte also nur zu geringen Änderungen der Klassifikationsrate führen. Zahlreiche Parameter werden jeweils nach oben und unten variiert. Die Tabelle 5.2 gibt eine Übersicht über die hierbei betrachteten Parameter mit Default-Einstellungen. Die Testdatenströme aus dem vorherigen Test werden jeweils mit allen Varianten klassifiziert. Wie die Ergebnisse der absoluten Klassifikationsrate in Tabelle 5.11 zeigen, variieren die Klassifikationsraten dabei nur geringfügig. Eine kleine Ausnahme bilden die Ergebnisse bei $k_{new} = 3$, die meist deutlich schlechter ausfallen. Hierbei repräsentieren nur die aktuellsten drei Instanzen der Testumgebung das aktuelle Modell. Dadurch können schon zwei benachbarte fehlerhafte Instanzen (Rauschen) für ein fehlerhaftes Modell in ihrer Umgebung sorgen. Diese Variante ist also sensibler für Rauschen, wodurch sich die Ergebnisse erklären lassen. Dagegen lässt sich zwischen der Default-Einstellung $k_{new} = 5$ und $k_{new} = 7$ kein signifikanter Unterschied der Ergebnisse erkennen.

Datensatz	Instanzen	Klassen	Attribute	ausgewähltes Attribut
Balance	625	3	4	right-distance (numerisch)
Car	1728	4	6	safety (diskret)
Nursery	11025	5	8	health (diskret)

Tabelle 5.12: Zu Datenströme umfunktionierte UCI-Datensätze

5.4.6 Real-World-Daten

Bisher wurden ausschließlich synthetische Daten verwendet, die eine flexible und umfangreiche Evaluation erlauben. Trotzdem ist die Anwendung auf Real-World-Daten ebenso wünschenswert. Wie schon öfter erwähnt, gibt es bislang keine veröffentlichten Real-World-Datenströme für das Klassifikations-Setting. Es existieren einige große UCI Zeitreihen, die allerdings zur Klassifikation ungeeignet sind. Aus diesem Grund wurden im ersten Experiment statische UCI-Datensätze auf einfache Weise um Concept Drift erweitert. Im zweite Experiment wurden selbst aufgezeichnete Aktienkurse verwendet.

Erstes Experiment Für dieses Experiment wurden statische Benchmark-Datensätze aus dem UCI-Repository [DNM98] ausgewählt und zu Datenströmen aufbereitet. Jeder Datensatz kann als Datenstrom oder zumindest als Abschnitt eines Datenstroms angesehen werden, wenn eine feste Reihenfolge der Daten festgelegt wird. Allerdings beinhalten die statischen Datensätze normalerweise keinen Concept Drift. Um Concept Drift zu simulieren, wurde ein Verfahren angewendet, das einem Verfahren aus [LZ05] ähnelt: Zunächst wird das für die Klasse relevanteste Attribut ermittelt mittels der „Correlation-based Feature Subset Selection“, die in WEKA implementiert ist. Anschließend wird der gesamte Datensatz nach diesem Attribut sortiert. Zuletzt wird dieses Attribut komplett gelöscht, wodurch der direkte Zusammenhang zwischen dem Attribut und der Klasse verloren geht. Dieser Zusammenhang besteht aber noch indirekt durch die Reihenfolge der Daten. Jedesmal wenn sich der Wert dieses „versteckten“ Attributs ändert, ändert sich der Zusammenhang der übrigen Attribute zu der Klasse. Dadurch entsteht im Falle eines diskreten Attributs Concept Shift und bei einem numerischen Attribut kontinuierlicher Concept Drift.

Ein geeigneter Datensatz sollte zunächst nicht zu klein sein und eine ausreichende Zahl an Klassen besitzen. Außerdem sollte kein Attribut mit dem ausgewählten Attribut hoch korrelieren, da sonst durch das Löschen des Attributes kaum Concept Drift entsteht. Diese Kriterien werden nur von ein paar wenigen UCI-Datensätzen erfüllt. Für das Experiment wurden die Datensätze „Balance“, „Car“ und „Nursery“ ausgewählt. Die wichtigsten Eigenschaften dieser Datensätze sind in Tabelle 5.12 zusammengefasst.

IBL-DS wurde hierbei mit den Defaulteinstellungen ausgeführt. Für die übrigen Verfahren wurden die Parameter optimiert. Für LWF wurden die Parameter $\beta = 0.04$ und

	Balance	Car	Nursery
IBL-DS	0.805	0.889	0.900
LWF04	0.782	0.700	0.899
LWF10	0.846	0.700	0.842
Win50	0.813	0.819	0.827
Win100	0.815	0.862	0.856
Win200	0.785	0.888	0.878
TWF99	0.795	0.889	0.877
TWF995	0.787	0.887	0.900
kNN	0.693	0.745	0.839

Tabelle 5.13: Ergebnisse für Datenströme aus UCI-Datensätze

Klasse:	Fallend	Unverändert	Steigend
Anteil:	37.3%	23.8%	38.9%

Tabelle 5.14: Verteilung der Klassen

$\beta = 0.10$ verwendet, für TWF wurde $weight = 0.99$ und $weight = 0.995$ gewählt und für die feste Fenstergröße des Sliding-Window-Ansatzes (Win) die Werte $size = 50, 100, 200$ verwendet. Um zu zeigen, dass tatsächlich Concept Drift vorhanden ist, wurde zusätzlich der k-Nearest-Neighbor (kNN) Klassifizierer mit $k = 5$ und ohne Erkennung von Concept Drift angewendet.

Die Tabelle 5.13 präsentiert die Klassifikationsraten für die drei Datenströme. Wie man sieht, sind die Ergebnisse von IBL-DS für diese unterschiedlichen Arten von Datenströmen und ohne Parameteränderungen sehr gut. Der k-Nearest-Neighbor-Lerner schneidet dagegen deutlich schlechter ab, was auch für die Existenz von Concept Drift spricht.

Zweites Experiment Für dieses Experiment wurden Aktienkurse von sieben verschiedenen Automobilherstellern vom 1.1.1990 bis zum 30.9.2006 verwendet. Jeder der täglichen Datensätze beschreibt die relativen Veränderungen der Kurse zum Vortag (in Prozent). Insgesamt hat der Datenstrom 4088 Einträge. Nun soll die Veränderung der VW-Aktie anhand der Veränderung der anderen Kurse vorhergesagt werden. Dazu wurden die Veränderungen der VW-Aktie in steigend ($\geq 0.5\%$), fallend ($\leq -0.5\%$) und unverändert eingeteilt. Um die Existenz von Concept Drift zu zeigen, wurde als Vergleich auch der einfache k-Nearest-Neighbor (kNN) Klassifizierer mit $k = 5$ angewendet. Die Daten sind sicherlich nicht gleichverteilt und enthalten sehr viel Rauschen. Die Verteilung der Klassen ist in Tabelle 5.14 dargestellt und die Ergebnisse in Tabelle 5.15.

Die Ergebnisse zeigen, dass eine Verbesserung des Verfahrens ohne Concept Drift (k-NN) zu erreichen ist. Allerdings schaffen das nicht die Verfahren mit einem einfachen

Verfahren	Trefferquote
IBL-DS	0.563
LWF02	0.569
LWF04	0.559
Win200	0.519
Win400	0.528
Win800	0.529
TWF996	0.520
TWF998	0.520
kNN	0.539

Tabelle 5.15: Ergebnisse für Datenstrom aus Aktienkursen

Zeitfenster. Hierfür ist vermutlich die extreme Verteilung der Daten verantwortlich. Größere Veränderungen der Kurse kommen nur selten vor, haben aber oft einen relativ großen Einfluss auf die anderen Kurse. Bei den Verfahren mit lokaler Optimierung der Fallbasis fallen die Ergebnisse deutlich besser aus. IBL-DS kann auch hier mit LWF konkurrieren, wobei durch Optimierung des Parameters von LWF die Trefferquote noch verbessert werden kann. Allerdings gilt auch hier, dass in der Praxis aufgrund des Online-Settings eine Optimierung der Parameter normal nicht möglich ist. Die durchschnittliche Größe der Fallbasis lag für IBL-DS bei 377 Instanzen, für LWF02 dagegen bei 860.

5.4.7 Qualitativer Vergleich

In diesem Unterabschnitt werden generelle Eigenschaften der einzelnen Verfahren gegenübergestellt. Die Kriterien betreffen die Flexibilität, Kontrolle und Adaptionskriterien der Fallbasis sowie der Umgang mit Concept Drift und Concept Shift. Diese sind in Tabelle 5.16 gegenübergestellt. Hierbei steht $+$ für „trifft zu“ und $-$ für „nicht zutreffend“. IBL-DS erfüllt hierbei alle Kriterien, da sie auch Grundlage für die Entwicklung des Verfahrens waren. Da die Vergleichsverfahren Concept Drift ohne direkte Erkennung behandeln, sind zahlreiche Kriterien nicht erfüllt. Ein besonderer Punkt ist die Festlegung der Größe der Fallbasis bei LWF. Hierbei kann man zwar durch den freien Parameter die Größe der Fallbasis indirekt festlegen, die genaue Größe ist dann aber von der Verteilung der Daten und der Distanzfunktion abhängig und kann fast nur durch Erfahrungswerte vorhergesagt werden.

	LWF	TWF	Win	IBL-DS
Flexibel anpassbare Größe der Fallbasis	−	−	−	+
Größe der Fallbasis festlegbar	±	+	+	+
Kontrollmöglichkeit der Verarbeitungszeit	−	−	−	+
Instanzauswahl hängt vom Alter ab	+	+	+	+
Instanzauswahl hängt von lokaler Umgebung ab	+	−	−	+
Instanzauswahl hängt von der Konsistenz ab	−	−	−	+
Schnelle Anpassung an Concept Shift	−	−	−	+
Kontinuierliche Anpassung Concept Drift	+	+	+	+

Tabelle 5.16: Qualitativer Vergleich der Lernalgorithmen

5.5 Variationen

Das vorgestellte Verfahren ist eine allgemeine Online-Erweiterung des einfachen instanzbasierten Ansatzes zur Klassifikation. Dieses lässt sich wie im statischen Fall für unterschiedliche Situationen und Anfragestellungen anpassen. Die erste Variation zeigt hierbei, wie sich das Verfahren auf einfache Weise zu einem adaptiven Regressionsverfahren verändern lässt. Der Unterabschnitt über Online-Feature-Selection diskutiert, inwieweit eine Online-Variante der Feature-Selection sinnvoll und möglich ist.

5.5.1 Regression

Während bei der Klassifikation aufgrund von Trainingsdaten eine diskrete Ausgabe für neue Daten bestimmt werden soll, ist das Ziel bei der Regression, eine numerisches Ausgabe c zu schätzen. Bei der Klassifikation mit k-Nearest-Neighbor wird aus den k Klassen der nächsten Elemente die vorhergesagte Klasse meistens durch Mehrheitsvotum oder durch Gewichtung aufgrund der Abstände bestimmt. Bei der k-Nearest-Neighbor-Variante für die Regression muss hingegen die numerische Ausgabe c einer neuen Instanz z durch (gewichtetes) Mitteln der Ausgabewerte der k nächsten Elemente geschätzt werden:

$$est(z) = \sum_{i=1}^{k} w_i c_{x_i}, \text{mit} \sum_{\mathrm{i=1}}^{\mathrm{k}} \mathrm{w_i} = 1,$$

wobei x_i der $i.$ nächste Nachbar von x ist und c_{x_i} der Wert der Ausgabe c von x_i ist. Zum Vergleich von Regressionsverfahren wird anstatt der Klassifikationsrate der durchschnittliche quadratische Fehler zum wahren Attributswert als Gütemaß benutzt. Für eine Menge von Testdaten $Z = \{z_1, \ldots, z_n\}$ berechnet sich dieser Fehlerwert durch:

$$error = \frac{1}{n} \sum_{i=1}^{n} (est(z_i) - c_{z_i})^2.$$

Um den adaptiven Klassifizierer in ein adaptives Regressionsverfahren zu überführen, muss die Update-Funktion angepasst werden. Dabei kann man beide Hauptbestandteile der Update-Funktion separat betrachten. Der einfachere Teil ist die statistische Erkennung von Concept Shift. Hierbei wird versucht zu testen, ob sich die Klassifikationsrate über ein Zeitfenster signifikant verschlechtert. Für die Regression lässt sich dies einfach anpassen, indem man nun auf ein signifikantes Ansteigen des Fehlerwertes testet. Hierfür bietet sich ebenfalls ein einseitiger t-Test an.

Für die Umstellung des lokalen Updates muss neu definiert werden, unter welchen Bedingungen in der Testumgebung die aktuellen Instanzen einem anderen Modell folgen als die älteren Daten. Da angenommen wird, dass innerhalb der gesamten Testumgebung der Ausgabewert ähnlich verteilt ist, reicht es zu testen, ob sich der mittlere Ausgabewert der neuen Daten von dem der alten Daten unterscheidet. Dies lässt sich mit einem zweiseitigen t-Test gut entscheiden. Das Signifikanzniveau sollte hierbei allerdings größer sein als bei der Erkennung von Concept Shift, da das lokale Update auch für kleinere Änderungen des wahren Konzepts (gradueller Concept Drift) zuständig ist. Außerdem sind die Folgen einer falschen Entscheidung weniger kritisch, da nur die älteren Daten der Umgebung gelöscht werden. Ein sinnvolles Signifikanzniveau wäre hier z.B. $\alpha = 0.95$.

Mit diesen wesentlichen Modifikationen lässt sich eine leicht verändernde Updatefunktion für die adaptive Regression implementieren. Sicherlich sind auch hier einige weitere Optimierungen im Detail möglich, um die Performanz des Verfahrens zu optimieren. Dieses Beispiel zeigt aber, dass das Verfahren durch einfache Umstellungen auch auf andere Lernprobleme anpassbar ist.

5.5.2 Online Feature Selection

Instanzbasierte Verfahren haben generell Probleme mit hochdimensionalen Daten. Grund hierfür ist der sogenannte *Fluch der Dimensionalität* (*Curse of Dimensionality*) [Bel61], der zu Ineffizienz und Ineffektivität führt. Durch die hohe Anzahl der Dimensionen nähern sich die Distanzen des entferntesten und des nächsten Nachbarn eines beliebigen Punkts an. Außerdem erhöht sich der Speicher- und Laufzeitaufwand. Aus diesem Grund wird in solchen Fällen vor dem instanzbasiertem Lernen eine *Feature Selection* [GE03] ausgeführt, um die Dimensionalität der Daten deutlich zu verringern. In den meisten Verfahren wird hierbei auf Grundlage eines Relevanzmaßes eine Teilmenge der relevantesten Attribute ermittelt. Da es nochmal exponentiell viele mögliche Teilmengen gibt, kommen hierbei meist Greedy- und Hill-Climbing-Verfahren zum Einsatz.

Beim adaptiven Lernen erhöht sich bei hochdimensionalen Daten zusätzlich der Aufwand für ein Update, wodurch auch hierbei eine Aufwandsminimierung besonders wünschenswert ist. Eine Feature-Selection ist hierfür sinnvoll. Allerdings besteht das Problem, dass diese nicht wie im statischen Fall im Voraus möglich ist. Die Datenmenge, die für

die Auswahl der Attribute entscheidend ist, ändert sich ständig, wodurch sich auch die Teilmenge der optimalen Attribute ändern kann. Das Feature-Selection-Verfahren muss also ebenfalls online ausgeführt werden und Concept Drift berücksichtigen. Die Datenmenge, die der Feature-Selection als Grundlage dient, muss ebenso bei Concept Drift adaptiv angepasst werden. Die einfachste und effizienteste Lösung besteht darin, als Datenmenge die aktuelle Fallbasis des adaptiven Lerners zu benutzen. Das Lernen und die Feature-Selection werden somit integriert. Wenn der Lerner Daten in die Fallbasis einfügt oder löscht, muss das Feature-Selection-Verfahren ein Update machen. Falls die Feature-Selection die ausgewählte Attributsmenge ändert, muss der Lerner angepasst werden. Bei dem hier vorgestellten adaptiven Lerner bedeutet dies vor allem, dass sich die Distanzen komplett ändern können und damit der M-Tree neu aufgebaut werden muss. Da dies mit erheblichen Aufwand verbunden ist, sollte das Feature-Selection-Verfahren möglichst robust in Hinsicht auf die Häufigkeit von Veränderungen der ausgewählten Attributsmenge sein.

Die Entwicklung einer adaptiven Feature-Selection ist eine noch zu lösende Aufgabenstellung. Die wichtigsten Anforderungen im Vergleich zu statischen Verfahren sind die Folgenden: Erstens muss es sich um ein inkrementelles Verfahren handeln und zweitens muss es eine hohe Robustheit besitzen. Ein derartiges Verfahren wäre mit zahlreichen adaptiven Data-Mining-Verfahren sinnvoll kombinierbar.

5.6 Resümee

Im Zentrum dieses Kapitels stand die Präsentation eines flexiblen, instanzbasierten, adaptiven Lerners. Ein wichtiges Ziel bei der Entwicklung des mit IBL-DS bezeichneten Verfahrens war, die unterschiedlichen Formen von Concept Drift möglichst früh zu erkennen und angemessen darauf zu reagieren. Dabei besteht ein Update aus drei Abschnitten, der Erkennung von Concept Drift durch ein lokales heuristisches Verfahren, die Erkennung von Concept Shift durch einen statistischen Test und die Erkennung von redundanten Daten. In allen Fällen werden jeweils diejenigen Daten aus der Fallbasis entfernt, die für überflüssig oder potentiell schädlich erachtet werden.

Bei der Evaluation wurde darauf geachtet, Datenströme mit möglichst unterschiedlichen Arten von Concept Drift zu verwenden. Empirisch wurde gezeigt, dass IBL-DS in den Default-Einstellungen in allen Fällen mit den parameteroptimierten Vergleichsverfahren konkurrieren kann, das Verfahren also ohne Vorwissen und Optimierungen auf beliebige Datenströme effizient und effektiv anwendbar ist. Das Experiment zur Robustheit dieser Parameter zeigt, dass sie in einem sinnvollen Intervall robust zu guten Ergebnissen führen und damit nicht kritisch für die Qualität der Ergebnisse sind. Besonders auffallend hierbei sind auch die relativ guten und robusten Ergebnisse von LWF trotz einer pauschalen Adaptionsstrategie.

Zuletzt wurde skizziert, wie man das Verfahren durch wenige Änderungen auch für Online-Regressionsprobleme verwenden kann. Außerdem wurde diskutiert, wie man das instanzbasierte Verfahren durch eine Online-Feature-Selection erweitern bzw. kombinieren kann. Beide Variationsmöglichkeiten sprechen für die Flexibilität und Erweiterbarkeit des Verfahrens. Im nächsten Kapitel wird der instanzbasierte Ansatz mit modellbasierten Lernverfahren kombiniert und am Beispiel eines Regellerners umgesetzt. Hierbei werden die Vorteile des instanzbasierten Ansatzes beim adaptiven Lernen mit den Vorteilen der modellbasierten Verfahren bei der Klassifikation verbunden.

Kapitel 6

Regelbasierte Klassifikation auf Datenströmen

Nachdem das rein instanzbasierte adaptive Lernen das Thema des letzten Kapitels war, konzentriert sich dieses Kapitel nun auf modellbasierte Ansätze. Da der instanzbasierte Ansatz wichtige Vorteile beim adaptiven Lernen hat, sollen diese soweit möglich mit modellbasierten Verfahren kombiniert werden, um deren Vorteile bei der Klassifikation zu integrieren. Zunächst wird die grundsätzliche Idee, wie instanzbasiertes und modellbasiertes Lernen in einer adaptiven Umgebung zusammengebracht werden können, vorgestellt. Anschließend wird diese Idee am Beispiel eines Regellerners umgesetzt und in der Evalution mit dem rein instanzbasierten Verfahren des letzten Kapitels verglichen. Den Abschluss bildet ein kurzes Resümee zum Kapitel.

6.1 Modellbasiertes adaptives Lernen

6.1.1 Modellbasiertes Lernen

Modellbasierte Lernverfahren wie Entscheidungsbäume oder Regellerner generieren in der Lernphase ein Modell aus den vorhandenen Trainingsdaten. Dieses Modell soll das Wissen aus den Trainingsdaten kompakt darstellen, so dass neue Daten anhand des Modells effizient klassifiziert werden können. Nach der Lernphase werden die Trainingsdaten nicht mehr benötigt, was normalerweise einen deutlichen Speichergewinn zu den instanzbasierten Verfahren bedeutet. In der Lernphase können wichtige, unwichtige und fehlerhafte Daten erkannt und im Modell dauerhaft berücksichtigt werden, wodurch modellbasierte Verfahren meist auch besser generalisieren. Ein interpretierbares Modell bietet dem Benutzer zusätzlich die Möglichkeit, die Entscheidungsstrategie zur Klassifikation und damit das wahre Konzept verstehen zu können. Schließlich sind modellbasierte Verfahren effizienter in der Auswertungsphase, also bei der Klassifikation neuer Beispiele. Der größte Nachteil liegt darin, dass die verwendeten Modelle nur bestimmte wahre Konzepte repräsentieren können, weil der zugrunde liegende Hypothesenraum im Allgemeinen deutlich weniger flexible Entscheidungsgrenzen zulässt als instanzbasierte Verfahren.

6.1.2 Adaptives Lernen mit modellbasierten Verfahren

Die Tatsache, dass Lern- und Klassifikationsphase gleichzeitig stattfinden, führt bei modellbasierten Verfahren zu besonderen Problemen. Der Vorteil, dass ein einmal gelerntes Modell die Speicherung der zugrunde liegenden Trainingsdaten unnötig macht und auf Dauer eine einfache und effiziente Klassifikation neuer Daten ermöglicht, ist nicht mehr gegeben. Das Lernen von neuem Wissen und das Erkennen und Vergessen von veraltetem Wissen erfordert eine hohe Flexibilität, die bei modellbasierten Ansätzen meist nicht gegeben ist. Entscheidungsbäume wie bei BOAT, CVFDT und UFFT müssen dafür z.B.

komplette Teilbäume entfernen und gleichzeitig neue Teilbäume lernen.

Ein Update des Modells bei vorhandenem Concept Drift ist ohne Speicherung der zugrunde liegenden Daten nur ungenau möglich. Es ist durchaus möglich, festzustellen, dass Teile des gelernten Modells nicht mehr gültig sind. Da in einem Modell Bereiche des Datenraums zusammengefasst werden, ist ohne Kenntnis der in einem betroffenen Bereich vorhandenen Daten aber nicht feststellbar, welcher Teil sich genau geändert hat. Man hat nur die Möglichkeit, den kompletten Bereich neu zu lernen. Neben dem aktuellen Modell ist es deshalb sinnvoll eine ausgewählte aktuelle Datenmenge zu behalten, die als Grundlage für das Modell dient. Alle bisherigen modellbasierten Ansätze wie CVFDT, UFFT und FLORA (vergleiche Abschnitt 3.4) speichern direkt (als explizite Datenmenge) oder indirekt (z.B. ausführliche Zusammenfassung in den Baumknoten) die Daten, auf denen das aktuelle Modell basiert.

Bei einem Update ist es damit notwendig, das Modell und die dem Modell zugrundeliegende Datenmenge zu aktualisieren. Zur Aktualisierung der Datenmenge wurden dazu bisher nur Standard-Strategien wie das Sliding Window benutzt (z.B. CVFDT, UFFT), das die Datenmenge unabhängig von vorhandenem Concept Drift anpasst, oder einfache Erweiterungen, die die Größe des Sliding Windows optimieren.

Die neue Idee ist es nun, auch das Update der zugrundeliegenden Datenmenge anhand des vorhandenen Modells und vorhandenem Concept Drift zu aktualisieren. Die Größe der Datenmenge kann sich damit der Komplexität des Modells anpassen. Bei einfachen Modellen kann auch die zugehörige Datenmenge klein gehalten werden, wohingegen bei komplexeren Modellen mehr Daten benötigt werden.

Ein großer Vorteil bei der Verwendung von gut generalisierenden Verfahren ist, dass die Datenmenge auch einige fehlerhafte Daten beinhalten kann, ohne dass die Qualität darunter entscheidend leidet. Unter dieser Voraussetzung muss man nicht mehr sofort entscheiden, ob Daten fehlerhaft sind, und es besteht die Möglichkeit, mit dem Löschen von Daten zu warten, bis die Entscheidung besser abgesichert ist. Effiziente und erfolgreiche instanzbasierte Strategien, wie die statistische Concept-Shift Erkennung, können damit auch für beliebige modellbasierte Verfahren angewendet werden.

Das zugrunde liegende Modellschema des verwendeten Verfahrens sollte ein möglichst flexibles und einfaches Update ermöglichen. Hierbei sind insbesondere solche Modelle von Vorteil, deren Bestandteile unabhängig voneinander sind, so dass Teile des Modells unabhängig vom Rest verändert werden können. In den folgenden Abschnitten wird ein konkretes Verfahren auf Basis eines Regellerners vorgestellt.

6.2 Regelbasiertes Lernen

Dieser Abschnitt gibt einen kurzen Einblick in Regellernverfahren für statische Daten. Bevor die unterschiedlichen Verfahren angesprochen und erläutert werden, wird zunächst eine Beschreibung von üblicherweise verwendeten Regeln gegeben. Am Ende des Abschnitts folgt eine Diskussion darüber, welche Regelverfahren für das adaptive Lernen geeignet sind.

6.2.1 Regeln

Der Zweck einer Regel ist es, einem bestimmten Teilbereich des Datenraums eine Klasse zuzuordnen. Dabei gibt hauptsächlich zwei verschiedene Arten von Regeln, die in den bekanntesten Verfahren benutzt werden. Dies sind einmal die *separierenden Regeln* und die *Hyperbox-Regeln*. Eine Regel R ist aus einer zu erfüllenden Bedingung B und einer Klassenzuweisung C in der Konklusion aufgebaut:

$$R: \text{ IF } B \text{ THEN } C.$$

Eine Regelbedingung B besteht dabei aus einer Konjunktion einzelner Attributbedingungen (Selektoren genannt) B_i ($i \in \{1, \ldots, k\}$):

$$B = B_1 \wedge \ldots \wedge B_k.$$

Mehrere Regel können dabei auch zu einer *IF-THEN-ELSE*-Hierarchie aufgebaut sein. Bei der Form der Selektoren muss man zwischen numerischen und diskreten Attributen unterscheiden.

Bei separierenden Regeln sollen die einzelnen Bedingungen den Raum und damit die Datenmenge aufteilen. Sie wurden zunächst für Daten mit rein diskreten Attributen entwickelt. Ein Selektor B_i für ein diskretes Attribut a_i mit den möglichen Ausprägungen $\{v_1, \ldots, v_n\}$ besitzt dabei eine Form

$$B_i = (a_i = v_j) \text{ mit } j \in \{1, \ldots, n\}.$$

Später wurden auch numerische Attribute zugelassen, wobei Selektoren der Form

$$B_i = (a_i < c) \text{ mit } c \in \mathbb{R}$$

verwendet wurden. Statt '$<$' sind auch die Vergleichsoperatoren '$\leq$', '$>$' und '$\geq$' möglich. Es sind auch mehrere Bedingungen für ein numerisches Attribut möglich. Typische Verfahren, die separierende Regeln verwenden, sind die im nächsten Unterabschnitt beschriebenen Separate-and-Conquer-Strategien und Verfahren auf Basis von Entscheidungsbäumen.

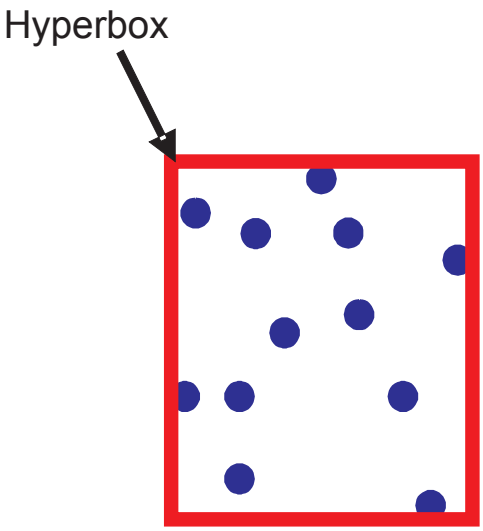

Abbildung 6.1: Beispiel einer Hyperbox

Hyperbox-Regeln sollen dagegen eine bestimmte Teilmenge der Daten minimal umfassen. Im Gegensatz zu den separierenden Regeln wurden Hyperbox-Regeln zunächst für Datensätze mit rein numerischen Attributen verwendet. Die Form einer Bedingung B_i für ein Attribut A_i besitzt dabei die Form

$$B_i = (a_i \in [l_i, h_i]) \text{ mit } l_i \leq h_i.$$

Jede Bedingung grenzt also den Wert eines Attributs auf ein Intervall ein, wobei es für jedes Attribut auch genau eine Bedingung gibt. Hyperbox-Regeln werden meist als minimal umfassendes Hyper-Rechteck einer Menge von Daten erzeugt. Als Beispiel sei eine Datenmenge $X = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathbb{R}^2$ gegeben. Die Bedingung der Regel lautet dann wie folgt (siehe auch Abbildung 6.1):

$$x \in [\min_i x_i, \max_i x_i] \wedge y \in [\min_i y_i, \max_i y_i].$$

Die durch eine Regel zusammengefassten Beispiele werden generalisiert, indem der gesamte, von der Regel überdeckte Teilraum der (häufigsten) Klasse der Beispiele zugeordnet wird. Für die Erweiterung auf diskrete Attribute werden Bedingungen der Form $B_i = (a_i = v)$ benutzt, um den Attributswert auf eine Ausprägung zu beschränken. Mit der Bedingung $B_i = (a_i = TRUE)$ wird der gesamte Wertebereich des Attributs zugelassen. Eine Beschränkung des Attributswertes auf eine echte Teilmenge mit mehreren Attributswerten wird nur selten, wie in FACIL [FTARR06], berücksichtigt. Hyperbox-Regeln werden unter anderem in den im nächsten Abschnitt erwähnten EACH- und RISE-Verfahren verwendet.

6.2.2 Regellerner

In der Klassifikation auf statischen Daten gibt es verschiedene Herangehensweisen zum Lernen von Regeln. Die verbreitetste Variante sind die *Separate-and-Conquer*-Strategien, wie IREP [FW94], Ripper [Coh95] oder CN2 [CN89]. Hierbei werden Regeln gelernt, die eine Menge von bisher nicht überdeckten Instanzen von den restlichen Instanzen separieren. Anschließend werden nur aus den restlichen Instanzen weitere Regeln gelernt. Dies geschieht solange, bis alle Instanzen von einer Regel überdeckt werden. Das Ergebnis ist eine Liste von Regeln mit fester IF-THEN-ELSE-Hierarchie. Dadurch sind die Regeln direkt voneinander abhängig und eine Änderung einer Regel hat direkten Einfluss auf die darunter angeordneten Regeln.

Es gibt allerdings auch einige Erweiterungen, die versuchen, die hierarchische Anordnung der Regeln zu vermeiden. Vorteile dieser Ansätze sind die meist sehr geringe Anzahl von Regeln und die effiziente Berechnung. Allerdings ist dieser Ansatz auf kategorielle Daten beschränkt. Numerische Daten müssen zuvor diskretisiert werden, womit immer ein Informationsverlust verbunden ist. Allerdings bietet dieser Ansatz kaum Möglichkeiten für einen inkrementelle Anwendung. Ein begrenztes Update ist über Ausnahmeregeln möglich, allerdings würde man damit in kürzester Zeit ein sehr komplexes System aus Regeln und Ausnahmen bekommen, da jeweils nur neue Regeln und Ausnahmen hinzu kommen. Als Alternative müssten in jedem Schritt die Regeln auf Basis der Fallbasis komplett neu gelernt werden, was natürlich sehr aufwändig wäre. Damit ist eine Verwendung dieses Ansatzes für das adaptive Lernen weniger geeignet.

Ein weiterer oft benutzter Ansatz ist das Lernen von Regeln auf der Basis eines Entscheidungsbaumes. Dabei wird zunächst ein Entscheidungsbaum gelernt und dieser dann in eine Menge von gleichwertigen Regeln umgewandelt. Meist findet anschließend noch ein Pruning der Regelmenge statt. Einer der in der Literatur am häufigsten verwendeten Vertreter ist „C4.5Rule“ [Qui93]. Der Vorteil hierbei ist eine vergleichbar gute Performanz und Generalisierung wie bei Klassifikationsbäumen, dafür aber auch eine bessere Darstellung der Entscheidungsstrategie. Allerdings ist die Bestimmung der Regeln über den Umweg eines Klassifikationsbaums umständlich.

Ein Ansatz, der seltener auftritt, sind Hyperbox-Regellerner. Eine Regel sind hierbei zu einer Hyperbox generalisierte Instanzen, wobei jede Regel durch eine zugehörige Instanzmenge definiert ist. Deshalb kann man hierbei auch von instanzbasiertem Regellernen sprechen. Zunächst wird eine einzelne Instanz als Regel interpretiert, die nur diese Instanz überdeckt. Durch Hinzunahme von weiteren Instanzen wird die Regel sukzessive erweitert. Wie in vielen anderen Varianten benötigt man eine Entscheidungsstrategie zur Auflösung von Konflikten, bei denen eine neue Instanz von mehreren Regeln überdeckt wird. Im Gegensatz zu den anderen Ansätzen, in denen normalerweise der gesamte Datenraum durch die Regeln abgedeckt wird, gibt es noch den Sonderfall, dass eine neue Instanz von keiner Regel überdeckt wird. Eine hierbei häufig gewählte Strategie ist, die

nächstgelegene Regel zu verwenden. Dadurch und aufgrund der Tatsache, dass Regeln nichts anderes als generalisierte Instanzen sind, ist eine Verwandtschaft zum instanzbasierten Lernen gegeben. Dieser Ansatz wurde in EACH [Sal91] verwendet, hatte aber erst mit dem RISE-System von Domingos [Dom95, Dom96, Dom97] seinen Durchbruch. RISE produziert sehr gute Klassifikationsraten, die mit denen von C4.5 vergleichbar sind, und ermöglicht die Verwendung von numerischen und diskreten Attributen durch eine geeignete Abstandsfunktion, die im letzten Kapitel (siehe Abschnitt 5.3.3) schon vorgestellt wurde. Bei der Klassifikation einer neuen Instanz sucht man die Regel, die der Instanz am nächsten ist. Der Kern des Verfahrens ist allerdings die Strategie zur Induktion der Regeln, in der zunächst jede Trainingsinstanz als Regel interpretiert wird. Durch Hinzunahme von angrenzenden, noch nicht überlagerten Instanzen gleicher Klasse werden die Regeln sukzessive vergrößert. Als Gütefunktion zur Entscheidung, welche neue Regel akzeptiert wird, dient in RISE keines der sonst üblichen Maße, sondern die aktuelle Klassifikationsrate aller Trainingsdaten.

6.2.3 Adaptives Regellernen

Wie in Abschnitt 6.1.2 schon erwähnt, sollte das Modell eines adaptiven Lerners möglichst flexibel und einfach anzupassen sein. Bei Regellernern ist es deshalb vorteilhaft, die Regeln möglichst direkt aus den Daten zu erzeugen und nicht über komplexe Umwege, wie etwa im Fall von Entscheidungsbäumen. Außerdem sollten keine Abhängigkeiten zwischen den Regeln bestehen, damit einzelne Regeln unabhängig von anderen Regeln geändert werden können. Deshalb kommen auch keine Verfahren mit hierarchischen IF-THEN-ELSE-Regeln für das adaptive Lernen in Frage. Die beste Möglichkeit, Regeln effizient auf eine durch Löschen und Hinzufügen geänderte Datenmenge anzupassen, bieten dabei Hyperbox-Regeln, da sie direkt aus einer zugrunde liegenden Datenmenge erzeugt werden. Ändert sich diese Menge, kann die Regel effizient neu bestimmt werden. Die Strategie, Regeln durch Hinzunahme neuer Instanzen zu erweitern, ermöglicht es, die Regeln in der adaptiven Umgebung jederzeit weiter zu lernen, ohne komplett neu beginnen zu müssen. Allerdings erfordert eine adaptive Umgebung, dass Regeln nicht nur erweitert sondern auch eingeschränkt (spezialisiert) werden können. Dies ist aber durch den direkten Zusammenhang der Regel mit einer Datenmenge einfach möglich, indem Daten am Rand der Hyperbox aus der Menge gelöscht werden und die Regel neu erzeugt wird. Aus diesen Gründen basiert der im Folgenden vorgestellte adaptive Regellerner auf dem Hyperbox-Lernverfahren des RISE-Systems. Dieses Lernverfahren wurde für die adaptive Umgebung erweitert und angepasst.

6.3 Der Algorithmus RL-DS

Wie schon in der Einleitung des Kapitels erwähnt, basiert das selbst entwickelte adaptive Regelverfahren für Datenströme (RL-DS) auf zwei Mengen, der Fallbasis und der Regelmenge. Die Regeln basieren dabei auf Instanzen der Fallbasis und werden mit deren Hilfe permanent aktualisiert. Regeln präsentieren Hypothesen und weisen einem größeren Bereich des Datenraums eine Klasse zu. Die Fallbasis hat dagegen das Potenzial, aktuelles, noch ungesichertes Wissen von Instanzen zunächst zu speichern. Falls sich das Wissen bestätigt, wird es durch Regeln gefestigt, ansonsten können die Instanzen wieder entfernt werden. Für Bereiche, die durch Regeln und Instanzen die Klassifikation klar bestimmt sind, sind nur wenige Daten in der Fallbasis notwendig. Dadurch teilt sich ein Update in zwei Bereiche auf: Update der Fallbasis und Update der Regeln. In beiden Fällen wird besonderen Wert darauf gelegt, ein Update nur lokal auszuführen, um auch in größeren Datenräumen mit höheren Dimensionen effizient zu bleiben.

6.3.1 Update der Fallbasis

Wenn neue Trainingsdaten erscheinen, sollen sie natürlich in die aktuelle Fallbasis aufgenommen werden. Allerdings müssen auch regelmäßig wieder Daten gelöscht werden, damit sie flexibel und aktuell bleibt. Für die Erkennung von Concept Shift wird das gleiche Verfahren wie in IBL-DS verwendet (siehe Abschnitt 5.3.2). Die Entscheidung darüber, welche Instanzen bei Concept Drift gelöscht werden können, geschieht in Abstimmung mit der aktuellen Regelmenge. Dabei werden zwei Fälle unterschieden: Instanzen, die von Regeln gleicher Klasse überdeckt werden und Instanzen ohne passende Regeln. Um das Update lokal auszuführen, werden nur Regeln beachtet, die eine neue Trainingsinstanz überdecken. Instanzen ohne Regeln werden dagegen global betrachtet, da es effizienter ist, diese extra zu global zu indizieren, als sie in der lokalen Umgebung neuer Trainingsinstanzen ausfindig zu machen. Um die Laufzeit und den Speicheraufwand des Verfahrens regeln zu können, ist eine Obergrenze für die Größe der Fallbasis möglich. Wird diese überschritten, müssen auch Instanzen gelöscht werden. Die genauen Vorgehensweisen werden im Folgenden beschrieben:

Instanzen mit Unterstützung Beispiele, also Instanzen mit bekannter Klasse, gelten als unterstützt, wenn sie von einer Regel überdeckt werden, deren Markierung der eigenen Klasse entspricht. Diese Instanzen werden nur dann gelöscht, wenn sie redundant sind. Eine Instanz gilt als redundant, wenn in einer kleinen Nachbarschaft der Größe k_{red} nur Instanzen der gleichen Klasse vorhanden sind. Allerdings darf die Instanz nicht auf dem Rand der Regel liegen, also nicht für die Grenzfestlegung der Regel verantwortlich sein. Solange die Mindestanzahl k_{rule} von Instanzen für eine Regel eingehalten wird, werden

Symbol	Bedeutung
m	Anzahl der Klassen
l	Anzahl der Attribute
X	Menge neuer Trainingsbeispiele
I	aktuelle Fallbasis
R	aktuelle Regelmenge
c_i	Klasse einer Instanz i
c_r	Klasse einer Regel r
max_I	maximale Größe der Fallbasis
max_R	maximale Größe der Regelmenge
k_{rule}	Maximale Anzahl von Daten, die für die Unterstützung einer Regel notwendig sind.
k_{red}	Nachbarschaft einer Instanz die einheitlich sein muss, damit sie als Redundanz gelöscht werden darf.
k_{noise}	Mindestanzahl an Instanzen einer Regel, damit auf Rauschen (Noise) überprüft wird.
k_{step}	Notwendige Anzahl neuer Trainingsdaten für ein Update der Regeln
$p_{current}$	Anteil Instanzen einer Regel, die zu den aktuellen gehören.
p_{young}	Anteil jüngster Trainingsdaten, die auch ohne Regel gespeichert werden.
h	Anzahl Instanzen, die zur Berechnung der Fehlerrate benutzt werden
h_{last}	Anzahl Instanzen zur Berechnung der aktuellen Fehlerrate bei Concept Shift
$instancesOf(r)$	Menge aller Instanzen aus I, die von der Regel r überdeckt werden
$rulesOf(i)$	Menge aller Regeln aus R, die die Instanz i überdecken
$quality(r)$	Güte einer Regel r

Tabelle 6.1: RL-DS: verwendete Notationen

die ältesten redundanten Instanzen gelöscht.

Desweiteren bieten Regeln die Möglichkeit, Rauschen oder veraltete Daten zu entdecken. Dazu dient die einfache Strategie: Wenn die $p_{current}\%$ jüngsten Instanzen einer Regel der Klasse der Regel entsprechen, werden ältere Instanzen anderer Klassen als Rauschen oder veraltet gelöscht. Allerdings muss die Regel für eine sinnvolle und zuverlässige Entscheidung eine Mindestanzahl k_{noise} von Instanzen überdecken. Der Pseudocode für diesen Teil des Updates ist in Algorithmus 6.1 zu finden.

Instanzen ohne Unterstützung Instanzen zu speichern, die von keiner passenden Regel überdeckt werden, macht nur dann Sinn, wenn sie noch relativ aktuell sind und somit das Potenzial haben, durch weitere neue Instanzen neue Regeln zu generieren und damit neues Wissen zu repräsentieren. Die Entscheidung, derartige Daten zu löschen, ist besonders kritisch. Je länger derartige Instanzen gespeichert werden, desto wahrscheinlicher können sich auch relativ kleine Regeln bilden. Andererseits bleibt dann auch Rauschen länger erhalten, was die Wahrscheinlichkeit erhöht, dass fehlerhafte Regeln erzeugt werden. In RL-DS werden alle Instanzen gelöscht, die nicht zu den $p_{young}\%$ aktuellsten Daten der Fallbasis gehören. Die Default-Einstellung ist $p_{young} = 25\%$.

Statistische Concept Shift Erkennung Zur Erkennung von Concept Shift wird die Klassifikationsrate der letzten h Trainingsdaten beobachtet. Falls ein statistischer t-Test eine signifikante Verschlechterung feststellt oder die Rate gar schlechter als zufälliges Raten wird, werden je nach Ausmaß der Veränderung Daten aus der Fallbasis gelöscht. Da dieser Teil identisch zur Statistischen Concept-Shift-Erkennung des instanzbasierten Verfahrens ist, sei hier für eine genauere Beschreibung auf Abschnitt 5.3.2 verwiesen. Eine Änderung gibt es allerdings doch: Falls der Fehler größer als beim trivialen Klassifikator wird, also $h_{last} > 1 - \frac{1}{m}$ gilt, wird nicht nur die komplette Fallbasis gelöscht, sondern auch die komplette Regelmenge, da ein Neulernen der Regel auf jeden Fall erfolgreicher ist. Die Default-Einstellungen sind wie bei IBL-DS $h = 100$ und $h_{last} = 20$.

Überlauf der Fallbasis Grundsätzlich sollte jede Regel nur eine relativ geringe Anzahl von Instanzen benötigen, die sie überdecken. Durch Überschneidungen mit Regeln anderer Klassen und durch Rauschen wird dies allerdings deutlich erhöht. Eine große Fallbasis führt allerdings auch zu vielen Regeln, was die Effizienz und die Verallgemeinerung des gelernten Modells sehr beeinträchtigt. Deshalb macht es Sinn, die Größe der Fallbasis zu beschränken. Falls mehr Daten vorhanden sind, werden die ältesten gelöscht, bis die Obergrenze wieder erreicht wird. Die Obergrenze max_I, die auch jederzeit verändert werden kann, bietet eine Möglichkeit, den notwendigen Speicherplatz und die Laufzeit zu verändern. In den Experimenten werden maximal $max_I = 1000$ Trainingsinstanzen erlaubt.

Algorithmus 6.1 : RL-DS Update: Update der Fallbasis

Input : neue Instanzen X
`statistischeShiftErkennung(`X`);`
$C \leftarrow \{\}$;
foreach *Instanz* $x \in X$ **do**
  $I \leftarrow I \cup \{x\}$ `// *`
  neue Instanzen hinzufügen $C \leftarrow C \cup$`rulesOf(`x`)` `// *`
  neue Regeln erzeugen
end
`// Redundante und fehlerhafte Instanzen aus Regeln löschen`
foreach *Regel* $r \in C$ **do**
  if |`instancesOf(`r`)`| > k_{noise}
  $\wedge$ $p_{current}$% *jüngste aus* `instancesOf(`r`)` *gehören zur gleichen Klasse* c_r
  then
    $R \leftarrow R \setminus \{i \in$`instancesOf(`$r$`)`$|c_i \neq c_r\}$;
  end
  $i \leftarrow$ älteste Instanz in `instancesOf(`r`)`;
  while |`instancesOf(`r`)`| > k_{rule}
  $\wedge$ *noch nicht alle Instanzen aus* `instancesOf(`r`)` *überprüft*
  do
    if $\forall p \in$`rulesOf(`i`)`$.c_p = c_i$
    $\wedge$ k_{red} *nächste Nachbar von* i *sind gleicher Klasse*
    $\wedge$ i *liegt nicht auf dem Rand von* r **then**
      $I \leftarrow I \setminus \{i\}$;
    end
    $i \leftarrow$ nächst jüngere Instanz aus `instancesOf(`r`)`;
  end
end
`// Ältere Instanzen ohne Regel löschen`
$I' \leftarrow I \setminus \{p_{young}\%$ jüngste Instanzen von $I\}$;
$I \leftarrow I \setminus \{i \in I' | \forall r \in$`rulesOf(`$i$`)`$: c_r \neq c_i\}$;
`// Überlauf der Fallbasis kontrollieren`
while $|I| > s$ **do**
  $I \leftarrow I \setminus \{$älteste Instanz $x \in I\}$;
end

6.3.2 Update der Regeln

Die Regelmenge wird in vier Schritten aktualisiert. Zunächst werden die neu eingefügten Trainingsinstanzen verarbeitet. Dabei können sowohl unpassende Regeln verkleinert (spezialisiert) als auch neue Regeln erzeugt werden. Desweiteren werden Regeln, die neue Instanzen überdecken, durch generalisieren und spezialisieren weiter gelernt, außerdem werden diese Regeln bei Bedarf zu allgemeineren Regeln vereinigt. Letztendlich werden Regeln wenn nötig auch gelöscht. Alle Phasen beim Update von Regeln werden aus Effizienzgründen lokal ausgeführt. Dabei werden nur Regeln betrachtet, die eine neue Trainingsinstanz überdecken. Die einzige Ausnahme ist das Löschen von zu kleinen Regeln, was alle Regeln betreffen kann.

Diese vier Schritte werden im Folgenden detailliert beschrieben, wobei auf die Definition und Beschreibung des Gütemaßes und der Distanzen mit Regeln in den technischen Details (Abschnitt 6.4) eingegangen wird. Der komplette Pseudocode für das Update der Regeln befindet sich in Algorithmus 6.7. Die Hauptfunktion für das Update des kompletten Lerners befindet sich in Algorithmus 6.8.

Trainingsinstanz einfügen Zu einer neu hinzugefügten Trainingsinstanz werden alle Regeln ermittelt, die sie überdecken. Falls dabei keine Regel der gleichen Klasse zu finden ist, wird eine neue Regel, die zunächst nur aus dieser Instanz besteht, erzeugt. Regeln anderer Klassen, die von der Instanz überdeckt werden, finden ebenso Beachtung. Es wird zunächst durch die Funktion $reduce()$ (Beschreibung in Abschnitt 6.4.1) eine neue Regel erzeugt, die die Trainingsinstanz nicht mehr beinhaltet. Falls das Gütemaß der neuen Regel nicht schlechter ist, ersetzt sie die alte Regel in der Regelmenge. Diese Aufgabe wird in der Hauptfunktion des Algorithmus 6.7 durchgeführt.

Regeln generalisieren und spezialisieren In traditionellen Regellernverfahren mit Hyperbox-Regeln werden Regeln nur durch Hinzunahme von Instanzen verallgemeinert. Für eine adaptive Umgebung ist das allerdings nicht mehr ausreichend. Aufgrund von Concept Drift kann es sinnvoll werden, Regeln durch den Ausschluss von überdeckten Instanzen anderer Klassen zu spezialisieren. Der in RL-DS verwendete Algorithmus zum generalisieren von Regeln lehnt sich dabei an das Induktionsverfahren von RISE an. Hierbei wird zu einer Regel die am nächsten liegende Instanz gesucht, die nicht überdeckt wird. Die Regel wird dabei genau dann mit der Instanz verallgemeinert, wenn das Gütemaß verbessert bzw. wenigstens nicht verschlechtert wird (zur Distanzberechnung siehe Abschnitt 6.4.2). Eine Illustration ist in Abbildung 6.2 dargestellt.

Bei der Spezialisierung von Regeln wird ähnlich vorgegangen. Zunächst wird eine überdeckte Instanz einer anderen Klasse gesucht. Dabei wird diejenige gewählt, die vom Zentrum der Regel am weitesten entfernt liegt. Anschließend wird mit der Funk-

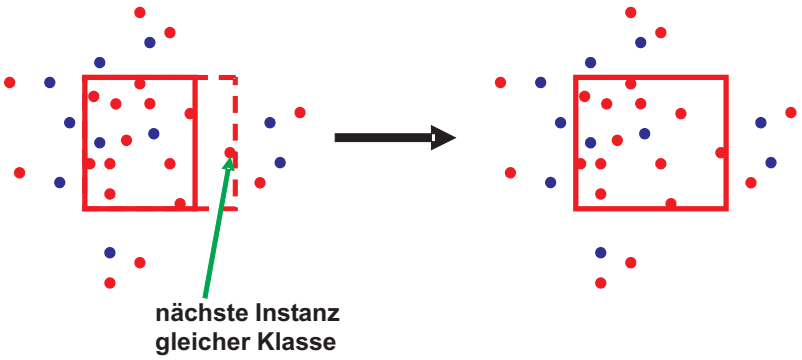

Abbildung 6.2: Regel verallgemeinern

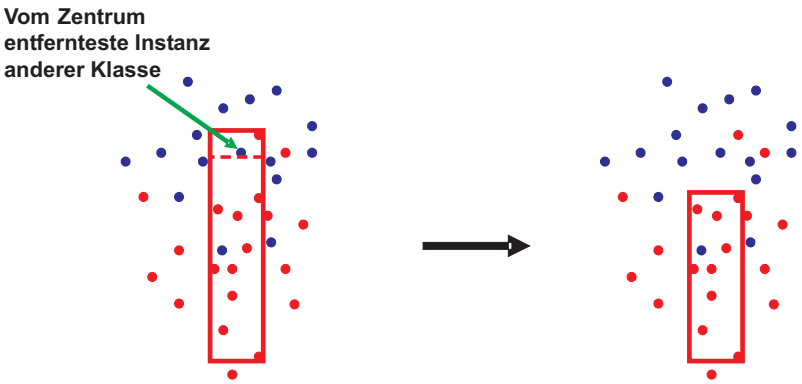

Abbildung 6.3: Regel spezialisieren

tion $reduce()$ eine neue Regel erzeugt, die die gewählte Instanz nicht mehr beinhaltet. Es wird schließlich die Regel mit dem besten Gütemaß verwendet (bei Gleichheit wird die Regel gewählt, die mehr Instanzen überdeckt). Das Beispiel in Abbildung 6.3 zeigt auch, warum es sinnvoll ist, dieses Maß zu verwenden und nicht den geringsten Abstand zum Rand. Ansonsten würden häufig extrem lange und schmale Regeln entstehen, was es zu vermeiden gilt, da sie sehr schlecht generalisieren. Eine Variante wäre noch, mehrere nächste Instanzen zu testen. Dies würde aber den Aufwand deutlich erhöhen lassen und zumindest für die Verallgemeinerung zeigt Domingos mit RISE [Dom97], dass dadurch keine signifikante Besserung zu erreichen ist. Nur in dem Sonderfall, dass mehrere Instanzen gleich nah sind, werden alle Instanzen zum Testen verwendet. Bei einem Update werden nur Regeln verallgemeinert oder spezialisiert, die von einer Trainingsinstanz überdeckt werden. Die Generalisierung und Spezialisierung einer Regel wird in der Funktion $learnRule()$ des Algorithmus 6.7 ausgeführt.

Regeln vereinigen Hierbei werden Regeln gleicher Klasse, die sich überlappen, zunächst zum Test vereinigt. Ist das Gütemaß mindestens so gut wie der durchschnittliche

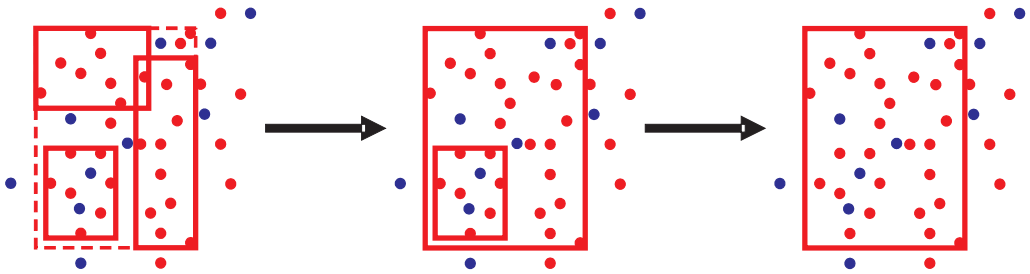

Abbildung 6.4: Modell vereinfachen durch Vereinigung von Regeln

Wert der ursprünglichen Regeln, ersetzt die Vereinigung die ursprünglichen Regeln. Diese Maßnahme ist sehr hilfreich, um die Anzahl der Regeln zu verringern, das Modell also zu vereinfachen. Neues Wissen wird hierbei kaum generiert. Ein einfaches Beispiel ist in Abbildung 6.4 dargestellt. Dabei werden aufgrund des lokalen Updates nur Regeln berücksichtigt, die ein neues Beispiel überdecken. Diese Regeln werden paarweise auf eine mögliche Vereinigung getestet. Die Funktion $unionRules()$ des Algorithmus 6.7 führt die Vereinigung von Regeln durch.

Regeln löschen Regeln können aus vier unterschiedlichen Gründen gelöscht werden, die im Folgenden beschreiben werden:

- *redundante Regeln:*
  Bei Regeln, die neu eingefügt oder verändert werden, wird untersucht, ob schon eine Regel existiert, die sie vollständig überdeckt und nach dem Gütemaß besser ist. Ebenso wird überprüft, ob durch neue und veränderte Regeln eine existierende Regel auf gleiche Weise redundant wird. Redundante Regeln werden gelöscht bzw. nicht eingefügt.

- *Regeln ohne Unterstützung:*
  Regeln, die weniger als 4 Instanzen gleicher Klasse überdecken, werden gelöscht, da sie zu wenig generalisieren.

- *schlechte Regeln:*
  Regeln bei denen die eigentliche Klasse nicht mehr die häufigste Klasse der überdeckten Instanzen ist oder deren Fehlerrate der letzten zehn überdeckten Trainingsinstanzen mindestens 0.5 ist gelten als zu schlecht und werden gelöscht.

- *zuviele Regeln:*
  Um die Laufzeit eines Updates und den Speicheraufwand regulieren zu können, kann eine Obergrenze max_R für die Anzahl der verwendeten Regeln angegeben werden. Wird diese überschritten, müssen Regeln gelöscht werden. Da schlechtere und kleine Regeln den geringsten Nutzen haben, werden die Regeln gelöscht, die

das geringste Produkt aus dem Gütemaß und der Anzahl überdeckter Instanzen besitzen ($quality(r) \cdot |instancesOf(r)|$). Als Default-Einstellung wird $max_R = 300$ verwendet.

Für das Löschen von Regeln sind zwei verschiedene Funktionen innerhalb des Algorithmus 6.7 zuständig. Die Funktion $checkRemove()$ wird lokal auf die Regeln, die ein neues Beispiel überdecken, ausgeführt und sucht nach redundanten und schlechten Regeln sowie nach Regeln ohne Unterstützung. Die Funktion $checkAllRules()$ wird dagegen auf die gesamte aktuelle Regelmenge ausgeführt, sucht nach Regeln ohne Unterstützung und löscht Regeln bei einem Überlauf der Regelmenge.

Prozedur `learnRule(r)`

Input : Rule r

repeat
  `// Regel generalisieren`
  $n \leftarrow \text{argmin}_{\text{i}\in\{\text{x}\in\text{R}\setminus\texttt{instancesOf(r)}\,|\,\text{c}_\text{x}=\text{c}_\text{r}\}}(\texttt{distance(r,i)})$;
  $p \leftarrow$ `generalize(r,n)`;
  `// Regel spezialisieren`
  if $\exists i \in$ `instancesOf(r)` $.c_i \neq c_r$ **then**
    $n' \leftarrow \text{argmin}_{\text{i}\in\{\text{x}\in\texttt{instancesOf(r)}\,|\,\text{c}_\text{x}\neq\text{c}_\text{r}\}}(\texttt{innerdistance(r,i)})$;
    $q \leftarrow$ `reduce(r,n')`;
    $b \leftarrow \text{argmax}_{\text{z}\in\{\text{r,p,q}\}}(\texttt{quality(z)})$;
  else
    $b \leftarrow \text{argmax}_{\text{z}\in\{\text{r,p}\}}(\texttt{quality(z)})$;
  end
  `// falls bessere Regel gefunden, alte Regel durch bessere ersetzen`
  if $b \neq r$ **then**
    $R \leftarrow R \setminus \{r\}$;
    $R \leftarrow R \cup \{b\}$;
    $r \leftarrow b$;
  end
until r *is unchanged*;

6.3.3 Initialisierung

Die Initialisierung ist im Gegensatz zum instanzbasierten Verfahren, bei dem die Initialisierungsdaten einfach als aktuelle Fallbasis übernommen werden, etwas umfangreicher. Um auch die Regelmenge zu initialisieren, müssen auch schon Regeln aus den Daten

Prozedur `unionRules(`U`)`

Input : Pairs of Rules U

foreach *Pair of Rules* $(r_1, r_2) \in U$ **do**
  `// vereinige Regeln`
  $r \leftarrow$`generalize(`r_1, r_2`);`
  if `quality(`r`)` $\geq \frac{1}{2}$`(quality(`r_1`)+quality(`r_2`))` **then**
    $R \leftarrow R \setminus \{r_1, r_2\}$;
    $R \leftarrow R \cup \{r\}$;
  end
end

Prozedur `checkRemove(`L`)`

Input : Rules L

`// Lösche kleine und schlechte Regeln`
foreach *Rule* $r \in L$ **do**
  $B \leftarrow$`instancesOf(`r`);`
  if $(|\{x \in B | c_x = c_r\}| < 4) \vee (|\{x \in B | c_x = c_r\}| < |\{x \in B | c_x \neq c_r\}|)$
  $\vee$`(lastError(`r`)` ≥ 0.5) **then**
    $R \leftarrow R \setminus \{r\}$;
    $L \leftarrow L \setminus \{r\}$;
  end
end
`// Lösche redundante Regeln`
foreach *Rules* $r_1, r_2 \in L, r_1 \neq r_2, c_{r_1} = c_{r_2}$ **do**
  if $(r_1 \subseteq r_2) \wedge$ `(quality(`r_1`)` $\leq$ `quality(`r_2`))` **then**
    $R \leftarrow R \setminus \{r_1\}$;
    $L \leftarrow L \setminus \{r_1\}$;
  end
end

Prozedur `checkAllRules`

```
// Lösche zu kleine Regeln
foreach Rule r ∈ R do
    B ←instancesOf(r);
    if (|{x ∈ B|c_x = c_r}| < 4) then
        R ← R \ {r};
    end
end
// Lösche Regeln bei Überlauf
while |R| > max_R do
    R ← R \ {argmin_{r∈R}(quality(r) · |instancesOf(r)|)}
end
```

gelernt werden. Dies wird auf sehr einfache Weise durchgeführt. Zunächst werden alle Daten auch als Regel initialisiert und einmal komplett gelernt. Im Lernschritt wird die zunächst hohe Zahl von Regeln wieder deutlich verringert. Die Initialisierung ist in Algorithmus 6.9 als Pseudocode dargestellt.

6.3.4 Klassifikation

Ein weiterer Bestandteil eines jeden Klassifikationsalgorithmus ist die eigentliche Klassifikation neuer Daten mit Hilfe des Modells. Bei diesem Regelverfahren wird zur Klassifikation einer neuen Instanz zunächst überprüft, ob die Instanz eine Regel der aktuellen Regelmenge erfüllt. Für den Fall, dass keine Regel die Instanz überdeckt, wird die Regel gesucht, die den geringsten Abstand besitzt. Falls mehrere Regeln in Frage kommen, wird für die Vorhersage der Klasse die Regel mit dem besten Gütemaß gewählt.

6.4 Technische Details

6.4.1 Funktionen

Dieser Unterabschnitt gibt eine kurze Definition der die im Algorithmus 6.1 und 6.7 verwendeten Funktionen:

- $rulesOf(i)$: Liefert zu einer Instanz i alle Regeln, von denen sie überdeckt wird. Zur Optimierung wird diese Menge zu jeder Instanz gespeichert. Wenn Regeln er-

Algorithmus 6.7 : RL-DS Update: Update der Regeln

Input : neue Instanzen X

$U \leftarrow \emptyset$;
foreach *Instanz* $x \in X$ **do**
  $L \leftarrow \emptyset$;
  foreach *Regel* $r \in$ `rulesOf(`x`)` **do**
    if $c_r = c_x$ **then**
      `// passende Regeln weiter lernen`
      `learnRule(`r`);`
      $L \leftarrow L \cup r$;
    else
      `// unpassende Regeln evtl. verkleinern`
      $p \leftarrow$ `reduce(`r`,`x`);`
      if `quality(`p`)>quality(`r`)` **then**
        $R \leftarrow R \setminus \{r\}$;
        $R \leftarrow R \cup \{p\}$;
        $L \leftarrow L \cup p$;
      end
    end
  end
  `// neue Regel erzeugen, falls keine passende vorhanden`
  if $\forall r \in R.c_r \neq c_x$ **then**
    $r \leftarrow$ `newRule(`x`);`
    $R \leftarrow R \cup r$;
    `learnRule(`r`);`
    $L \leftarrow L \cup r$;
  end
  `// neue und veränderte Regeln auf Löschen testen`
  `checkRemove(`L`);`
  `// potentielle Paare von Regeln zur Vereinigung hinzufügen`
  $U \leftarrow U \cup \{(r_1, r_2) \in$ `rulesOf(`x`)`$^2 | r_1 \neq r_2 \wedge c_{r_1} = c_{r_2}\}$;
end
`unionRules(`U`);`
`checkAllRules();`

Algorithmus 6.8 : RL-DS Update

Input : neue Trainigsinstanz x
Data : Fallbasis I
Data : Regelmenge R
Data : gesammelte neue Trainingsinstanzen X

$X \leftarrow X \cup \{x\}$;
if $|X| = k_{step}$ **then**
    $Update\ der\ Fallbasis(X)$;
    $Update\ der\ Regeln(X)$;
    $X \leftarrow \emptyset$;
end

Algorithmus 6.9 : RL-DS Initialisierung

Input : Initialisierungsmenge T
Data : Fallbasis I
Data : Regelmenge R
Data : gesammelte neue Instanzen X

$X \leftarrow \emptyset$;
$I \leftarrow T$;
$R \leftarrow \emptyset$;
foreach *Instanz* $x \in I$ **do**
    $R \leftarrow R \cup$ `newRule(`x`);`
end
`learnRules();`

weitert, spezialisiert, vereinigt oder gelöscht werden, wird diese Menge betreffender Instanzen aktualisiert.

- $instancesOf(r)$: Liefert zu einer Regel r alle Instanzen, die sie überdeckt. Diese Menge wird ebenfalls zu jeder Regel mitgeführt. Wenn sich die Regel selbst ändert oder Instanzen eingefügt oder gelöscht werden, muss diese Menge aktualisiert werden.

- $generalize(r, i)$: Erweitert eine Regel r um eine Instanz i. Dabei wird die Hyperbox der Regel so vergrößert, dass i überdeckt wird. Für numerische Attribute wird das Intervall, falls es den entsprechenden Attributwert der Instanz nicht beinhaltet, vergrößert und der neue Attributwert bildet nun eine neue Intervallsgrenze. Bei diskreten Attributen wird die Regel auf den gesamten Wertebereich erweitert, falls der Attributwert der Instanz noch nicht überdeckt ist. Dabei können natürlich weitere Instanzen, die bisher nicht in der Regel lagen, überdeckt werden.

- $reduce(r, i)$: Verkleinert die Hyperbox der Regel r soweit, dass die Instanz i nicht mehr überdeckt wird. Dazu wird die Begrenzung der Hyperbox angepasst, die am nächsten zu i liegt.

- $generalize(r_1, r_2)$: Vereinigt die Regeln r_1 und r_2 zu einer neuen Regel. Die neue Regel ist die minimale Regel, die r_1 und r_2 umfasst.

- $newRule(i)$: Erzeugt aus der Instanz i eine neue Regel, die nur diese Instanz überdeckt.

- $overlap(r_1, r_2)$: Ermittelt, ob die beiden Regeln r_1 und r_2 einen gemeinsamen Schnittbereich haben.

- $localrate(r)$: Liefert die Fehlerrate der Regel für die letzten 10 überdeckten Trainingsdaten.

- $distance(r, x)$: Berechnet den kürzesten Abstand zwischen einer Regel r und der Instanz x. Falls r die Instanz x überdeckt, beträgt der Abstand 0. Die genaue Berechnungsfunktion wird im nächsten Unterabschnitt 6.4.2 angegeben. Diese Funktion ist entscheidend zur Ermittlung von Instanzen, die für eine Erweiterung der Regel in Frage kommen.

- $innerDistance(r, x)$: Berechnet, wie zentral die Instanz x in der Regel r liegt. Anschaulich ist dies der Abstand des Zentrums der Regel zu der Instanz. Die genaue Formel wird ebenfalls im nächsten Unterabschnitt 6.4.2 vorgestellt. Diese Funktion ist notwendig, um überdeckte Instanzen zur Verkleinerung der Regel zu bestimmen.

6.4.2 Distanzberechnungen

Zur Berechnung der Distanz zwischen zwei Instanzen wird die Distanzfunktion aus dem letzten Kapitel angewendet (siehe Abschnitt 5.3.3). Bei dem hier vorgestellte Regelverfahren treten darüber hinaus zwei zusätzliche Distanzen auf. Dies ist einmal der Abstand einer Regel zu einer Instanz, die noch nicht von der Regel überdeckt wird und der Abstand des Zentrums einer Regel zu einer Instanz, die bereits überdeckt wird. Beide Distanzen lassen sich über die Formel

$$d^2(r,x) = \sum_{i=1}^{l} \sigma_i^2(r,x_i)$$

berechnen. Allerdings unterscheiden sich die Definitionen der $\sigma_i^2(r,x_i)$, die den Abstand bezüglich eines Attributs angeben. Diese Funktion wird im Folgenden für beide Distanzen definiert:

- Distanz einer Regel r zu einer nicht überdeckten Instanz x ($distance(r,x)$): Die Idee dieser Distanz besteht darin, die kürzeste Entfernung einer Instanz zu dem Rand der Regel zu messen. Der Abstand für die einzelnen Attribute ergibt sich daher wie folgt:
  - Für numerische Attribute A_i:

$$\sigma_i(r,x_i) = \begin{cases} l_i - x_i & \text{für } x_i < l_i \\ x_i - h_i & \text{für } x_i > h_i \\ 0 & \text{sonst} \end{cases}$$

  - Für diskrete Attribute A_i:

$$\sigma_i(r,x_i) = \begin{cases} \delta(x_i,v) & \text{für } B_i = (a_i = v) \\ 0 & \text{für } B_i = (a_i = TRUE) \end{cases}$$

- Distanz einer Instanz x vom Zentrum einer Regel r ($innerDistance(r,x)$): Der Abstand für die einzelnen Attribute ergibt sich in diesem Fall wie folgt:
  - Für numerische Attribute A_i:

$$\sigma_i(r,x_i) = \left| x_i - \frac{l_i + h_i}{2} \right|$$

  - Für diskrete Attribute A_i:

$$\sigma_i(r,x_i) = \begin{cases} 0 & \text{für } B_i = (a_i = x_i) \\ \delta_i(x_i,?) & \text{für } B_i = (a_i = TRUE) \end{cases}$$

max_I	=	1000
max_R	=	300
k_{rule}	=	50
k_{red}	=	10
k_{noise}	=	10
k_{step}	=	5
$p_{current}$	=	50%
p_{young}	=	25%
h	=	100
h_{last}	=	20

Tabelle 6.2: Default-Einstellung der Parameter

6.4.3 Parameter

Wie der instanzbasierte Ansatz des vorhergehenden Kapitels besitzt der Regellerner ebenso einige freie Parameter, die festgelegt werden können und müssen. Der Ursprung der meisten dieser Parameter liegt in der Verwendung einer Fallbasis und damit im instanzbasierten Bereich. Sie beschreiben hauptsächlich beschreiben absolute oder relative Größen von betrachteten Teilmengen. Die freien Parameter des Verfahrens sind: max_I, max_R, k_{rule}, k_{red}, k_{noise}, k_{step}, $p_{current}$, p_{young}, h und h_{last}. Tabelle 6.1 enthält eine Kurzbeschreibung der Parameter, im Text werden diese ausführlicher erklärt. Die Default-Einstellungen der Parameter befindet sich in Tabelle 6.2.

6.4.4 Gütemaß

Beim Lernen von Regeln wird wie bei den meisten Verfahren ein Gütemaß verwendet, um die Güte von Regeln zu vergleichen. RISE verwendet hierbei die Klassifikationsrate der Testdaten. Um dies effizient berechnen zu können, ist dabei allerdings die Verwendung und Organisation unterschiedlicher Datenstrukturen notwendig und lässt sich nicht einfach auf den adaptiven Fall erweitern. In dem hier entwickelten Verfahren wird daher der relative Anteil der Klasse unter allen überdeckten Instanzen mit Laplace-Korrektur als Maß benutzt, das wie folgt definiert ist:

$$quality(r) = \frac{|\{x \in instancesOf(r) | c_x = c_r\}| + 1}{|instancesOf(r)| + |C|},$$

wobei $instancesOf(r)$ die Menge der von r überdeckten Instanzen, c_x die Klasse einer Instanz x, c_r die Klasse von r und C die Menge aller Klassen ist.

6.4.5 Implementierung

Die Implementierung erfolgte wie beim rein instanzbasierten Ansatz mit Hilfe der Data Mining Bibliothek WEKA. Grundlage des adaptiven Lerners ist wieder die abstrakte Klasse *UpdateableClassifier*. Über die Funktion *buildClassifier* wird der Lerner initialisiert, mit *updateClassifier* wird das Modell mit neuen Trainingsdaten aktualisiert und mittels *classifyInstance* lassen sich neue Daten mit dem aktuellen Modell klassifizieren. Auf der Fallbasis müssen sowohl effizient Nächste-Nachbar-Anfragen als auch zeitliche Anfragen gestellt werden können, weshalb ein binärer zeitlich sortierter Rot-Schwarz Baum, sowie der M-Tree zum Indizieren der Daten benutzt werden. Beim M-Tree handelt es sich um die gleiche Variante wie beim instanzbasierten Ansatz (siehe Abschnitt 5.3.3).

6.5 Evaluation

Für die Evaluation des regelbasierten Lerners auf Datenströme wurden die gleichen Datenströmen wie zur Evaluation des instanzbasierten Verfahrens angewendet. Dementsprechend ist die Evaluation sehr ähnlich aufgebaut, wobei hauptsächlich mit IBL-DS verglichen wird, um Gemeinsamkeiten und unterschiedliche Eigenschaften zu diskutieren. Eine zusätzliche Komponente zur Evaluierung ist allerdings die Regelbasis, deren Veränderung ähnlich interessant ist, wie die Fallbasis bei IBL-DS.

6.5.1 Klassifikationsraten

Bei den durchschnittlichen Klassifikationsraten liegen RL-DS und IBL-DS sehr dicht beieinander. Wie zu erwarten war, sind die Ergebnisse für Datenströme mit achsenparallelen Abgrenzungen bzw. mit diskreten Attributen besonders gut. Die absolute Klassifikationsrate für den DISTRIB-Datenstrom zeigen, dass das Speichern von Informationen, die längere Zeit nicht durch neue Trainingsdaten unterstützt werden, deutlich besser als bei allen anderen Verfahren funktioniert. Bei den HYPER- und MEANS-Datenströmen sind die Ergebnisse von RL-DS und IBL-DS ebenfalls sehr ähnlich.

6.5.2 Robustheit der Parameter

Wie schon IBL-DS besitzt leider auch RL-DS eine Vielzahl von Parametern. Diese legen aber nur grobe Einstellungen fest, die unabhängig von dem konkreten Datenstrom vorgenommen werden können. Um diese Robustheit gegenüber den unterschiedlichsten Datenströmen zu zeigen, wurde das Verfahren mit zahlreich veränderten Parametern wie-

	RL-DS	IBL-DS	bestes Vergleichsverfahren	
GAUSS	**.849** (.0058)	.840 (.0017)	LWF04	.838 (.0016)
SINE2	.915 (.0024)	**.926** (.0017)	LWF04	.901 (.0012)
DISTRIB	**.970** (.0130)	.906 (.0079)	LWF02	.894 (.0050)
RANDOM	**.718** (.0083)	.708 (.0019)	LWF04	**.718** (.0013)
MIXED	.902 (.0043)	**.922** (.0019)	LWF04	.895 (.0033)
STAGGER	.966 (.0048)	**.971** (.0027)	WIN200	.924 (.0044)
HYPER2	.963 (.0048)	.957 (.0056)	LWF02	**.974** (.0092)
HYPER5	.856 (.0184)	.865 (.0048)	LWF04	**.892** (.0110)
MEAN2	.936 (.0050)	.934 (.0044)	LWF02	**.946** (.0043)
MEAN5	.816 (.0056)	.794 (.0077)	LWF02	**.830** (.0060)

Tabelle 6.3: Absolute Klassifikationsrate mit Standardabweichung

	RL-DS	IBL-DS	bestes Vergleichsverfahren	
GAUSS	**.812** (.0062)	.804 (.0033)	LWF04	.803 (.0029)
SINE2	.874 (.0030)	**.884** (.0025)	LWF04	.862 (.0022)
DISTRIB	**.955** (.0027)	.931 (.0023)	LWF04	.945 (.0017)
RANDOM	**.718** (.0087)	.709 (.0038)	LWF04	**.718** (.0035)
MIXED	.860 (.0046)	**.878** (.0030)	LWF04	.855 (.0039)
STAGGER	.921 (.0047)	**.925** (.0031)	WIN200	.870 (.0042)
HYPER2	.917 (.0047)	.912 (.0057)	LWF02	**.927** (.0086)
HYPER5	.821 (.0166)	.828 (.0049)	LWF04	**.853** (.0102)
MEAN2	.902 (.0055)	.900 (.0050)	LWF02	**.911** (.0046)
MEAN5	.786 (.0057)	.766 (.0080)	LWF02	**.800** (.0062)

Tabelle 6.4: Klassifikationsrate der Trainingsdaten mit Standardabweichung

derholt angewendet. Eine genaue Beschreibung der Parameter findet sich in Abschnitt 6.4.3, und die Default-Einstellungen der Parameter sind in Tabelle 6.2 zu finden. Das Verfahren wurde für jeden Parameter mit zwei unterschiedlichen sinnvollen Werten nochmals angewendet. Die Ergebnisse sind in Tabelle 6.5 zu finden. Diese zeigen, dass die Ergebnisse auf allen Datenströmen trotz unterschiedlicher Parameter immer sehr ähnlich sind. Dies zeigt, dass die Performanz des Verfahrens relativ robust gegenüber der Parametereinstellung ist.

6.5.3 Update der Regelmenge und der Fallbasis

Dieser Abschnitt betrachtet die Mengen der erzeugten Regeln für die einzelnen Datenströme. Die Tabelle 6.6 listet die durchschnittliche Anzahl von verwendeten Regeln bei den einzelnen Datenströmen auf. Wie zu erwarten, ist die Größe der Regelmenge von dem konkreten Modell und der Anzahl der Dimensionen abhängig. Bei Modellen, die sich gut durch Hyperrechtecke und diskrete Bedingungen beschreiben lassen, wie bei STAGGER und GAUSS, ist die Anzahl der Regeln deutlich geringer als bei komplexeren Klassengrenzen, wie bei SINE2 und MEANS. Besonders großen Einfluss hat dabei allerdings auch die Anzahl der Dimensionen, wie bei MEANS und HYPER aber auch schon bei MIXED zu erkennen ist.

Die Tabelle 6.7 vergleicht die durchschnittliche Anzahl verwendeter Instanzen zwischen RL-DS und IBL-DS. Wie zu erkennen ist, benötigt RL-DS meist deutlich weniger Daten als IBL-DS. Dies lässt sich dadurch erklären, dass das eigentliche Wissen in den Regeln enthalten ist, und die Instanzen nur dazu dienen, die Regeln zu unterstützen.

Die Abbildung 6.5 zeigt den Verlauf der Klassifikationsrate und der Anzahl der Regeln beim STAGGER-Datenstrom, wobei einmal der Zeitraum zwischen der 10.000. und 20.000. Trainingsinstanz und genauer um die 16.000. Trainingsinstanz betrachtet wird. Wie zu erkennen, ist die Anzahl der Regeln konstant sehr niedrig und die Klassifikationsrate großteils auf 1. Wie das Beispiel des Concept Shifts nach der 16.000. Instanz zeigt, funktioniert die Adaption schnell. Das Beispiel zeigt auch, dass bei geeigneter (einfacher) Trennung der Klassen auch einfache und passende Regeln erzeugt werden.

6.5.4 Laufzeiten

Die durchschnittlichen Laufzeiten für 1000 Updates sind in Tabelle 6.8 angegeben, die Zeiten für 1000 Klassifikationen in Tabelle 6.9. Im Vergleich mit IBL-DS zeigt RL-DS ein erwartetes Verhalten. Aufgrund des komplexeren Modells sind die Update-Zeiten höher, dagegen liegen die Klassifikationszeiten deutlich unter denen von IBL-DS. Da ein Update des Modells nur lokal ausgeführt wird, ist der Anstieg der Laufzeit für Updates

	Default	$h = 50$	$h = 150$	$h_{last} = 10$	$h_{last} = 30$	$k_{step} = 2$	$k_{step} = 10$
GAUSS	.849	.847	.848	.848	.848	.849	.848
SINE2	.915	.915	.913	.916	.913	.915	.914
DISTRIB	.970	.968	.968	.968	.969	.969	.975
RANDOM	.718	.718	.695	.704	.697	.701	.693
MIXED	.902	.897	.900	.909	.898	.903	.900
STAGGER	.966	.968	.964	.968	.964	.961	.969
HYPER2	.963	.963	.963	.963	.963	.963	.964
HYPER5	.856	.848	.854	.857	.852	.856	.847
MEAN2	.936	.938	.937	.937	.937	.936	.940
MEAN5	.818	.814	.816	.815	.815	.815	.816

	$k_{rule} = 30$	$k_{rule} = 70$	$k_{red} = 5$	$k_{red} = 15$	$k_{noise} = 5$	$k_{noise} = 15$
GAUSS	.847	.849	.849	.849	.849	.850
SINE2	.913	.915	.914	.915	.915	.915
DISTRIB	.967	.966	.973	.943	.969	.969
RANDOM	.705	.698	.699	.699	.686	.702
MIXED	.898	.905	.903	.903	.903	.903
STAGGER	.969	.962	.965	.965	.967	.967
HYPER2	.964	.963	.960	.966	.963	.963
HYPER5	.838	.860	.853	.854	.855	.855
MEAN2	.938	.938	.935	.935	.937	.936
MEAN5	.814	.816	.814	.816	.816	.816

	$p_{current} = 30\%$	$p_{current} = 70\%$	$p_{young} = 15\%$	$p_{young} = 40\%$	$max_I = 750$	$max_I = 1500$
GAUSS	.849	.850	.840	.848	.848	.848
SINE2	.915	.915	.914	.918	.915	.915
DISTRIB	.975	.967	.978	.951	.951	.981
RANDOM	.701	.700	.676	.687	.701	.701
MIXED	.903	.902	.896	.906	.903	.903
STAGGER	.965	.965	.953	.974	.965	.965
HYPER2	.964	.963	.962	.965	.963	.963
HYPER5	.855	.855	.822	.856	.867	.840
MEAN2	.936	.935	.935	.937	.937	.937
MEAN5	.814	.816	.818	.816	.809	.816

Tabelle 6.5: Absolute Klassifikationsrate für RL-DS mit verschiedenen Parametern

	RL-DS
GAUSS	24.6
SINE2	33.4
DISTRIB	58.4
RANDOM	29.9
MIXED	44.2
STAGGER	3.3
HYPER2	17.0
HYPER5	274
MEAN2	42.6
MEAN5	238

Tabelle 6.6: Mittlere Anzahl verwendeter Regeln

	RL-DS	IBL-DS
GAUSS	254	638
SINE2	369	659
DISTRIB	619	3513
RANDOM	186	508
MIXED	422	613
STAGGER	139	3405
HYPER2	213	3549
HYPER5	911	2745
MEAN2	544	2007
MEAN5	971	2433

Tabelle 6.7: Vergleich der durchschnittlichen Größe der Fallbasis

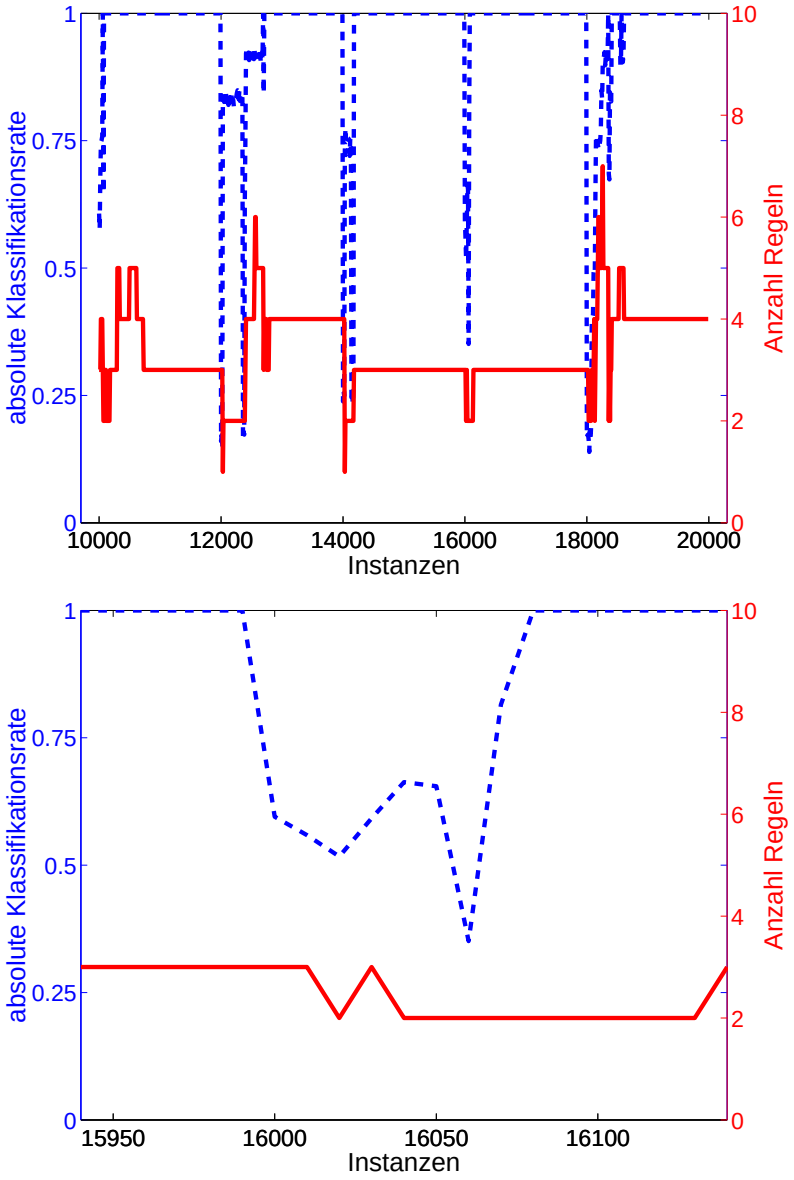

Abbildung 6.5: Vergleich von Klassifikationsrate und Anzahl der Regeln
Dieses Beispiel zeigt die absolute Klassifikationsrate (blau gestrichelt) und die Anzahl der Regeln (rot durchgehend) von RL-DS beim STAGGER-Datenstrom im Bereich der 10.000. und 20.000. (oben) bzw. um der 16.000. Trainingsinstanz (unten).

	RL-DS	IBL-DS
GAUSS	1.52	.56
SINE2	1.94	.36
DISTRIB	3.14	.49
RANDOM	1.10	.33
MIXED	7.41	.63
STAGGER	5.09	.70
HYPER2	1.38	.50
HYPER5	17.8	4.3
MEANS2	3.28	.47
MEANS5	23.2	3.2

Tabelle 6.8: Laufzeit für 1000 Updates (in Sekunden)

	RL-DS	IBL-DS
GAUSS	.013	.17
SINE2	.016	.090
DISTRIB	.023	.13
RANDOM	.014	.078
MIXED	.028	.16
STAGGER	.006	.19
HYPER2	.010	.13
HYPER5	.16	1.3
MEANS2	.019	.12
MEANS5	.14	.92

Tabelle 6.9: Laufzeit für 1000 Klassifikationen (in Sekunden)

bei höheren Dimensionen relativ gering. Je nach Laufzeitanforderungen für Updates und Klassifikationen kann damit RL-DS oder IBL-DS geeigneter sein.

6.5.5 Real-World-Daten

Als Real-World-Daten wurden ebenso die Daten des letzten Kapitels angewendet. Eine genaue Beschreibung der Daten und der Experimente sind dort im Abschnitt 5.4.6 zu finden. Die Ergebnisse für die modifizierten UCI-Daten finden sich in Tabelle 6.10. Die Ergebnisse sind dabei sehr verschieden. Während das Ergebnis für die Balance-Daten deutlich besser als bei IBL-DS ist, liegen die Resultate für Car und Nursery darunter. Anders sieht dies bei den Aktiendaten in Tabelle 6.11 aus. Hier liefert RL-DS von allen Verfahren das beste Ergebnis. Diese Tests zeigen auch, dass die Wahl des optimalen Lernverfahrens stark von den angewendeten Daten abhängig ist.

	Balance	Car	Nursery
RL-DS	0.850	0.752	0.850
IBL-DS	0.805	0.889	0.900
LWF04	0.782	0.700	0.899
LWF10	0.846	0.700	0.842
Win50	0.813	0.819	0.827
Win100	0.815	0.862	0.856
Win200	0.785	0.888	0.878
TWF99	0.795	0.889	0.877
TWF995	0.787	0.887	0.900
kNN	0.693	0.745	0.839

Tabelle 6.10: Ergebnisse für Datenströme aus UCI-Datensätze

Verfahren	Trefferquote
RL-DS	0.596
IBL-DS	0.563
LWF02	0.569
LWF04	0.559
Win200	0.519
Win400	0.528
Win800	0.529
TWF996	0.520
TWF998	0.520
kNN	0.539

Tabelle 6.11: Ergebnisse für Datenstrom aus Aktienkursen

6.6 Resümee

Der Anfang des Kapitels beschäftigte sich mit modellbasiertem Lernen und mit einer Idee, die modellbasierte Verfahren mit dem adaptiven instanzbasierten Ansatz kombiniert. Dies wurde anschließend anhand des Regellerners RL-DS umgesetzt. Hierbei werden sowohl eine Fallbasis als auch eine Regelmenge gespeichert und bei neuen Trainingsdaten aktualisiert. Das Verfahren verwendet dabei Hyper-Rechtecke und eine Distanzfunktion, die auf dem Regellernverfahren RISE beruhen. Das Besondere beim Lernen der Regeln im Vergleich zu den bisher bekannten Verfahren ist, dass Regeln nicht nur verallgemeinert, sondern auch wieder spezialisiert werden können. Alle bisherigen Verfahren müssen zu allgemeine Regeln löschen und neue Regeln lernen, was bei kontinuierlicher Veränderung ständig passieren kann. Dadurch wurde eine sehr gute Adaption auch bei numerischen Datenströmen mit relativ schnellem Concept Drift erreicht. Wie die Klassifikationsraten zeigen, kann RL-DS durchaus mit dem flexiblen instanzbasierten Lerner IBL-DS konkurrieren. Wie man es von einem modellbasierten Lerner erwartet, ist die Klassifikation auch deutlich schneller als beim instanzbasierten Verfahren. Das Verfahren bietet damit in adaptiven Umgebungen, in denen deutlich mehr Daten zu klassifizieren als zu lernen sind, Vorteile zum instanzbasierten Ansatz.

Kapitel 7

Diskussion und Ausblick

Dieses Kapitel dient dazu, die Arbeit noch einmal im Ganzen zu betrachten. Dazu gehört zunächst eine komplette Zusammenfassung des Inhalts. Anschließend werden die Ergebnisse ausgewertet und diskutiert. Zum Schluss folgt ein Ausblick auf sinnvolle Arbeiten, die sich hieran anschließen könnten und sollten.

7.1 Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Thema „Data Mining auf Datenströmen“, ein Bereich, in dem in den letzten Jahren immer mehr Forschung betrieben wurde. Im 2. Kapitel wurde eine Einführung in das Themengebiet gegeben, wobei die Bereiche „Data Mining“ und „Datenströme“ zunächst für sich beschrieben wurden. Neben den Grundlagen wurden auch die besonderen Problemstellungen, die bei der Arbeit mit Datenströmen existieren, erläutert und einige häufig verwendete Strategien vorgestellt.

Das 3. Kapitel beschäftigt sich mit bisherigen Arbeiten zu einigen Aufgabenstellungen im Bereich Datenströme, darunter auch die zwei Gebiete, mit denen sich die späteren Kapitel beschäftigen: Clustering und Klassifikation. Zunächst wurden die einzelnen Aufgabenstellungen beschrieben und anschließend vorhandene Lösungsansätze vorgestellt und diskutiert. Wie im Fall von Frequent-Itemset-Mining und Clusterverfahren, zeigt sich, dass oftmals zunächst nicht alle Einschränkungen, die für die Verarbeitung von Datenströmen gelten, eingehalten wurden. Erst nach weiteren Forschungstätigkeiten kamen Lösungen zustande, die für echte Datenströme geeignet sind. In anderen Verfahren, wie bei der Klassifikation, wurden zunächst einfachere Lösungen erarbeitet, die aber nicht für beliebige Datenströme geeignet sind. Auch hier führten erst weitere Forschungsarbeiten zu besseren und allgemeineren Verfahren. Dieses Kapitel betrachtete auch das Problem der Evaluierung von derartigen Verfahren an. Das Auffinden von Real-World-Datenströmen, die zu den Problemstellungen passen, stellt hierbei das größte Problem dar. Dies ist auch darin begründet, dass Datenströme oft nur ungern von Unternehmen und Institutionen zur Verfügung gestellt werden, da sie als zu sensibel und kritisch gelten.

In den folgenden drei Kapiteln wurden eigene Ansätze zu speziellen Problemstellungen vorgestellt. Im 4. Kapitel wurde eine Problemstellung aus dem Bereich Clustering betrachtet. Das besondere hierbei ist, dass die hier konkret betrachtete Problemstellung zuvor noch nicht in Forschungsarbeiten untersucht wurde. Hierbei geht es darum, mehrere Datenströme anhand ihres aktuellen Verlaufs zu clustern. Dabei gab es zahlreiche kleinere Probleme zu lösen, um das Ziel eines effizienten und flexiblen Verfahrens zu erreichen. Die Berechnung der Distanzen zwischen Datenströmen wurde durch eine adaptiven DFT-Komprimierung erreicht. Dabei wurde ein bereits existierendes Verfahren um eine Gewichtung über dem aktuellen Zeitfenster erweitert. Für das Clustering selbst wurde auf das effiziente Fuzzy-C-Means-Verfahren zurückgegriffen. Da sich die Datenströme und damit auch die Clusterstrukturen ständig (graduell) verändern können, ist die

Verwendung der Fuzzy-Variante sinnvoll. Ein weiteres Problem, dass hierbei auftauchte, war die Bestimmung der Clusteranzahl. Diese muss sich während des gesamten Clusterings flexibel verändern und optimieren lassen. Fuzzy-C-Means verlangt dagegen eine vorherige Festlegung der Clusteranzahl. Deshalb war es notwendig, das Verfahren zu erweitern. Es wurde ermöglicht, dass sich die Clusteranzahl in jedem Clusterschritt um eins verändern kann, wofür jedesmal nach einer guten Clusterstruktur mit einem Cluster mehr und einem weniger gesucht wurde. Mithilfe einer Gütefunktion ist es möglich zu entscheiden, ob eine der Clusterstrukturen und die damit verbundenen Clusteranzahlen besser als die aktuelle ist. Da sich existierende Gütefunktion als nicht ausreichend geeignet zeigten, wurde die am häufigsten benutzte Xie-Beni-Funktion modifiziert und auf die besonderen Bedingungen angepasst. Da es für die betrachtete Problemstellung noch kein Vergleichsverfahren gibt, war auch kein direkter Vergleich in der Evaluation möglich. Mithilfe von erzeugten Datenströmen wurde deshalb das Verhalten des Verfahrens auf Plausibilität überprüft. Dabei wurde sowohl die Anpassung der Clusterstruktur wie auch der Clusteranzahl untersucht. Für ein einfaches Beispiel mit Aktienkursen und UCI-Daten wurde auch die Anwendbarkeit auf Real-World Daten demonstriert. Am Ende folgte eine Diskussion und Beschreibung zu möglichen Erweiterungen.

Das 5. und 6. Kapitel widmen sich der adaptiven Klassifikation auf Datenströmen, wobei ein rein instanzbasiertes Lernverfahren und ein Regellerner entwickelt wurden. Als erstes wurden die besonderen Probleme dieser Fragestellung angesprochen, insbesondere das Problem der Erkennung von Concept Drift. Dadurch unterscheidet sich die Klassifikation auf Datenströmen deutlich von der Klassifikation auf statischen Daten. Die besondere Schwierigkeit wird auch bei der Betrachtung der bisher veröffentlichten Verfahren deutlich. Die meisten Ansätze aktualisieren ihr Modell auf immer gleiche Weise, ohne tatsächlich vorhandenen Concept Drift zu erkennen und darauf entsprechend zu reagieren. Ein Einfluss besteht nur bei der Wahl der Parameter, die festgelegt werden müssen, bevor die Klassifikation gestartet wird. Da das besondere der Verarbeitung von Datenströmen darin besteht, dass sie kontinuierlich und potentiell beliebig lang abläuft, ist aber im Allgemeinen eine optimierte Auswahl der Parameter vorher nicht möglich. Nur sehr wenige Verfahren versuchen auch konkret auftretenden Concept Drift zu erkennen und entsprechend zu reagieren. Da zum adaptiven Lernen ein äußerst flexibles Lernverfahren notwendig ist, wurde der instanzbasierte Ansatz als Grundlage für die Entwicklung des vorgestellten Verfahrens IBL-DS verwendet. Es erkennt und behandelt unterschiedliche Arten von Concept Drift. Es ist sehr flexibel anwendbar, da es sowohl mit numerischen wie auch diskreten Attributen umgehen kann. Bei der Evaluation wurde besonderen Wert darauf gelegt, dass Datenströme mit unterschiedlichen Formen von Concept Drift und mit numerischen und diskreten Attributen zum Einsatz kommen. Obwohl das neue Verfahren im Gegensatz zu den Vergleichsverfahren auch den konkreten Concept Drift erkennen und passend adaptieren muss, sind die Ergebnisse durchweg konkurrenzfähig. Für die Anwendbarkeit auf Real-World-Daten wurden zwei verschiedene Experimente durchgeführt, einmal mit Standard UCI-Datensätzen, denen Concept Drift hinzugefügt wurde, und mit

Aktienkursen von Automobilherstellern. Auch hier wurde die praktische Anwendbarkeit anhand der guten und vergleichbaren Ergebnisse gezeigt. Am Ende des 5. Kapitels wurden noch zwei mögliche Erweiterungen vorgeschlagen und diskutiert.

Das 6. Kapitel beginnt mit der Diskussion der Idee, modellbasierte Verfahren mit instanzbasierten Verfahren für die adaptive Klassifikation zu kombinieren. Der Kern der Idee ist, dass neben dem zu lernenden Modell eine kleine optimierte Fallbasis gelernt wird, auf der das Modell basiert und die Vorteile bei der Erkennung und Adaption von Concept Drift bietet. Bei dem anschließend vorgestellten adaptiven Regelverfahren RL-DS wurde diese Idee umgesetzt. Das Verfahren beruht dabei auf einer Instanzbasis und einer Regelbasis, die beide und unter gegenseitiger Zuhilfenahme bei jedem Update aktualisiert werden. Die Evaluation beinhaltet hauptsächlich einen Vergleich von RL-DS mit dem instanzbasierten Verfahren IBL-DS, wobei die schon im vorherigen Kapitel verwendeten Datenströme benutzt werden. Darüber hinaus wird auch anhand eines Datenstroms der Umfang und die Adaption der erlernten Regelmenge untersucht.

7.2 Diskussion und Ausblick

Der Überblick wie auch die vorgestellten Verfahren geben einen Einblick in den Bereich von Data Mining auf Datenströmen. Es gibt sehr viele interessante Aufgabenstellungen, die bislang gar nicht oder nur unzureichend erforscht wurden. Immer mehr Forscher beschäftigen sich mit diesen Bereich, wodurch auch immer mehr wissenschaftliche Arbeiten dazu publiziert werden. Es werden aber auch die Probleme dieses Forschungsbereichs deutlich. Für viele Aufgabenstellungen fehlen die konkreten Anwendungen, um den praktischen Nutzen der Verfahren überprüfen und sehen zu können. Die meisten Arbeiten beschränken sich weiterhin mit der vagen Diskussion über mögliche Anwendungen, verwenden aber nur synthetische oder speziell optimierte Daten in den Experimenten, die keinen praktischen Nutzen zeigen können. Um eine weitere starke Zunahme der Forschungstätigkeiten in diesem Bereich zu begründen, ist es sicherlich notwendig, sich auch konzentrierter mit konkreten Anwendungen der einzelnen Verfahren zu beschäftigen. Dies wird vor allem dann notwendig, wenn immer speziellere Problemstellungen behandelt werden.

Die folgenden Unterabschnitte beschäftigen sich nun mit den konkret in dieser Arbeit vorgestellten Verfahren, die sicherlich auch von der erwähnten Problematik betroffen sind. Sie betrachten die mit den Verfahren erreichten Ziele aber auch den Bedarf an weiteren und an diese Arbeit möglicherweise anschließenden Forschungsthemen.

7.2.1 Clustering von Datenströmen

Das für diese Anwendung entwickelte Clusterverfahren beinhaltete die Lösung zahlreicher Teilprobleme, wie die sich dynamisch verändernde Clusteranzahl, die passende Fuzzy-Gütefunktion, das Distanzmaß für Fuzzy-Clusterpartitionen und die adaptive Komprimierung der Datenströme zur effizienten Speicherung und Distanzberechnung. Diese Probleme wurden allesamt für diese Anwendung ausreichend gelöst, so dass ein anwendbares Verfahren entstand, dass das Erwartete leistet. Jedes der einzelnen Problemlösungen ist allerdings auch für weitere Anwendungen verwendbar und bietet damit Raum für weitere Forschungstätigkeiten. Gerade das Problem der sich dynamisch anpassenden Clusteranzahl ist für zahlreiche Anwendungen von Interesse und sollte deshalb weiter vertieft werden. Von Interesse ist auch die Frage, welche weiteren Clusterverfahren (z.B. SOM's und DBSCAN) sich für eine dynamische Umgebung anpassen lassen. Ebenso könnten auch weitere adaptive Komprimierungsverfahren und -strategien untersucht werden.

7.2.2 IBL-DS

Der instanzbasierte Lerner IBL-DS ist verständlicherweise deutlich komplexer als ein Lerner, der pauschal auf Concept Drift reagiert. Die Ergebnisse zeigen ebenso, dass dieser ohne Änderung von Parametern sehr flexibel einsetzbar ist und für unterschiedlichste Bedingungen relativ gute Ergebnisse erzeugt. Bei der Entwicklung des Verfahrens wurden die zahlreichen Probleme bei der Erkennung von verschiedenen Arten von Concept Drift deutlich. Trotzdem kann das vorgestellte Verfahren nur als ein erster Ansatz betrachtet werden, und es ist durchaus zu erwarten, dass es einfachere, effizientere und bessere Verfahren gibt, die es zu finden gilt. Ein erster Ansatz wäre, genauer zu analysieren, welche Maßnahme bei welcher Art von Concept Drift entscheidend für eine gute Erkennung und Behandlung ist, und welches die einfachsten Varianten sind. Außerdem müssen die Mängel der bisherigen Ansätze noch weiter aufgedeckt und analysiert werden, um sie beheben zu können.

7.2.3 RL-DS

Das Beispiel des regelbasierten Verfahrens RL-DS zeigt ebenso die scheinbar unvermeidliche Komplexität eines nicht pauschalen, modellbasierten Verfahrens zur Erkennung und Behandlung von Concept Drift in der Klassifikation. Die Grundidee zum Lernen und Optimieren von Regeln ist sicherlich einfach, trotzdem wird das Verfahren bei der Umsetzung relativ komplex. Hierbei ist das Update der Fallbasis mit entscheidend, da hierfür trotz Regeln immer noch konkrete Nächste-Nachbar-Umgebungen zu betrachten sind. Dies ist auch der Hauptgrund für die leider immer noch relativ hohe Anzahl an Parametern. Die

Ergebnisse durchaus vergleichbar mit denen des instanzbasierten Verfahrens. Besonders bei Datenströmen mit achsenparallelen Klassengrenzen oder diskreten Attributen zeigen sich die Stärken dieses modellbasierten Ansatzes. Neben den guten Klassifikationsraten und der effizienten Klassifikation ist auch die erzeugte Regelbasis relativ klein und damit potentiell zur weiteren inhaltlichen Verarbeitung geeignet. Der Ansatz, die Instanzbasis und das Modell parallel zu optimieren, um flexible modellbasierte adaptive Lerner zu erzeugen, hat sich aber sicherlich bewährt. Die Tatsache, dass das Lernen deutlich aufwändiger ist als bei rein instanzbasierten Verfahren, war zu erwarten und ist vom Lernen auf statischen Daten schon bekannt. Der Nutzen eines modellbasierten Verfahrens liegt bei der deutlich schnelleren Klassifikation und einer besseren Verallgemeinerung, zumindest wenn das Modell geeignet ist, das wahre Konzept effizient und ausreichend genau zu beschreiben. Eine bessere Verständlichkeit des Modells gegenüber den instanzbasierten Verfahren spielt beim adaptiven Lernen aber eher eine geringe Rolle, zumindest wenn Concept Drift regelmäßig auftritt. In diesen Fällen bleibt keine Zeit, die sich ständig ändernden Modelle jeweils gründlich zu analysieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Literaturverzeichnis

[ABB+03] A. ARASU, B. BABCOCK, S. BABU, M. DATAR, K. ITO, I. NISHIZAWA, J. ROSENSTEIN und J. WIDOM: *STREAM: the stanford stream data manager (demonstration description)*. In: *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, Seiten 665–665, New York, NY, USA, 2003. ACM Press.

[ACC+03] D.J. ABADI, D. CARNEY, U. CETINTEMEL, M. CHERNIACK, C. CONVEY, S. LEE, M. STONEBRAKER, N. TATBUL und S. ZDONIK: *Aurora: a new model and architecture for data stream management*. The VLDB Journal, 12(2):120–139, 2003.

[AHWY03] C.C. AGGARWAL, J. HAN, J. WANG und P.S. YU: *A Framework for Clustering Evolving Data Streams.* In: *VLDB*, Seiten 81–92, 2003.

[AIS93] R. AGRAWAL, T. IMIELINSKI und A.N. SWAMI: *Mining Association Rules between Sets of Items in Large Databases*. In: PETER BUNEMAN und SUSHIL JAJODIA (Herausgeber): *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Seiten 207–216, Washington, D.C., 26–28 1993.

[AM04] A. ARASU und G.S. MANKU: *Approximate counts and quantiles over sliding windows*. In: *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Seiten 286–296, New York, NY, USA, 2004. ACM Press.

[AS94] R. AGRAWAL und R. SRIKANT: *Fast Algorithms for Mining Association Rules*. In: JORGE B. BOCCA, MATTHIAS JARKE und CARLO ZANIOLO (Herausgeber): *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, Seiten 487–499. Morgan Kaufmann, 12–15 1994.

[BBD+01] J. BERCKEN, B. BLOHSFELD, J. DITTRICH, J. KRÄMER, T. SCHÄFER, M. SCHNEIDER und B. SEEGER: *XXL - A library approach to supporting e.cient implementations of advanced database queries*. In: *Proceedings of the VLDB*, Seiten 39–48, 2001.

[BBD+02] B. BABCOCK, S. BABU, M. DATAR, R. MOTWANI und J. WIDOM: *Models and Issues in Data Stream Systems*. In: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, Seiten 1–16. ACM, 2002.

[BDM02] B. BABCOCK, M. DATAR und R. MOTWANI: *Sampling from a moving window over streaming data*. In: *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, Seiten 633–634, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[BDMO03] B. BABCOCK, M. DATAR, R. MOTWANI und L. O'CALLAGHAN: *Maintaining variance and k-medians over data stream windows.* In: *PODS*, Seiten 234–243, 2003.

[Bel61] R.E. BELLMAN: *Adaptive Control Processes: A Guided Tour.* Princeton Universsity Press, 1961.

[Ben75] J.L. BENTLEY: *Multidimensional binary search trees used for associative searching*. Commun. ACM, 18(9):509–517, 1975.

[BF98] P.S. BRADLEY und U.M. FAYYAD: *Refining Initial Points for K-Means Clustering*. In: *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, Seiten 91–99, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[BH06] J. BERINGER und E. HÜLLERMEIER: *Online Clustering of Parallel Data Streams*. Data Knowl. Eng., 58(2):180–204, 2006.

[BH07] J. BERINGER und E. HÜLLERMEIER: *Efficient instance based learning on data streams*. Intell. Data Anal., (to appear), 2007.

[Bor05] C. BORGELT: *Prototype-based Classification and Clustering*. Habilitationsschrift, Otto-von-Guericke-Universität Magdeburg, 2005.

[BW69] B.G. BACHELOR und B.R. WILKINS: *Method for Location of Clusters of Patterns to Initiate a Learning Machine*. Electronics Letters, 5:481–483, 1969.

[CCD+03] S. CHANDRASEKARAN, O. COOPER, A. DESHPANDE, M.J. FRANKLIN, J.M. HELLERSTEIN, W. HONG, S. KRISHNAMURTHY, S. MADDEN, V. RAMAN, F. REISS und M.A. SHAH: *TelegraphCQ: Continuous Dataflow Processing for an Uncertain World.* In: *CIDR*, 2003.

[CCP+04] Y.D. CAI, D. CLUTTER, G. PAPE, J. HAN, M. WELGE und L. AUVIL: *MAIDS: mining alarming incidents from data streams*. In: *SIGMOD '04:*

Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Seiten 919–920, New York, NY, USA, 2004. ACM Press.

[CD97] S. CHAUDHURI und U. DAYAL: *An Overview of Data Warehousing and OLAP Technology*. SIGMOD Record, 26(1):65–74, 1997.

[CDTW00] J. CHEN, D.J. DEWITT, F. TIAN und Y. WANG: *NiagaraCQ: a scalable continuous query system for Internet databases*. In: *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, Seiten 379–390, New York, NY, USA, 2000. ACM Press.

[CEQZ06] F. CAO, M. ESTER, W. QIAN und A. ZHOU: *Density-Based Clustering over an Evolving Data Stream with Noise.* In: *SDM*. SIAM, 2006.

[CFPR00] C. CORTES, K. FISHER, D. PREGIBON und A. ROGERS: *Hancock: a language for extracting signatures from data streams*. In: *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 9–17, New York, NY, USA, 2000. ACM Press.

[CGM03] G. CASTILLO, J. GAMA und P. MEDAS: *Adaptation to Drifting Concepts*. In: *Progress in Artificial Intelligence, 11th Portuguese Conference on Artificial Intelligence, EPIA 2003, Lecture Notes in Artificial Intelligence, v. 2902*, Seiten 279–293. Springer Verlag, 2003.

[CL03] J.H. CHANG und W.S. LEE: *Finding recent frequent itemsets adaptively over online data streams*. In: *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 487–492, New York, NY, USA, 2003. ACM Press.

[CL04] J.H. CHANG und W.S. LEE: *A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams.* J. Inf. Sci. Eng., 20(4):753–762, 2004.

[CL05] J.H. CHANG und W.S. LEE: *estWin: Online data stream mining of recent frequent itemsets by sliding window method*. J. Inf. Sci., 31(2):76–90, 2005.

[CN89] PETER CLARK und TIM NIBLETT: *The CN2 Induction Algorithm*. Machine Learning, 3:261–283, 1989.

[Coh95] W.W. COHEN: *Fast Effective Rule Induction*. In: ARMAND PRIEDITIS und STUART RUSSELL (Herausgeber): *Proc. of the 12th International Conference on Machine Learning*, Seiten 115–123, Tahoe City, CA, Juli 9–12, 1995. Morgan Kaufmann.

[CPRZ97] P. CIACCIA, M. PATELLA, F. RABITTI und P. ZEZULA: *Indexing metric spaces with M-tree*. In: MATTEO CRISTIANI und LETIZIA TANCA (Herausgeber): *Atti del Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD'97)*, Seiten 67–86, Verona, Italy, Juni 1997.

[CWYM04] Y. CHI, H. WANG, P.S. YU und R.R. MUNTZ: *Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window*. icdm, 00:59–66, 2004.

[DH00] P. DOMINGOS und G. HULTEN: *Mining high-speed data streams*. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 71–80. ACM Press, 2000.

[DH03] P. DOMINGOS und G. HULTEN: *A General Framework for Mining Massive Data Streams*. Journal of Computational and Graphical Statistics, 12(4):945–949, 2003.

[DNM98] C.L. BLAKE D.J. NEWMAN, S. HETTICH und C.J. MERZ: *UCI Repository of machine learning databases*, 1998.

[Dom95] P. DOMINGOS: *Rule Induction and Instance-Based Learning: A Unified Approach*. In: *IJCAI*, Seiten 1226–1232, 1995.

[Dom96] P. DOMINGOS: *Unifying Instance-Based and Rule-Based Induction*. Machine Learning, 24(2):141–168, 1996.

[Dom97] P. DOMINGOS: *A Unified Approach to Concept Learning*. Ph.D. dissertation, University of Washington, 1997.

[FK97] H. FRIGUI und R. KRISHNAPURAM: *Clustering by competitive agglomeration.* Pattern Recognition, 30(7):1109–1119, 1997.

[FPSS96] U.M. FAYYAD, G. PIATETSKY-SHAPIRO und P. SMYTH: *From data mining to knowledge discovery: an overview*. Seiten 1–34, 1996.

[FTARR06] F. FERRER-TROYANO, J.S. AGUILAR-RUIZ und J.C. RIQUELME: *Data streams classification by incremental rule learning with parameterized generalization*. In: *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, Seiten 657–661, New York, NY, USA, 2006. ACM Press.

[FW94] J. FURNKRANZ und G. WIDMER: *Incremental Reduced Error Pruning*. In: *Proceedings the Eleventh International Conference on Machine Learning*, Seiten 70–77, New Brunswick, NJ, 1994.

[GE03] I. GUYON und A. ELISSEEFF: *An Introduction to Variable and Feature Selection*. Journal of Machine Learning Research, 3:1157–1182, 2003.

[GGRL99] J. GEHRKE, V. GANTI, R. RAMAKRISHNAN und W.-Y. LOH: *BOAT-Optimistic Decision Tree Construction.* In: *SIGMOD Conference*, Seiten 169–180, 1999.

[GK01] M. GREENWALD und S. KHANNA: *Space-efficient online computation of quantile summaries*. In: *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, Seiten 58–66, New York, NY, USA, 2001. ACM Press.

[GLZT05] J. GAO, J. LI2, Z. ZHANG und P.-N. TAN: *An Incremental Data Stream Clustering Algorithm Based on Dense Units Detection*. In: *PAKDD 2005: Advances in Knowledge Discovery and Data Mining*, Seiten 420–425. Springer, 2005.

[GMCR04] J. GAMA, P. MEDAS, G. CASTILLO und P. RODRIGUES: *Learning with Drift Detection.* In: *SBIA*, Seiten 286–295, 2004.

[GMM+03] S. GUHA, A. MEYERSON, N. MISHRA, R. MOTWANI und L. O'CALLAGHAN: *Clustering Data Streams: Theory and Practice.* IEEE Trans. Knowl. Data Eng., 15(3):515–528, 2003.

[GMMO00] S. GUHA, N. MISHRA, R. MOTWANI und L. O'CALLAGHAN: *Clustering Data Streams*. In: *IEEE Symposium on Foundations of Computer Science*, Seiten 359–366, 2000.

[GMR05] J. GAMA, P. MEDAS und P. RODRIGUES: *Learning decision trees from dynamic data streams*. In: *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, Seiten 573–577, New York, NY, USA, 2005. ACM Press.

[Hid99] C. HIDBER: *Online association rule mining*. In: *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, Seiten 145–156, New York, NY, USA, 1999. ACM Press.

[HK00] J. HAN und M. KAMBER: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000. HAN j2 01:1 1.Ex.

[Hoe63] W. HOEFFDING: *Probability inequalities for sums of bounded random variables.* Journal of the American Statistical Association, 58:13–30, 1963.

[HSD01] G. HULTEN, L. SPENCER und P. DOMINGOS: *Mining time-changing data streams*. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 97–106. ACM Press, 2001.

[KBDG04] D. KIFER, S. BEN-DAVID und J. GEHRKE: *Detecting Change in Data Streams.* In: *VLDB*, Seiten 180–191, 2004.

[KGK95] R. KRUSE, J. GEBHARDT und F. KLAWONN: *Fuzzy-Systeme*. Leitfaeden der Informatik. Teubner, Stuttgart, 2. Auflage, 1995.

[Kli04] R. KLINKENBERG: *Learning Drifting Concepts: Example Selection vs. Example Weighting*. Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift, 8(3):281–300, 2004.

[KM03] J.Z. KOLTER und M.A. MALOOF: *Dynamic Weighted Majority: A new ensemble method for tracking concept drift*. In: *Proceedings of the Third IEEE International Conference on Data Mining*, Seiten 123–130, Los Alamitos, CA, 2003. IEEE Press.

[KP99] R. KOTHARI und D. PITTS: *On finding the number of clusters*. Pattern Recogn. Lett., 20(4):405–416, 1999.

[KS04] J. KRÄMER und B. SEEGER: *PIPES: a public infrastructure for processing and exploring streams*. In: *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Seiten 925–926, New York, NY, USA, 2004. ACM Press.

[KW95] M. KUBAT und G. WIDMER: *Adapting to Drift in Continuous Domains*. Lecture Notes in Computer Science, 912:307ff., 1995.

[KY95] G.J. KLIR und B. YUAN: *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, May 1995.

[LLXY04] X. LIN, H. LU, J. XU und J. XU YU: *Continuously Maintaining Quantile Summaries of the Most Recent N Elements over a Data Stream*. In: *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, Seite 362, Washington, DC, USA, 2004. IEEE Computer Society.

[LPT00] L. LIU, C. PU und W. TANG: *WebCQ: Detecting and Delivering Information Changes on the Web*. In: *CIKM*, Seiten 512–519, 2000.

[LVB04] M. LAZARESCU, S. VENKATESH und H.H. BUI: *Using multiple windows to track concept drift.* Intell. Data Anal., 8(1):29–59, 2004.

[LZ05] Y.-N. LAW und C. ZANIOLO: *An Adaptive Nearest Neighbor Classification Algorithm for Data Streams*. In: A. JORGE, L. TORGO, P. BRAZDIL, R. CAMACHO und J. GAMA (Herausgeber): *Knowledge Discovery in Databases: PKDD 2005*, Seiten 108–120. Springer-Verlag, 2005.

[Mal03] M.A. MALOOF: *Incremental rule learning with partial instance memory for changing concepts*. In: *Proceedings Of The International Joint Conference On Neural Networks 2003*, Band 4, Seiten 2764–2769, 2003.

[Mit97] T. MITCHELL: *Machine Learning*. McGraw Hill, 1997.

[MM02] G.S. MANKU und R. MOTWANI: *Approximate Frequency Counts over Data Streams*. In: *Analyzing Data Streams - VLDB 2002*, Seiten 346–357. Springer, 2002.

[MM04] M.A. MALOOF und R.S. MICHALSKI: *Incremental learning with partial instance memory*. Artif. Intell., 154(1-2):95–126, 2004.

[MMS04] M. MOTOYOSHI, T. MIURA und I. SHIOYA: *Clustering stream data by regression analysis*. In: *ACSW Frontiers '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, Seiten 115–120, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[MRL98] G.S. MANKU, S. RAJAGOPALAN und B.G. LINDSAY: *Approximate medians and other quantiles in one pass and with limited memory*. Seiten 426–435, 1998.

[OMM+02] L. O'CALLAGHAN, A. MEYERSON, R. MOTWANI, N. MISHRA und S. GUHA: *Streaming-Data Algorithms for High-Quality Clustering*. icde, 00:0685, 2002.

[PR67] L.I. PRESS und M.S. ROGERS: *IDEA-a conversational, heuristic program for Inductive Data Exploration and Analysis*. In: *Proceedings of the 1967 22nd national conference*, Seiten 35–40, New York, NY, USA, 1967. ACM Press.

[Qui93] J.R. QUINLAN: *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[Sal91] S. SALZBERG: *A Nearest Hyperrectangle Learning Method.* Mach. Learn., 6(3):251–276, 1991.

[Sal97] M. SALGANICOFF: *Tolerating Concept and Sampling Shift in Lazy Learning Using Prediction Error Context Switching*. Artif. Intell. Rev., 11(1-5):133–155, 1997.

[SD04] T. SU und J. DY: *A Deterministic Method for Initializing K-Means Clustering*. ictai, 00:784–786, 2004.

[SK01] W.N. STREET und Y.S. KIM: *A streaming ensemble algorithm (SEA) for large-scale classification*. In: *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 377–382, New York, NY, USA, 2001. ACM Press.

[SK07] M. SCHOLZ und R. KLINKENBERG: *Boosting Classifiers for Drifting Concepts*. Intelligent Data Analysis (IDA), Special Issue on Knowledge Discovery from Data Streams, (to appear), 2007.

[SRG86] J.C. SCHLIMMER und JR. R.H. GRANGER: *Incremental Learning from Noisy Data*. Mach. Learn., 1(3):317–354, 1986.

[Sta03] K.O. STANLEY: *Learning Concept Drift with a committee of decision trees*. Technischer Bericht AI-TR-03-302, Department of Computer Science, University of Texas at Austin, USA, 2003.

[Sul96] M. SULLIVAN: *Tribeca: A Stream Database Manager for Network Traffic Analysis.* In: *VLDB*, Seite 594, 1996.

[SW86] C. STANFIL und D. WALTZ: *Toward memory-based reasoning*. Communications of the ACM, 29:1213–1228, 1986.

[Thi95] H. THIELE: *Einführung in die Fuzzy-Logik (Skript)*. Universität Dortmund, Fachbereich Informatik, Lehrstuhl für Informatik I, 1995.

[Vit85] J.S. VITTER: *Random sampling with a reservoir*. ACM Trans. Math. Softw., 11(1):37–57, 1985.

[WF05] I.H. WITTEN und E. FRANK: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2. Auflage, 2005.

[WFYH03] H. WANG, W. FAN, P.S. YU und J. HAN: *Mining concept-drifting data streams using ensemble classifiers*. In: *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Seiten 226–235, New York, NY, USA, 2003. ACM Press.

[WK92] G. WIDMER und M. KUBAT: *Learning Flexible Concepts from Streams of Examples: FLORA 2*. In: *European Conference on Artificial Intelligence*, Seiten 463–467, 1992.

[WK93] G. WIDMER und M. KUBAT: *Effective Learning in Dynamic Environments by Explicit Context Tracking*. In: *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, Band 667, Seiten 227–243. Springer-Verlag, 1993.

[WK96] G. WIDMER und M. KUBAT: *Learning in the presence of concept drift and hidden contexts*. Machine Learning, 23(1):69–101, 1996.

[YG02] Y. YAO und J.E. GEHRKE: *The cougar approach to in-network query processing in sensor networks*. ACM SIGMOD Record, 31(2):9–18, September 2002.

[ZS02] Y. ZHU und D. SHASHA: *StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time*. In: *Analyzing Data Streams - VLDB 2002*, Seiten 358–369. Springer, 2002.