Classification of User Behavior through Connectivity-Based Clustering

Bachelorarbeit

Melf Lorenzen

6. April 2018

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Arbeitsgruppe Software Engineering

Betreut durch: Prof. Dr. Wilhelm Hasselbring
Dr. Reiner Jung
M.Sc. Marc Adolf

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 6. April 2018

Zusammenfassung

Die automatische Klassifizierung von Benutzerverhalten ist für eine Vielzahl von Anwendungen nützlich. Das Überwachungs und Kontrollprogramm iObserve verwendet sie, um die von den Benutzen verursachte Last einer beobachteten Cloud-basierten Anwendung zu simulieren. Die Klassifizierung wird durch eine Transformation des Benutzerverhaltens in Vektoren und ein anschließendes Clustering dieser Vektoren realisiert.

Bislang wurde das Clustering durch die XMeans und Expectation-Maximization Clustering-Verfahren umgesetzt. Inhalt dieser Arbeit ist die Auswahl und Implementierung eines hierarchischen Clustering-Algorithmus sowie einer Methode zum Bestimmen der Clustergröße.

In iObserve implementiert wurden der BIRCH-Clustering-Algorithmus sowie die L-Method, die aus einer Hierarchie von Clusterings ein Clustering auswählt. Anschließend wurden die Ergebnisse der Klassifizierung des BIRCH-Algorithmus mit denen des XMeans und Expectation-Maximization Clusterings verglichen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ob für Werbetreibende, Systemtester- oder Entwickler: Das Untersuchen des Benutzerverhaltens ist ein wichtiger Aspekt in der Analyse von Anwendungssoftware. Häufig ist man dabei an einer Klassifizierung des Benutzerverhaltens interessiert. Was zeichnet die typischen Benutzer der Anwendung aus? Je umfangreicher und vielfältiger das beobachtete Benutzerverhalten, desto weniger praktikabel ist eine manuelle Auswertung. Umso mehr ist man an Lösungen interessiert, die das Verhalten der Benutzer automatisch kategorisieren können. Dies wird zum Beispiel durch den Einsatz von Clustering-Algorithmen erreicht. An Clustering-Algorithmen wird schon seit Jahrzehnten geforscht. Es gibt inzwischen eine große Auswahl an unterschiedlichen Algorithmen. Bei der Klassifizierung von Benutzerverhalten werden jedoch häufig nur eine Handvoll unterschiedlicher Standardalgorithmen verwendet. Ganze Familien von Verfahren, wie etwa das hierarchische Clustering, sind bislang kaum zum Einsatz gekommen. Es stellt sich die Frage, ob die Anwendung solcher Clustering Verfahren die Klassifizierung insgesamt verbessern könnte.

Das Überwachungs- und Steuerungsprogramm für Cloud-basierte Anwendungen, iObserve [Hasselbring u. a. 2013], klassifiziert beobachtetes Benutzerverhalten, um den Zustand der jeweiligen Anwendung zu simulieren. Zum Clustering wird der XMeans Algorithmus nach [Pelleg und Moore 2000] verwendet. Bei der Evaluation dieses Ansatzes [Dornieden 2017] zeigte sich, dass ähnliche, aber unterschiedliche Benutzerprofile im Ergebnis vermischt worden waren. Außerdem erfordert dieser Ansatz eine Schätzung der Anzahl der zu erwartenden Benutzerprofile. Dies entspricht nicht dem Ziel einer automatischen Klassifizierung.

Ziel dieser Arbeit ist daher, ein alternatives Clustering Verfahren zu finden, zu implementieren und mit den vorhandenen Methoden zu vergleichen.

1.1. Ziele dieser Arbeit

Dieser Abschnitt unterteilt das Ziel in eine Reihe von konkreten Teilzielen.

1.1.1. Auswahl eines Clustering Algorithmus

Zunächst muss ein Algorithmus ausgewählt werden. Er sollte aus dem Bereich des hierarchischen Clusterings stammen. Diese Verfahren zählen zu den Gängigsten und werden in

der Praxis für eine Vielzahl von Problemen eingesetzt [Berkhin 2006], wenn auch bislang noch nicht für die Klassifizierung von Benutzerverhalten. iObserve hat eine Echtzeitüberwachung zum Ziel. Dementsprechend sollte das Verfahren schnell sein. Dies wird durch eine geringe Laufzeitkomplexität gewährleistet. *Clustering* Verfahren arbeiten auf Vektoren. Bei der Darstellung von Benutzerverhalten durch Vektoren erreichen diese häufig hohe Dimensionen. Folglich sollte der Algorithmus also auch bei hoher Dimensionalität funktionieren.

1.1.2. Auswahl eines Verfahrens zur Bestimmung der Clusteranzahl

Ein hierarchische Clusteringverfahren erstellt eine Hierarchie von Clusterings. Um eine selbständige Klassifizierung zu erreichen, wird eine Methode benötigt, die aus dieser Hierarchie ein Clustering auswählt. Auch hier ist ein schnelles Verfahren zu bevorzugen.

1.1.3. Implementierung in iObserve

Sind ein Algorithmus sowie eine Methode zu Bestimmung der Clusteranzahl gefunden worden, werden sie in iObserve implementiert. Da nur das Clustering untersucht wird, sollten die anderen Programmteile der Analyse weiterverwendet werden. Außerdem sollte es möglich sein, durch die Konfiguration von Observe zwischen den verschiedenen Verfahren zu wechseln.

1.1.4. Parameterstudie und Evaluation anhand des JPetStores

Um die Effektivität des Ansatzes zu beurteilen, werden die Ergebnisse der Klassifizierung mit denen der bereits vorhandenen Verfahren, XMeans und dem Expectation Maximization (EM) Clustering-Verfahren, verglichen. Als Testfall werden eine Reihe von aufgezeichneten Benutzerdaten aus dem JPetStore (siehe Abschnitt 2.5) verwendet. Zu beobachten ist hier, ob der neue Algorithmus die verschiedenen Benutzerprofile besser unterscheiden kann.

Bevor man die Algorithmen vergleichen kann, wird in einer Parameterstudie eine möglichst geeignete Parametrisierung des Algorithmus für die JPetStore Daten gesucht. Wenn möglichst viele Benutzerprofile in den Testdaten korrekt erkannt werden, ist eine Parametrisierung geeignet.

1.2. Aufbau der Arbeit

In Kapitel 2 werden die in dieser Arbeit verwendeten Technologien näher erläutert. Danach führt Kapitel 3 aus, welche Algorithmen und Methoden ausgewählt wurden und wie diese umgesetzt werden sollen. Kapitel 4 schildert die Implementierung des Ansatzes. Hier werden neue Datenstrukturen und Veränderungen an der Struktur von iObserve vorgestellt. Daran schließt sich die Parameterstudie in Kapitel 5 an, in der die Effektivität

des Clusterings mit verschiedenen Parametern getestet wird. Kapitel 6 vergleicht die Ergebnisse der Parameterstudie dann mit anderen Clustering-Verfahren in iObserve. In Kapitel 7 werden mit dieser verwandte Arbeiten vorgestellt und die Unterschiede diskutiert. Schließlich fasst Kapitel 8 die Ergebnisse dieser Arbeit zusammen und weist auf zukünftige Verbesserungen und sich anschließende Fragen hin.

Kapitel 2

Grundlagen und Technologien

Dieses Kapitel erläutert die verschiedenen Technologien und Grundlagen, auf denen diese Arbeit basiert. Zunächst wird iObserve erläutert. Daran schließen verschieden Technologien an, die in iObserve zum Einsatz kommen: Das TeeTime Framework, die Modellierung von Benutzerverhalten, das Palladio-Component-Modell sowie der JPetStore, der als Testfall fungiert.

Darauf folgen spezielle Technologien dieser Arbeit: Clustering, hierarchisches Clustering sowie der BIRCH-Algorithmus als Vertreter des hierarchischen Clusterings. Ebenfalls zu diesen Technologien gehört die L-Method, welche die Auswahl aus Clusterings verschiedener Größe erlaubt. Zuletzt wird noch die GQM-Methode vorgestellt, ein Verfahren zur Auswertung.

2.1. Das iObserve Projekt zur Überwachung von Cloud-basierten Anwendungen

iObserve dient der Überwachung von Anwendungen, die mittels fremder Dienste und Hardware ausgeführt werden (etwa in der Cloud). Grundsätzlich sind in so einer Situation weniger Informationen über den Anwendungszustand vorhanden. Dementsprechend lassen sich Ziele wie etwa Performance schlechter gewährleisten.

iObserve tritt dem entgegen, indem es die vorhandenen Daten zur Laufzeit zur Erzeugung von Modellen nutzt, die den Zustand der Anwendung widerspiegeln. Aufgrund dieser Modelle können dann automatische Maßnahmen getroffen werden, um die Zielvorgaben zu erreichen. Zudem können die Daten genutzt werden, um Veränderungen an der Software durch die Entwickler vorzunehmen [Hasselbring u. a. 2013].

Die Implementierung von iObserve beruht auf dem TeeTime Framework. Zur Modellierung nutzt iObserve das Palladio-Component-Model.

2.2. Pipe and Filter Framework TeeTime

TeeTime ist eine Umsetzung des Software Architektur Patterns „Pipe & Filter“. In diesem Pattern besteht ein Programm aus einer Reihe von Filtern, die durch Pipes miteinander verbunden sind. In den Filterkomponenten findet die Informationsbearbeitung statt. Ein

Filter erhält aus einer Pipe Daten, die dann transformiert oder angereichert werden. Hat ein Filter die Daten verarbeitet, werden diese am Ausgang des Filters bereitgestellt und über eine Pipe zum nächsten Filter geleitet. Die Pipes verbinden die Filter und synchronisieren ihre Zusammenarbeit. Fragt ein Filter etwa am Eingang nach neuen Daten an, wird diese Anfrage von der Pipe beantwortet und nicht von dem vorangehenden Filter. In der einfachsten Form des Patterns hat jeder Filter genau einen Eingang und Ausgang. Ausgenommen davon sind der erste Filter, die sogenannte Quelle, sowie der letzte Filter, die Senke. In einem solchen Programm fließen die Daten stets nur in eine Richtung. Es gibt aber auch komplexere Varianten, in denen mehrere Ein- und Ausgänge erlaubt sind. Dann sind auch Verzweigungen und Rückkoppelungen in der Struktur möglich [Schmidt u. a. 1996].

Solch komplexere Varianten erlaubt auch TeeTime. Im Zusammenhang mit TeeTime wird von Stages statt Filter gesprochen. Als Filter werden alle Stages bezeichnet, die keine Senke oder Quelle sind. Die Stages in dem Framework verfügen je nach Typ über eine Anzahl von Ein- und Ausgangsports. Durch das Verbinden des Eingangsports einer Stage mit dem Ausgangsport einer anderen Stage werden sie von einer Pipe verbunden. Die in iObserve am häufigsten anzutreffenden TeeTime Stages sind von den Klassen AbstractConsumerStage, AbstractStage und CompositeStage abgeleitet. Eine AbstractConsumerStage hat nur einen Eingangsport und einen Ausgangsport. Sie verarbeitet eintreffende Daten und schickt das Ergebnis dann weiter. Die AbstractStage wird benötigt, wenn mehrere Ein- und Ausgangsport für den Filter nötig sind. Eine CompositeStage setzt sich aus mehreren Stages zusammen. Sie hat bei der Verarbeitung der Daten jedoch keine Bedeutung. Ihre Inputports verweisen auf die der enthaltenen Stages [Wulf u. a. 2017].

2.3. Palladio Component Model zur Architekturbeschreibung

Das Palladio Component Model (PCM) [Becker u. a. 2009] dient der Beschreibung einer Komponenten basierten Architektur. Insbesondere soll die Beschreibung alle Performance relevanten Faktoren miteinbeziehen, um so Quality of Service Aspekte wie Performance oder Zuverlässigkeit der Architektur voraussagen zu können.

Das PCM setzt sich aus vier Teilmodellen zusammen, die jeweils einen anderen Aspekt modellieren:

Component specification Beschreibt die einzelnen Komponenten. So wird etwa für jede Komponente spezifiziert, welche Dienste sie in Form von Interfaces anbietet und welche sie benötigt.

Architecture model In diesem wird modelliert, wie die Komponenten in Form des Systems zusammenarbeiten.

Das Resource model Dieses beschreibt die vorhandenen Ressourcen und wie die Komponenten des Systems diesen Ressourcen zugeteilt werden.

Usage Model In die Last modelliert wird, die das System zu erwarten hat. Dies geschieht über den *Workload* und das Benutzerverhalten.

2.4. Modellierung von Benutzerverhalten

Das Verhalten eines Benutzers lässt sich anhand seiner Entry-Calls, also der Aufrufe, die beim System eingehen, verfolgen. Zur Darstellung des Verhaltens definieren Menasce und Almeida [2000] den Customer-Behavior-Model-Graph (CBMG). Ein CBMG ist ein Zustandsübergangsgraph, bei dem die Systemaufrufe als Zustände modelliert werden. Eine Kante in dem Graph modelliert den Übergang von einem Aufruf zum anderen. Die Kanten werden mit den gemessenen Anzahlen oder Verhältnissen der Übergänge beschriftet. Ein solches Diagramm macht anschaulich, wie die Benutzer mit dem System interagieren.

Diesen Ansatz verwendet auch in iObserve. Hier wird Anzahl der Übergänge an den Kanten vermerkt. Knoten haben außerdem eine weitere Eigenschaft, die sogenannte Call-Information. Sie speichert zusätzliche Information zu dem Aufruf, den dieser Benutzer ausgelöst hat. Wurde etwa der Entry-Call *Catalog.viewCategory=&categoryId=REPTILE* registriert, wird dem Knoten *Catalog.viewCategory* des Call-Graphen die Call-Information REPTILE hinzugefügt. Abbildung 2.1 zeigt einen Call-Graphen mit Call-Information für einen Benutzer des JPetStore. In dem Beispiel wurde *Catalog.viewCategory* etwa vier mal mit der Call-Information BIRDS aufgerufen, einmal mit der Call-Information REPTILES.

Eng mit der CBMG verwandt ist die Markow-Kette [Mark und Csaba 2007]. Anders als beim CBMG werden die Kanten mit Übergangswahrscheinlichkeiten versehen. Auf Basis von Markow-Ketten lassen sich weitere Berechnungen anstellen. Mark und Csaba [2007] berechnen etwa die durchschnittliche Anzahl täglicher Besucher auf Basis stabiler Markow-Ketten.

2.5. JPetStore

Der JPetStore[1] ist eine frei verfügbare Beispielanwendung für einen Webserver. Er hostet eine Webseite, in der eine Reihe von Haustieren in unterschiedlichen Kategorieren angeboten werden, zum Beispiel Hunde, Katzen und Fische. Benutzer der Seite können im Katalog stöbern und Bestellungen aufgeben. Eine Suchfunktion und eine Benutzerregistrierung komplettieren das Funktionsangebot.

Dornieden [2017] hat für die Evaluation seiner Arbeit Monitoring-Daten aus dem JPetStore generiert. Zu diesem Zweck wurden sieben Benutzerprofile erstellt: *Account Manager*, *Browsing User*, *Cat Lover*, *Fish Lover*, *New Customer*, *Single Cat* und *Single Reptile*.

[1] https://github.com/mybatis/jpetstore-6

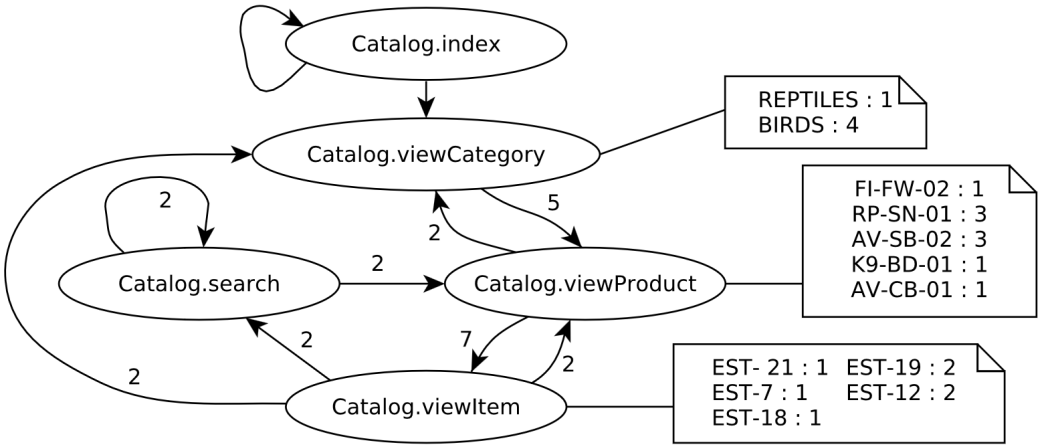

Abbildung 2.1. Beispiel eines Verhaltensmodells in iObserve für den JPetStore (Grafik aus [Jung u. a. 2017].

Diese sind so konzipiert, dass eine Klassifizierung sowohl Benutzerprofile unterscheiden können muss, die sich im Call-Graph, in der Call-Information oder in beiden Aspekten unterscheiden. Die Testdaten enthalten die Aktionen der Benutzerprofile im JPetStore. Jedes Benutzerprofil ist mit 20 Sessions in den Daten vertreten.

2.6. Clustering zur Informationsgewinnung

Clustering beschreibt die Aufteilung einer Menge von Objekten in Gruppen (Cluster). Objekte werden in einer Gruppe zusammengefasst, wenn sie sich untereinander ähnlich sind, aber unähnlich zu den Elementen der anderen Cluster. Abzugrenzen ist der Begriff von dem der Klassifikation. Hier steht vor Beginn bereits fest, welche Gruppen es gibt, aber nicht die Zuordnung der Objekte in diese.

Durch das Clustering soll die zugrundeliegende Struktur in der Menge der Objekte sichtbar gemacht werden. Außerdem wird es ermöglicht Objekte zusammenzufassen: Die Elemente eines Clusters lassen sich durch einen typischen Vertreter des Clusters repräsentieren. Eine Vielzahl unterschiedlicher Fachgebiete, wie etwa der Bilderkennung oder die Erforschung des Genoms, nutzt Clustering zu diesen Zwecken [Jain 2010].

2.7. Verschiedene Clustering Verfahren

Es werden zwei Arten unterschieden: Partitionierende und hierarchische Verfahren. Partitionierende Verfahren suchen eine Aufteilung der Objekte in eine bestimmte Anzahl von

Clustern. Dabei muss jedes Objekt zu einem Cluster gehören und diese Cluster müssen disjunkt sein. Bei hierarchischen Verfahren wird eine Folge von Partitionierungen erzeugt. Die Cluster einer Partitionierung entstehen dabei aus den Clustern der jeweils vorhergegangenen Partitionierung. Dies geschieht entweder durch Aufteilung oder Zusammenfassen der Cluster. Beim hierarchischen Clustering werden zwei Vorgehensweisen unterschieden: Das divise sowie das agglomerative Clustering. Beim divisen Clustering wird mit einem alle Objekte umfassenden Cluster angefangen, der dann in zwei Cluster aufgeteilt wird. In den nachfolgenden Schritten werden diese wieder aufgeteilt. Dieser Prozess kann mehrfach wiederholt werden. Aufgrund der Vielzahl von Möglichkeiten, einen Cluster zu zerteilen, messen Xu und Wunsch [2005] dieser Art der Clusterings eher geringe praktische Bedeutung zu. Das öfter umgesetzte Clustering ist daher das agglomerative Clustering. Diese Algorithmen beginnen mit einer Anzahl Clustern und verschmelzen diese dann in jedem Schritt mit ihrem nächsten Nachbarn. Wie man diesen nächsten Nachbarn bestimmt, hängt vom jeweiligen Algorithmus ab.

Bei beiden Arten des hierarchischen Clusterings entsteht ein Baum von Clusterings. Häufig wird dieser auch als Dendrogramm dargestellt, um die Beziehung zwischen den Clustern in den einzelnen Stufen zu verdeutlichen. Ein konkretes Clustering findet man, indem man sich für eine Stufe des Baums entscheidet. Neben der Qualität des Clusterings unterscheiden sich die Algorithmen noch in der Laufzeit. Diese hängt einerseits von der Anzahl der Objekte ab, aber auch von der Anzahl der Objektattribute, die der Algorithmus verwendet. Diese wird auch als Dimension bezeichnet und kann wesentlichen Einfluss auf die Laufzeit haben [Xu und Wunsch 2005].

2.8. Die Funktionsweise des BIRCH-Algorithmus

Der BIRCH-Algorithmus (nach [Zhang u. a. 1997]) ist ein hierarchischer Clustering Algorithmus. Er verbindet eine Reihe verschiedener Techniken, um typische Probleme hierarchischer Clustering Algorithmen zu vermeiden. Eine zentrale Idee ist die Reduzierung der Punktmenge auf eine leichter zu beherrschende Menge von Clustern. Dadurch erreicht er eine Laufzeit, die linear von der Anzahl der Vektoren abhängt. Abbildung 2.2 zeigt den BIRCH-Algorithmus im Überblick. Er besteht aus vier Phasen:

1. Die Vektoren in einer Baumstruktur zusammengefasst und angeordnet. Dieser Baum heißt Clustering-Feature Tree (CF-Tree).

2. Der Baum wird mit anderen Parametern neu aufgebaut und so verkleinert. Dies kann zum einen notwendig sein um Speicherplatz zu sparen. Zum anderen weil einer der folgenden Schritte mit einer kleineren Anzahl von Objekten besser funktioniert.

3. Ein für die Baumstruktur angepasstes Clustering Verfahren wird angewandt. Von den Autoren des BIRCH-Algorithmus ist hier ein agglomeratives Verfahren vorgesehen. Einsetzen lässt sich aber jedes Verfahren, das die Baumstruktur nutzen kann.

4. Zum Schluss werden die ursprünglichen Vektoren den im letzten Schritt gefundenen Clustermittelpunkten zugeordnet und die Ergebnisse so verfeinert.

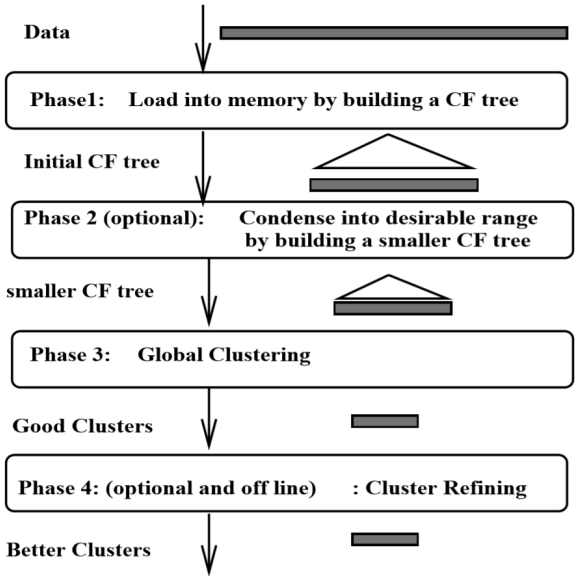

Abbildung 2.2. Die Phasen des BIRCH-Algorithmus (Grafik aus [Zhang u. a. 1997])

Im Folgenden werden die einzelnen Schritte genauer erläutert. Zunächst soll der Aufbau der Baumstruktur in der ersten Phase des Algorithmus vorgestellt werden.

2.8.1. Aufbau des Clustering-Feature-Baums

Von wesentlicher Bedeutung ist der Begriff des Clustering-Feature (CF) beziehungsweise Clustering-Features (CFs). Ein Clustering-Feature beschreibt die wesentlichen Charakteristika eines Clusters, ohne sich alle darin enthaltenen Vektoren merken zu müssen. Auf Basis des Clustering-Features lassen sich verschiedene Metriken berechnen, wie etwa sein Durchmesser oder der Clustermittelpunkt. Will man zwei Cluster vergleichen, lassen sich anhand der CFs eine Reihe von Abstandsmetriken berechnen. Für eine Menge N d-dimensionaler Vektoren $X = \{x_1, \cdots, x_N\}$ ist das Clustering-Feature dieser Vektoren definiert als

$$CF(X) = (N, LS, SS), \text{ wobei } LS = \sum_{i=1}^{N} x_i \text{ und } SS = \sum_{i=1}^{N} x_i^2.$$

Ein CF repräsentiert einen Cluster von Vektoren. Unabhängig von der Anzahl der Vektoren werden stets nur zwei Vektoren und eine Zahl benötigt, um den Cluster zu beschreiben.

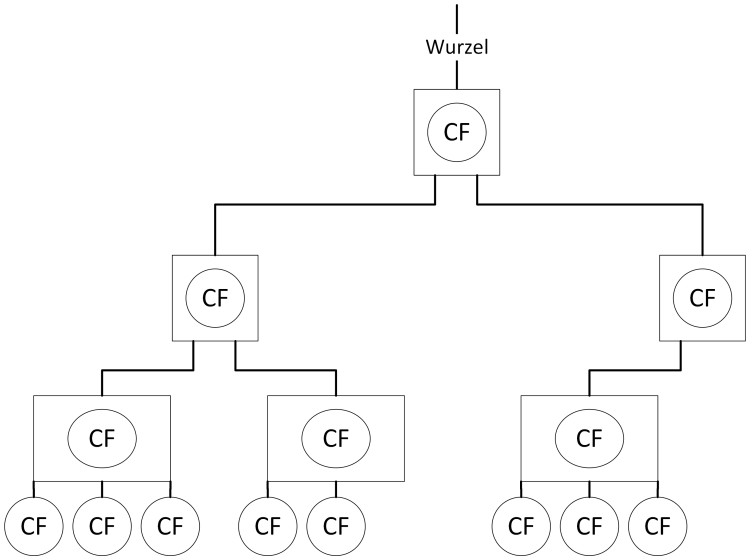

Abbildung 2.3. Beispiel eines CF-Trees

Da alle Bestandteile eines CFs additiv gebildet werden, gilt auch

$$CF(\{x_1, x_2\}) = CF(\{x_1\}) + CF(\{x_2\}.$$

Daraus folgt: Die Summe von zwei CFs entspricht gerade dem CF der vereinigten Cluster.

Im BIRCH-Algorithmus wird fast ausschließlich mit den CFs gearbeitet, die einzelnen zu clusternden Vektoren werden nur initial und in der Verfeinerung im letzten Schritt benötigt. In der ersten Phase werden die Vektoren in einen Suchbaum aus CFs überführt. Der CF-Baum besteht aus Knoten und Blättern. Abbildung 2.3 zeigt ein Beispiel eines solchen Baums mit 3 Knoten, die jeweils bis zu zwei Kinder aufnehmen können, sowie Blättern mit bis zu drei Einträgen. Die CFs befinden sich in den Blättern des Baums. Ein Blatt enthält ein oder mehrere Einträge von CFs sowie ein CF, das sich aus der Summe der Einträge zusammensetzt. Die Anzahl der CFs, die ein Blatt aufnehmen kann, ist konfigurierbar. Die Autoren schlagen vor diese Grenze so zu wählen, dass ein Blatt gerade in eine Speicherseite des Hauptspeichers des Systems passt. Ein Knoten besteht aus einer Reihe von Kindern sowie einem CF, dass sich – analog zum Blatt – aus der Summe der CFs der Kinder berechnet. Die Kinder sind entweder alle Blätter oder selbst wieder Knoten. Für optimale Leistung sollte die Anzahl der Kinder auch hier so gewählt sein, dass ein Knoten gerade auf die Größe einer Speicherseite kommt. Der wesentliche Arbeitsschritt beim Aufbau des Baums ist das Einfügen der Vektoren in den CF-Tree. Aus einem Vektor lässt sich ein CF berechnen, welches dann in den CF-Baum eingefügt wird. Zu Anfang besteht der Baum nur aus einem Blatt an der Wurzel. In Abbildung 2.4 wird das Einfügen in den Baum gezeigt. Beginnend an der Wurzel wird auf jeder Ebene des Baum das einzufügende

CF mit den CFs der Kinder verglichen. BIRCH bietet hier mehrere Metriken, um den Abstand zwischen zwei Clustern zu bestimmen. Ein Beispiel ist der durchschnittliche Abstand der Elemente eines Clusters $D2$,

$$D2(\{X_1, \cdots, X_N\}, \{Y_1, \cdots, Y_M\}) = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{M} (X_i - Y_j)^2}{NM} \right)^{\frac{1}{2}}.$$

Hat man den Kindknoten gefunden, der dem einzufügenden CF am nächsten ist, wird das Einfügen rekursiv auf den Kindknoten angewendet. Dieser Prozess endet bei einem Blatteintrag. Die Blatteinträge sind selbst CFs. Der Algorithmus kann jetzt entweder das einzufügende CF mit dem gefundenen CF verschmelzen oder das einzufügende CF als weiteren Eintrag an das Blatt hängen. Verschmolzen werden die CFs, wenn das CF, welches durch die Verschmelzung entstehen würde, die Schwellwert Bedingung erfüllt: Dazu muss Radius oder der Durchmesser des Cfs kleiner sein als der Wert t. Ob der Algorithmus den Radius oder den Durchmesser verwendet, kann konfiguriert werden. Schwellwert t wird dieser Phase als Parameter übergeben. Durchmesser und Radius eines Clusters $X = \{X_1, \cdots, X_N\}$ sind definiert als:

$$D(X) = \left(\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (X_i - X_j)^2}{N(N-1)} \right)^{\frac{1}{2}}.$$

$$R(X) = \left(\frac{\sum_{i=1}^{N} (X_i - X0)^2}{N} \right)^{\frac{1}{2}} \quad (X0 = \text{Clustermittelpunkt von } X).$$

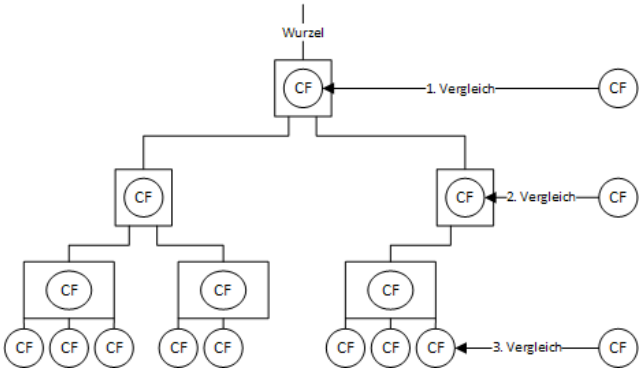

Abbildung 2.4. Einfügen in den CF-Tree

Konnten die CFs verschmolzen werden, ist der Vorgang beendet. Musste ein neuer Eintrag angelegt werden, kann es zu der Situation in Abbildung 2.5 kommen. Das rechte untere Blatt hat vier Einträge, obwohl nur drei pro Blatt erlaubt sind. In diesem Fall muss

das Blatt aufgeteilt werden.

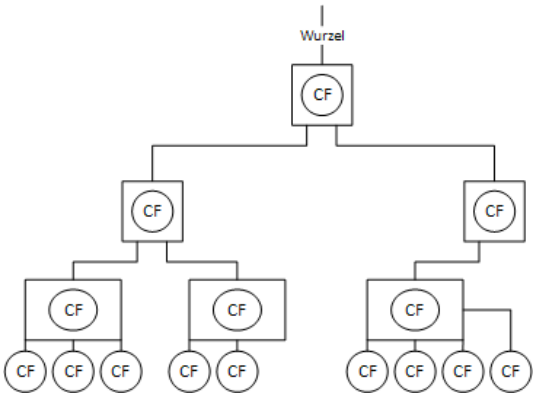

Abbildung 2.5. Blatt mit zu vielen Einträgen

Bei einer Aufteilung wird zunächst berechnet, welche der beiden CFs in den Einträgen am weitesten von einander entfernt sind. Dann werden die übrigen Einträge zwischen diesen beiden CFs aufgeteilt. Diese Aufteilung erfolgt danach, welchem CF der jeweilige Eintrag näher ist. Das eine CF bleibt mit seinen Einträgen in dem Blatt, das andere migriert in ein neues Blatt. Das neue Blatt wird dann dem Knoten darüber hinzugefügt (siehe Abbildung 2.6.

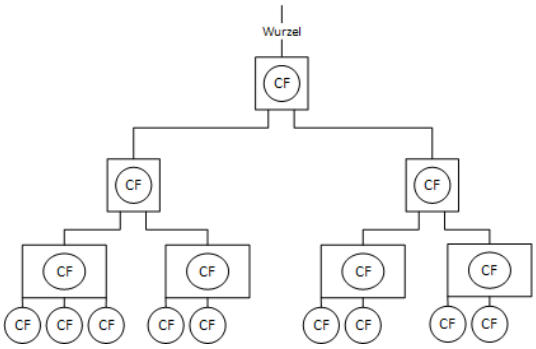

Abbildung 2.6. Nach Aufteilen des Blatts

Schlussendlich gibt es noch den Refinement-Merge. Erzeugt das Einfügen ein neues Blatt, wird dieses im Knoten darüber eingefügt. Möglicherweise muss dann auch dieser Knoten aufgespalten werden. Letztlich endet dieser Teilungsprozess auf einer Ebene, wenn ein Knoten noch Platz für ein weiteres Kind hat. Abbildung 2.7 zeigt diese Situation. Die Knoten können hier maximal drei Kinder aufnehmen.

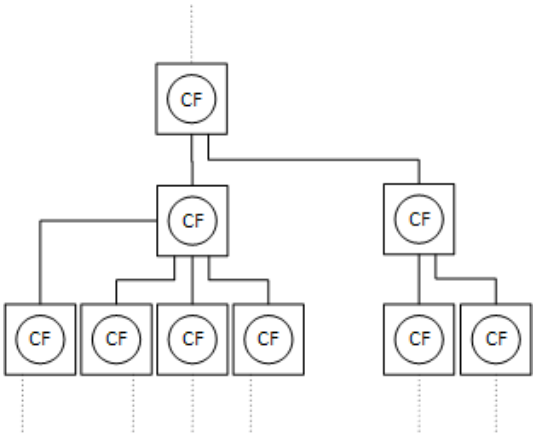

Abbildung 2.7. Ausgangssituation für Refinment-Merge

Der linke Kindknoten wird aufgespalten, wodurch ein neuer Knoten entsteht. Dieser findet dann Platz in der Ebene darüber (Abbildung 2.8).

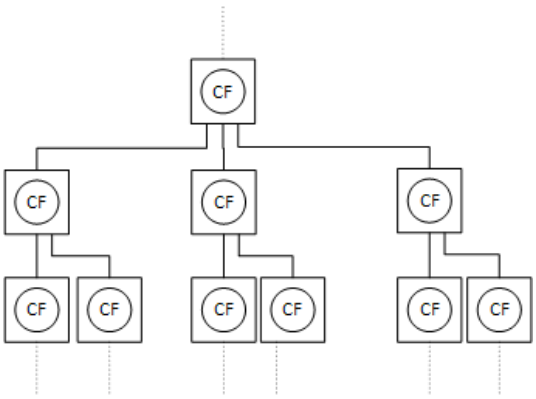

Abbildung 2.8. Nach Aufteilen des Knotens

Der Baum ist zwar korrekt. Jedoch müssen die beiden Kinder, die sich am nächsten sind, noch einmal miteinander verschmolzen werden, um gewissen Anomalien vorzubeugen. Dies ist allerdings nur nötig, wenn es sich bei diesen beiden Kindern nicht schon um die gerade gespaltenen Knoten handelt, weil diese erst durch das Verschmelzen entstanden sind. In Abbildung 2.9 ist ein mögliches Ergebnis des Refinement-Merge zu sehen: Der mittlere und der rechte Kindknoten waren sich am nächsten. Nach dem Merge ist eines der Kinder zum rechten Knoten gewandert.

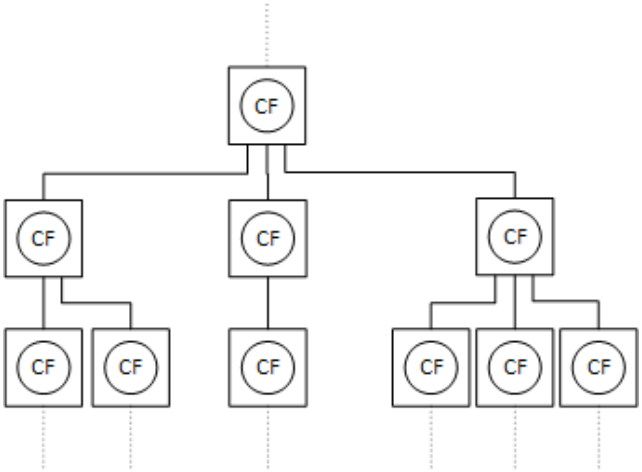

Abbildung 2.9. Nach Refinement-Merge

2.8.2. Verkleinern des Baums

Sollte der CF-Tree zu viel Speicherplatz in Anspruch nehmen oder die Anzahl der Cluster zu groß sein für das anschließende Clustering, kann der Baum in dieser Phase verkleinert werden. Die Anzahl der Cluster wird dadurch verringert, dass der maximale Durchmesser für ein CF im Blatt vergrößert wird. Der neue Schwellenwert wird so berechnet: Für jedes Blatt wird der Abstand der am nähesten beieinanderliegenden CFs berechnet, dann wird der Durchschnitt über diese Werte gebildet und als Schwellenwert für den neuen Baum benutzt.

Abbildung 2.10 zeigt das Vorgehen beim Aufbau des neuen Baums. Die Pfeile entsprechen einem Pfad durch den Baum, also einer Reihe von aufeinander folgenden Knoten von der Wurzel bis zu einem Blatt. Die Blätter des alten Baums werden der Reihe nach in den neuen Baum übertragen. Dabei werden im neuen Baum alle Knoten, die im alten Baum auf dem Pfad zum Blatt gelegen haben, in den neuen Baum eingefügt. Dies geschieht allerdings nur, falls diese Knoten nicht bereits angelegt wurden. Dieser Pfad aus Knoten ist in Abbildung 2.10 als NewCurrentPath zu sehen. Dann werden die CFs des Blattes einzeln übertragen. Bei jedem CF wird geprüft, ob es eine Möglichkeit gibt, sie vor dem NewCurrentPath einzufügen, ohne einen Split zu verursachen. Wenn dies möglich ist, werden sie an dieser Stelle eingefügt (der NewClosestPath in Abbildung 2.10). Ansonsten werden sie in den NewCurrentPath verschoben.

Als zusätzliche Option bietet der Algorithmus beim Neuaufbau die Chance Außreißer zu behandeln. Blättereinträge, die aus sehr viel weniger Punkten bestehen als der durchschnittliche Eintrag, werden beim Neuaufbau nicht in den neuen Baum eingefügt, sondern weggeschrieben. Wenn alle Vektoren verarbeitet sind, wird noch einmal geprüft, ob die Ausreißer jetzt vom fertigen Baum aufgenommen werden können. Falls ja, werden sie mit

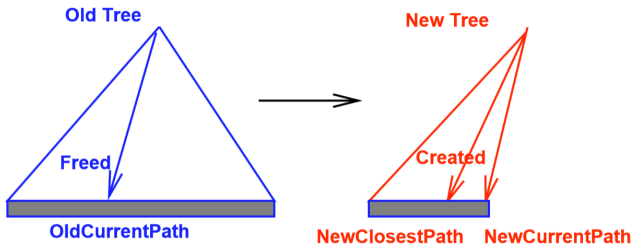

Abbildung 2.10. Vorgehen beim Verkleinern des Baums (Grafik aus [Zhang u. a. 1997])

dem entsprechenden Eintrag verschmolzen. Ist dies nicht möglich, werden sie entfernt.

2.8.3. Clustering und Verfeinerung

In der dritten Phase wird auf dem CF-Tree ein hierarchisches Clustering Verfahren durchgeführt. Die BIRCH Autoren schlagen den folgenden Algorithmus aus Algorithmus 15 vor:

Algorithm 1: Nearest Neighbour Algorithmus nach [Olson 1995]

Data: n+1 Cluster
Result: Folge von Clusterings

```
1  c_0 = ∅ ;
2  c_1 = Random(n);
3  i = 1;
4  k = 1;
5  while k < n do
6      repeat
7          i = i + 1;
8          c_i = Nearest-Neighbour(c_{i-1});
9      until c_i == c_{i-2};
10     Agglomeriere c_i und N(c_i);
11     if i > 3 then
12         i = i - 3 ;
13     else
14         i = 1;
15     k = k + 1;
```

Im letzten Schritt werden die ursprünglichen Vektoren gegen die Clustermittelpunkt geclustert. Dabei muss für jeden Vektor geprüft werden, welchem Clustermittelpunkt er am nächsten ist. Dieser Schritt behebt mögliche Fehlallokationen. Insbesondere wird dadurch

sichergestellt, dass identische Punkte auch zum selben Cluster gehören. Das Resultat dieser Phase ist das fertige Clustering.

2.9. L-Method zur Bestimmung der Clusteranzahl

Wie in Abschnitt 2.7 beschrieben, erzeugen hierarchische Verfahren nicht nur ein Clustering, sondern eine ganze Hierarchie von Clusterings. Die L-Method [Salvador und Chan 2004] ist ein Verfahren zur Bestimmung der richtigen Anzahl von Clustern in einer Menge von Clusterings. Sie benötigt zur Bestimmung nur die zu vergleichenden Clusterings und ein Evaluationsmaß. Der Algorithmus ist insbesondere da sinnvoll, wo bereits beim Clustering ein solches Evaluationsmaß berechnet wird.

Die L-Method geht wie folgt vor: in einem Graph wird die Anzahl der Cluster in einem Clustering dessen Wert in dem Evaluationsmaß gegenübergestellt. Dann wird für jede Anzahl von Clustern $c > 2$ folgender Wert berechnet:

$$RMSE_c = \frac{c-1}{b-1} RMSE(L_c) + \frac{b-c}{b-1} RMSE(R_c) \text{ mit } b = \text{Anzahl der Cluster.}$$

$RMSE$ ist die Wurzel der Fehlerquadrate der linearen Regression. Sie wird einmal angewendet auf die Punkte links im Graph von c und einmal auf die Punkte rechts von c. In Abbildung 2.11 sieht man, wie die gestrichelte Linie alle Punkte links von c approximiert, die flache durchgezogene Linie die Punkte rechts von c. Anschaulich wird so versucht das Knie des Evaluationsgraphen zu finden. Rechnerisch entspricht das Knie gerade dem Wert von c, an dem $RMSE_c$ sein Minimum annimmt. Dieses ist dann die optimale Anzahl von Clustern.

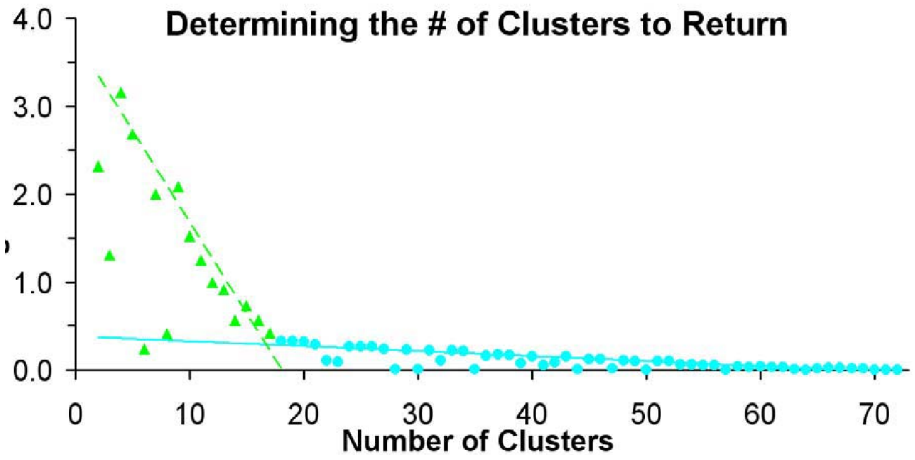

Abbildung 2.11. Finden des Knies im Evaluationsgraph (Grafik aus [Salvador und Chan 2004])

Diese Methode ist allerdings anfällig, wenn der Evaluationsgraph sehr schnell abfällt. Um dieses Problem zu beheben, schlagen die Autoren eine Iteration des Verfahrens vor, bei der der Ausläufer des Graphs nach und nach abgeschnitten wird. Folgender Algorithmus beschreibt das Verfahren:

Algorithm 2: Iterierte L-Method nach [Salvador und Chan 2004]

```
   Data: EvalGraph (a full evaluation graph)
   Result: suggested number of clusters
1  cutoff = ;
2  lastKnee = ;
3  currentKnee = EvalGraph.size();
4  repeat
5  |  lastKnee = currentKnee;
6  |  currentKnee = LMethod(evalGraph, cutoff);
7  until currentKnee ⩾ lastKnee;
8  RETURN currentKnee;
```

2.10. Goal/Question/Metric Ansatz zur Evaluation

Das Goal/Question/Metric Paradigma (GQM) nach [Basili 1992] ist ein Schema zur Analyse von Prozessen in der Softwareentwicklung sowie von Softwareprodukten. Durch Verwendung eines solchen Schemas soll die Analyse systematisch und nachvollziehbar erfolgen.

GQM folgt einem Top-Down Ansatz. Zuerst werden die Ziele festgelegt, die durch die Analyse verfolgt werden sollen. Diese werden durch den **Zweck**, die **Perspektive** sowie die **Umgebung** der Analyse näher charakterisiert.

Auf Basis eines Ziels werden dann Fragen erstellt, die dieses Ziel näher beschreiben. Die Fragen sollten *quantifizierbar* sein, damit sie mit Hilfe der Metriken beantwortet werden können. Die Metriken sind die Messwerte, die zur Beantwortung der Fragen nötig sind. Nachdem festgelegt wurde, um welche Metriken es sich handelt, muss die Erhebung der notwendigen Daten beschrieben werden.

Auf Basis dieses Schemas können dann die Metriken erhoben werden. Sie beantworten die Fragen, welche wiederum Auskunft darüber geben, inwieweit das Ziel erreicht wurde [Basili 1992].

Kapitel 3

Ansatz

Im folgenden Kapitel werden die Auswahl eines Clustering Algorithmus durchgeführt. Anschließend wird das Vorgehen bei der Implementierung umrissen. Danach wird eine Methode zur Bestimmung der Anzahl von Clustern bestimmt. Im letzten Abschnitt wird diskutiert, welche Features des BIRCH-Algorithmus für die Implementierung notwendig sind.

3.1. Auswahl eines Algorithmus für hierarchisches Clustering

Ziel war es aus dem Feld der hierarchischen Clustering Algorithmen einen Kandidaten zu finden, der folgende Kriterien erfüllt:

Geschwindigkeit Das Benutzerverhalten sollte in iObserve möglichst in Echtzeit beobachtet werden können. Eine geringe Laufzeit des Algorithmus ist daher von hoher Bedeutung.

Verträglichkeit mit hoher Dimensionalität Die Modellierung des Benutzerverhaltens durch Call-Graphen erzeugt Vektoren von hoher Dimension. Ist das beobachtete Programm sehr umfangreich, sind Vektoren mit Dimension im vierstelligen Bereich möglich. Für derartige Bereiche gibt es spezielle Ansätze, die nicht in den Bereich des hierarchischen Clustering fallen [Berkhin 2006]. Kandidaten sollten trotzdem möglichst robust im Umgang mit hoher Dimensionalität sein.

Während für fast alle Algorithmen die Laufzeit festgestellt werden kann, ist die Information zum Umgang mit hoher Dimensionalität nur vereinzelt zu finden. Dies mag auch daran liegen, das eigentlich alle hierarchischen Clustering Algorithmen auf Maße für die Abstände oder die Dichte an Objekten aufbauen. Laut [Beyer u. a. 1999a] nimmt die Aussagekraft solcher Maße mit steigender Dimension immer weiter ab. Es wird also keinen Spezialisten für sehr hohe Dimensionen unter den Algorithmen für hierarchisches Clustering geben. Daher wird wie folgt vorgegangen: Es werden die Algorithmen mit der geringsten Laufzeit in die nähere Auswahl genommen. Der bessere Umgang mit hoher Dimensionalität gibt dann den Ausschlag.

Tabelle 3.1 beinhaltet alle Algorithmen, die während der Recherche gefunden wurden. Unter *Klassisches HC* sind die Algorithmen zusammengefasst, die agglomerativ mit Hilfe einer Abstandsmetrik für Cluster eine Folge von Clusterings aufbauen (zum Beispiel Single-Link oder Ward's Method). DIANA und MONA sind divise Algorithmen. Die Autoren geben die Laufzeit nicht explizit an. Aus dem Algorithmus wird jedoch deutlich, dass diese mindestens quadratisch sein muss. ROCK und CURE bieten die Möglichkeit, den Algorithmus nur auf einem Sample auszuführen (daher N_S). Die Laufzeit ist zwar quadratisch, aber durch Beschränkung der Größe des Samples beherrschbar.

Tabelle 3.1. Liste der in Betracht gezogenen Algorithmen

Algorithmus	Laufzeit	Quelle
Klassisches HC	$> \mathcal{O}(N^2)$	[Murtagh und Contreras 2012]
DIANA	$> \mathcal{O}(N^2)$ (geschätzt)	[Kaufman und Rousseeuw 1990]
MONA	$> \mathcal{O}(N^2)$ (geschätzt)	[Kaufman und Rousseeuw 1990]
BIRCH	$\mathcal{O}(N)$	[Zhang u. a. 1997]
CURE	$\mathcal{O}(N_S^2 \log N_S)$	[Guha u. a. 1998]
ROCK	$\mathcal{O}(N_S^2 \log N_S)$	[Guha u. a. 2000]
Chameleon	$\mathcal{O}(N \log N)$	[Karypis u. a. 1999]
RelativeHC	$\mathcal{O}(N^3)$	[Mollineda und Vidal 2000]
SBAC	$\mathcal{O}(N^2)$	[Li und Biswas 2002]
AMOEBA	$\mathcal{O}(N \log N)$	[Estivill-Castro und Lee 2000]
HyperGraphBased	$> \mathcal{O}(N \log N)$	[Cherng und Lo 2001]
Fuzzy HC	$\mathcal{O}(N^2)$	[Geva 1999]
Support Vector C	$\mathcal{O}(N^2)$	[Ben-Hur u. a. 2001]
Baire-Metric	$\mathcal{O}(N)$	[Contreras und Murtagh 2012]

Da einer geringen Laufzeit die höchste Priorität eingeräumt wird, kommen BIRCH und Baire-Metric in die nähere Auswahl. Ein Sampling, wie es in ROCK und CURE durchgeführt wird, ist für sehr große Datenmengen gedacht. Ob diese in iObserve vorkommen, hängt stark vom jeweiligen Einsatz ab. Eine von vornherein geringe Laufzeit ist vorzuziehen: Prinzipiell lässt sich ein Sampling mit jedem Algorithmus verbinden.

Die Autoren des Baire-Metric Algorithmus machen keine Angaben zum Verhalten ihres Algorithmus bei hoher Dimension. Es muss daher von keiner besonderen Fähigkeit ausgegangen werden. Der BIRCH-Algorithmus basiert zwar ebenfalls auf Abstandsmaßen, die bei sehr hoher Dimensionalität allgemein an Aussagekraft verlieren. Die Autoren konnten jedoch mit BIRCH für 50-dimensionale Objekte noch erfolgreich Cluster berechnen. Diese Information gibt den Ausschlag für BIRCH.

In den folgenden Abschnitten wird das Vorgehen beim Umsetzen des Algorithmus beschrieben. Folgende Fragen sollten beantwortet werden:

1. Wie lässt sich der neue Algorithmus in iObserve einfügen?

2. Wie geht man mit den Ergebnissen des hierarchischen Clusterings um?
3. Der BIRCH-Algorithmus hat eine Reihe von Parametern und optionalen Funktionen. Welche sollten implementiert werden?

3.2. Nutzung der Schnittstellen in iObserve

Die Klassifizierung des Benutzerverhaltens lässt sich grob in drei Schritte unterteilen:

1. Verarbeitung der Monitoring Daten und Transformation zu Vektoren
2. Erstellen eines Clusterings auf den Vektoren
3. Transformation der geclusterten Vektoren in Modelle von Benutzerverhalten

Christoph Dornieden hat in seiner Arbeit [Dornieden 2017] bereits alle drei Schritte umgesetzt. Erstrebenswert wäre es, diese Implementierung um ein zusätzliches Verfahren für den Clustering Schritt zu erweitern, ohne dabei die Schnittstelle ändern zu müssen. Abbildung 3.1 illustriert diesen Ansatz.

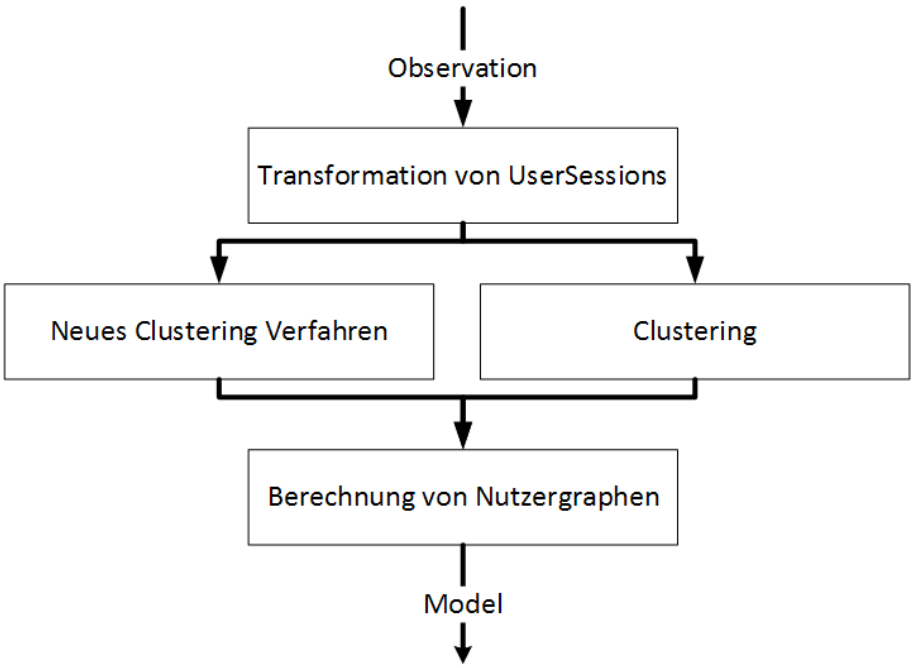

Abbildung 3.1. Geplantes Vorgehen bei der Implementierung in iObserve

Das bisherige Vorgehen ist eng mit der *weka.core.Instances* Klasse aus dem Weka-Paket zum Data Mining verbunden. Das gesamte Clustering wird auf dieser Datenstruktur

durchgeführt. Das Clustering wird auf Instances durchgeführt. Anschließend werden aus den Instances der Clustermittelpunkte die Verhaltensmodelle berechnet. Die Klasse dient einmal als Sammlung sämtlicher Vektoren, speichert aber auch die Namen der Attribute ab. Die bisherige Implementierung nutzt diese Möglichkeit, um die Information über die Graphen des Benutzerverhaltens in die dritte Phase zu transportieren. Um dort wieder Verhaltensmodelle aus den Vektoren zu berechnen, werden Informationen darüber benötigt, welche Call-Information beziehungsweise welche Übergänge durch welche Dimension des Vektors modelliert werden. Um dieses Verfahren wiederverwenden zu können, sollte die BIRCH Implementierung auch die Instances Klasse verwenden. Für die ersten drei Phasen des BIRCH-Algorithmus sind neue Datenstrukturen notwendig, die sich aus den Instances berechnen lassen. Die vierte Phase erlaubt die Wiederverwendung des Instances Objekts. In dieser Phase werden die Vektoren neu auf die vorher gefundenen Clustermittelpunkte verteilt. Dafür werden ohnehin die Instances benötigt. Das Objekt muss dann nur noch mit den neu berechneten Clustermittelpunkten gefüllt werden. Anschließend wird es an die Filter übergeben, die die Rückberechnung zu den Verhaltensmodellen durchführen. Auf diese Weise lässt sich die Schnittstelle der bisherigen Implementierung weiterverwenden.

3.3. Auswahl eines Clusterings

Unklar bleibt jedoch, wie mit der Hierarchie von Clusterings umgegangen werden soll, die der BIRCH-Algorithmus berechnet. Da am Ende des Analyseprozesses eine konkrete Menge von Verhaltensmodellen stehen soll, muss es einen Mechanismus zur Auswahl eines Clusterings geben. Eine einfache Lösung wäre, einen Parameter mit der erwarteten Anzahl von Clustern zu übergeben und dann das Clustering dieser Größe, falls vorhanden, zu verwenden. Im Sinne einer selbständigen Klassifizierung sollte jedoch von Seiten des Programms eine Entscheidung getroffen werden.

Zu diesem Zweck wird die in Abschnitt 2.9 vorgestellte Heuristik zum Berechnen der Clustergröße verwendet. Diese benötigt kaum zusätzlichen Rechenaufwand, da die benötigten Werte sich auf Basis der Clustering-Features in wenigen Schritten berechnen lassen. Die Autoren in [Salvador und Chan 2004] weisen außerdem daraufhin, dass die L-Method besonders gut mit Algorithmen funktioniert, die Greedy vorgehen, also in jedem Schritt das Optimum suchen. Für die Umsetzung der L-Method wird ein Evaluationsmaß für die Clustergüte benötigt. In [Zhang u. a. 1997] werden zwei solche Maße, die sich auf Basis von CFs berechnen lassen, vorgestellt. Der *Weighted Average Cluster Radius Square* (WACRS) sowie *Weighted Average Cluster Diameter Square* (WACDS) sind wie folgt definiert:

$$WACRS(CF_1, \cdots, CF_K) = \frac{\sum_i = 1^K N_i * R_i^2}{\sum_i = 1^K N_i}$$

wobei R der Clusterradius ist, siehe Abschnitt 2.8, und

$$WACDS(CF_1, \cdots, CF_K) = \frac{\sum_i = 1^K N_i(N_i - 1) * D_i^2}{\sum_i = 1^K N_i}$$

wobei D der Clusterdurchmesser ist, siehe Abschnitt 2.8.

Zusätzlich lässt sich noch die Summe der Summe der Fehlerquadrate (SSE) ermitteln. Sie berechnet sich wie folgt:

$$SSE(CF_1, \cdots, CF_K) = \sum_{i=1}^{K} SSE_C(CF_i) \quad \text{mit}$$
$$SSE_C(CF_i) = SS_i - 2LS(LS_i/N_i) + (LS_i/N_i)^2$$

3.4. Ausgestaltung des BIRCH-Algorithmus

Der BIRCH-Algorithmus besitzt eine Reihe von optionalen Features. Zunächst einmal stellen die Autoren in [Zhang u. a. 1997] fünf verschieden Metriken zum Messen des Abstands zwischen zwei Clustering-Features (CFs) vor. Nur zwei davon, die *average inter cluster distance*, als D2 bezeichnet, und D4, die *variance increase distance* haben die *reducibility property*, die korrekte Ergebnisse für das Clustering durch den Nearest-Neighbour Algorithmus in Phase drei garantiert. Daher werden auch nur diese beiden Metriken implementiert.

In der letzten Phase des BIRCH-Algorithmus, dem Refinement, wird ein hoher Aufwand betrieben, um das Clustering noch zu verfeinern. Dieser Schritt ist daher optional. Als Kompromiss zwischen besserem Clustering und Laufzeit des Algorithmus wird wie folgt vorgegangen: Zunächst wird ein Clustering aus der Hierarchie ausgewählt. Anschließend wird auf diesem Clustering wird die Verfeinerung durchgeführt.

Der BIRCH-Algorithmus bietet noch einige Optionen, die für große Datenmengen vorgesehen sind. Dabei handelt es sich um das Outlier-Handling und die Delay-Split Option. Delay-Split greift in dem Fall ein, wenn das System den Hauptspeicher ausgeschöpft hat, aber noch Elemente eingefügt werden sollen. Das Outlier-Handling erhöht Laufzeit und Komplexität der Implementierung für eine Sonderbehandlung von Blatteinträgen, die nur sehr wenige Vektoren beinhalten. Da das Ergebnis des Clusterings im Vordergrund steht und die Testdaten (Abschnitt 2.5) weder große Datenmengen noch Ausreißer enthalten, wird auf beide Funktionen verzichtet.

Kapitel 4

Implementierung

In diesem Kapitel werden die Details der Implementierung näher erläutert. Es beginnt mit einer Übersicht über die Datenstrukturen, die für den BIRCH-Algorithmus benötigt werden. Anschließend wird auf die Filter eingegangen, die den Algorithmus in iObserve beinhalten.

4.1. Hierarchie Modell des BIRCH-Algorithmus

Wie in Abschnitt 2.8 bereits erläutert wurde, beruht der Algorithmus auf der Datenstruktur des Clustering-Features (CFs), welche eine kompakte Darstellung eines beliebig großen Clusters ermöglicht. Diese CFs werden in einem Suchbaum angeordnet, der die Grundlage für das anschließende Clustering bildet. Diese Struktur lässt sich direkt umsetzen.

Abbildung 4.1 zeigt die Klassen und ihren Zusammenhang. Der Aufbau ist typisch für eine Baumstruktur mit Knoten und Blättern: Der CFTree besteht aus AbstractNodes, die entweder vom Typ Node oder Leaf sein können. Das ClusteringFeature fungiert einerseits als Inhalt der Blätter, andererseits zur Beschreibung sämtlicher AbstractNodes. Die Klassen im Einzelnen:

ClusteringFeature Das ClusteringFeature besteht wie in Abschnitt 2.8 beschrieben aus einem Integer für die Anzahl der Elemente des Clusters sowie zweier Double Arrays, die lineare und die quadrierte Summe der Vektoren enthält. Über das Attribut *metric* lässt sich festlegen, welche der verschiedenen Metriken beim Vergleich von ClusteringFeatures genutzt werden soll. Die Methoden erlauben das Addieren von ClusteringFeatures, ihren Vergleich und eine Reihe von Metriken über den Cluster, wie etwa die Summe der Fehlerquadrate.

ICFComparisonStrategy Mit ICFComparisonStrategy lässt sich festlegen, welche Metrik die ClusteringFeatures nutzen, wenn sie ihren gegenseitigen Abstand bestimmen. Implementiert sind die Metriken D2 und D4, wie in Abschnitt 3.4 beschrieben.

AbstractNode Die Klassen AbstractNode, Node und Leaf sind die Bausteine des CF-Baums. Da eine Node entweder Leafs oder Nodes enthält, ist es unvermeidlich die beiden aus einer abstrakten Oberklasse abzuleiten. Dies macht aber auch algorithmisch Sinn: Beim

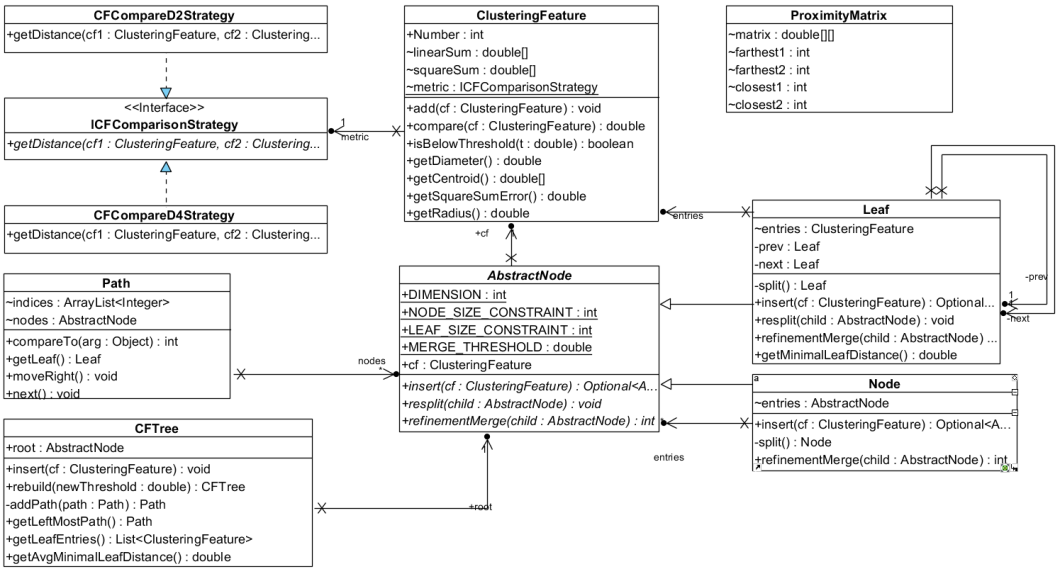

Abbildung 4.1. Klassendiagramm des Pakets org.iobserve.analysis.clustering.birch.model

rekursiven Einfügen von neuen ClusteringFeatures muss nicht jedes mal unterschieden werden, ob es in ein Leaf oder eine Node eingefügt werden soll. Die unterschiedliche Implementierung der *insert()* Methode in den beiden Klassen realisiert dies bereits. Beachtenswert ist noch die Verkettung der Blätter. Da beim Clustering nur auf den Leafs gearbeitet wird, sind diese untereinander verkettet, wodurch ein schnelles Iterieren über alle Einträge ermöglicht wird.

CFTree Der CFTree ist die Umsetzung des CF-Baums. Über die Wurzel enthält er alle ClusteringFeatures. Darüber hinaus dient er als Schnittstelle nach außen. Das Einfügen, der Neuaufbau sowie eine Liste aller Einträge werden als öffentliche Methoden angeboten. Außerdem gibt es eine Methode zum Abfragen des durchschnittlichen Mindestabstands der Clustering-Features in den Blättern. Diese Funktion wird beim Neuaufbau des Baums zum Anpassen des Schwellwerts beim Verschmelzen von ClusteringFeatures benötigt.

Path Durch die Path Klasse wird ein Pfad durch den Baum von der Wurzel bis zu einem Blatt beschrieben. Dies geschieht auf zwei unterschiedliche Weisen: Einerseits durch eine Liste der Knoten, die den Pfad ausmachen. Andererseits durch eine Liste von positiven Zahlen, die beschreiben, welchen Index der Knoten in der in der anderen Liste innehat. Beim Neuaufbau des Baums wird der alte Baum Pfad für Pfad in den neuen Baum übertragen.

ProximityMatrix Die ProximityMatrix ist eine Hilfsklasse, die aus einer Liste von CFs eine Abstandsmatrix berechnet. Dabei wird auch festgehalten, welche CFs den geringsten und welche den höchsten Abstand zueinander haben.

4.2. Stages der BIRCH Implementierung in iObserve

Die Implementierung hat drei verschiedene Ebenen. Auf der obersten ist eine teetime.framework.CompositeStage, die org.iobserve.analysis.feature.IBehaviorCompositeStage implementiert. Diese Ebene teilt sich BIRCH mit anderen Algorithmen, da sie die selbe Signatur verwenden. Wie man in Abbildung 4.2 sehen kann, werden die Daten aus der PreprocessingCompositeStage, die die Kieker-Records mit dem Benutzerverhalten verarbeitet und zu UserSessions zusammenfasst, in die ClassificationStage geleitet – hier die BirchClassificationStage. Außerdem gibt die PreprocessingCompositeStage über eine zweite Pipe einen Input vom Typ Long. Dieser Zeitstempel löst in festen Abständen die Klassifizierung aus. Die ClassificationStage erledigt alle Schritte vom Aufstellen und Clustern der Verhaltensmodelle bis hin zum Berechnen der Benutzerprofile aus dem Clustering. Die Verhaltensmodelle werden als Typ BehaviorModel and die BehaviorModelCompositeSinkStage weitergegeben, wo diese als Dateien ausgegeben, visualisiert oder anderweitig weiterverarbeitet werden.

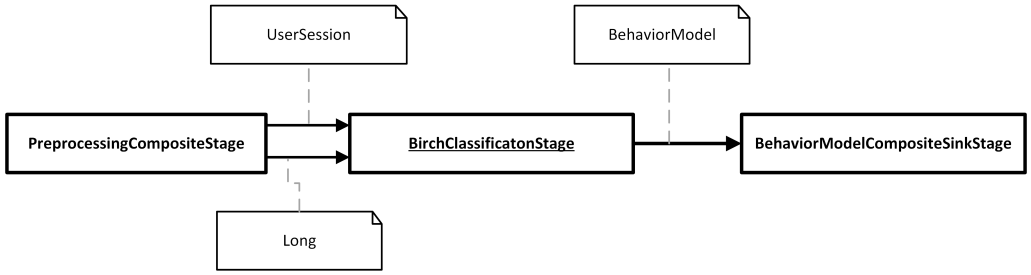

Abbildung 4.2. Einbettung der ClassificationStage in die Analyse

Eine Ebene darunter befindet sich die BirchClassificationStage. Sie setzt die Überlegungen aus Abschnitt 3.2 um. Abbildung 4.3 zeigt, dass die drei Schritte auch in drei Stages realisiert wurden. Der SessionsToInstances Filter ist eine CompositeStage, die sich aus Filtern der bereits bestehenden Implementierung zusammensetzt. Das Interface auf der darüberliegenden Ebene hat sich im Vergleich zur Implementierung von Christoph Dornieden [Dornieden 2017] geändert. Daher ist der SessionsToInstances Filter auch etwas anders zusammengestellt als der von ihm konzipierte Filter TBehaviorModelPreprocessing. Der Filter BirchClustering führt das Clustering der Instances durch und leitet ein Instances Objekt mit den Clustermittelpunkten an den TBehaviorModelCreation Filter weiter. Dieser Filter berechnet aus den Instances die Verhaltsmodelle der gefundenen Benutzerprofile

und wurde unverändert übernommen.

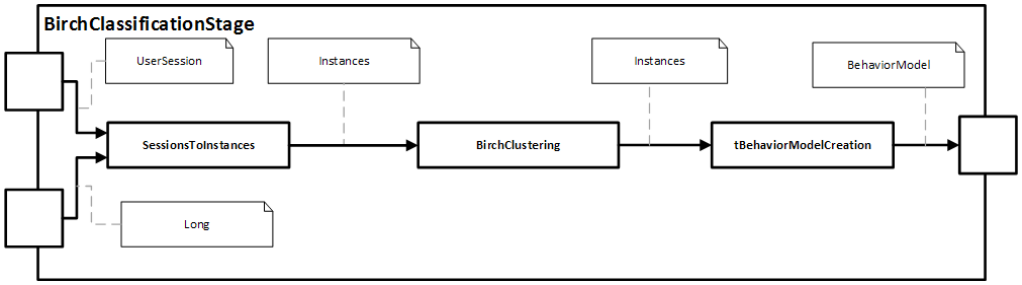

Abbildung 4.3. Aufbau der BirchClassificationStage

Bleibt noch der BirchClustering Composite Filter. Die einzelnen Stages entsprechen den Phasen des BIRCH-Algorithmus aus Abschnitt 2.8, mit folgender Ausnahme: Wie in Abschnitt 3.3 erläutert, wurde noch ein Filter zur Auswahl eines Clusterings eingebaut. Abbildung 4.4 zeigt den resultierenden Aufbau.

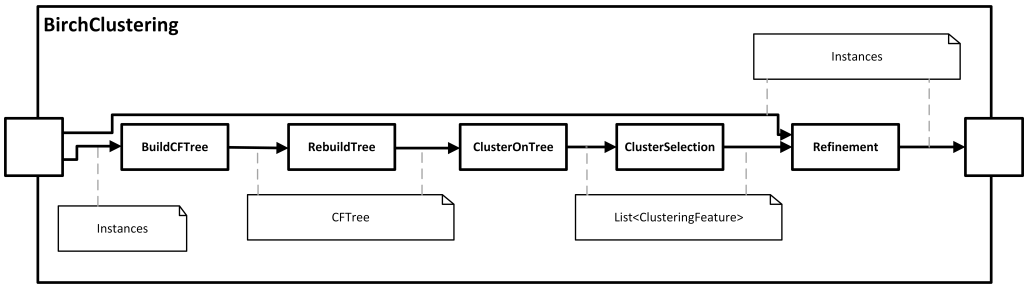

Abbildung 4.4. Aufbau der BirchClustering Stage

Im BuildCFTree Filter werden die Vektoren des Instance Objekts als Clustering-Features in den CF-Baum eingefügt. Im darauf folgenden Filter RebuildTree wird der CF-Baum verkleinert, falls er noch nicht die durch einen Parameter vorgegebene Größe hat. Der fertige Baum wird dann an die ClusterOnTree Stage geschickt. Die ClusteringFeatures in den Blättern des CF-Baums werden mit einer Anwendung des in Abschnitt 3.4 beschriebenen Nearest-Neighbour Algorithmus geclustert. Die dabei entstehenden Clusterings werden jeweils als Liste von ClusteringFeatures an die ClusterSelection Stage gesendet. Hier wird ein Clustering ausgewählt. Dazu wird ein Evaluationsgraph berechnet, auf den dann die L-Method aus Abschnitt 2.9 angewendet wird. Der Evaluationsgraph wird je nach Konfiguration mit einer der beiden Metriken erstellt, die in Abschnitt 3.3 für diesen Zweck vorgestellt wurden. Alternativ lässt sich die Stage auch so konfigurieren, dass ein Clustering mit einer vorgegebenen Größe ausgewählt wird. Das ausgewählte Clustering wird anschließend an die Refinement Stage geschickt. Diese Stage erhält sowohl das

ausgewählte Clustering, als auch die ursprünglichen Vektoren im Instance Objekt. Wie in Abschnitt 3.2 beschrieben wurde, werden die Vektoren gegen die Clustermittelpunkte des Clusterings aus der ClusterSelection Stage neu geclustert. Anschließend wird das Instances Objekt geleert und mit den Clustermittelpunkten verfeinerten Clusterings befüllt. Damit ist das Clustering fertig und wird über den Output Port der CompositeStage weitergesendet.

Kapitel 5

Parameterstudie

Der BIRCH-Algorithmus bietet, wie bereits in Abschnitt 3.4 erläutert wurde, eine Reihe von Parametern um den Algorithmus für unterschiedliche Aufgaben anzupassen. Die Parameter lassen sich in zwei Kategorien einteilen:

Performance-Parameter. Diese sind *MaxLeafEntries* und *MaxNodeEntries*. Sie kontrollieren die Größe der Blätter und Knoten des CF-Baums. Wird der Systemspeicher durch das Clustering stark ausgelastet, kann man den Algorithmus beschleunigen, indem man die Knoten- und Blättergröße gerade auf die Größe einer Speicherseite beschränkt. Der *MaxLeafEntries* Parameter bestimmt, wieviele Einträge auf Blätterebene höchstens zulässig sind. Je weniger Cluster im Baum enthalten sind, desto schneller ist das Verfahren.

Clustering-Parameter. Diese Parameter verändern, wie der Baum aufgebaut und wie das Clustering durchgeführt wird. Zu diesen zählt die Metrik, die zur Berechnung der Abstände zwischen den Clustern verwendet wird. Der *MaxLeafEntries* Parameter zählt zu beiden Kategorien. Das Nearest-Neighbour-Verfahren in Phase drei des Algorithmus führt bei anderen Ausgangsclustern auch zu anderen Ergebnissen.

Das Ziel dieser Arbeit ist die Verbesserung der Clustering Ergebnisse, nicht die Beschleunigung der Analyse. Daher wird im Folgenden die Parametrisierung mit Hinblick auf das Clustering untersucht. Auch die zur Verfügung stehenden Testdaten sind auf das Erkennen von Benutzerprofilen hin konzipiert und nicht auf einen Test der Geschwindigkeit. Dafür ist etwa die Anzahl der User-Sessions viel zu klein. Neben dem BIRCH-Algorithmus muss auch die L-Method zum Berechnen der richtigen Clusteranzahl parametrisiert werden.

5.1. Aufbau der Parameterstudie

Der Kontext des Clusterings ist folgender: iObserve verarbeitet eine Reihe von Kieker-Monitoring-Records, die dann zu User-Sessions zusammengefasst werden. An diesen Prozess setzt die hier beschriebene Implementierung an und berechnet aus den User-Sessions eine Reihe von Verhaltensmodellen, die möglichst repräsentativ für das aufgezeichnete Verhalten in den Kieker Monitoring Records sein soll. Als Maßstab für die Qualität des Clusterings werden User-Sessions aus dem JPetStore verwendet (siehe auch Abschnitt 2.5). Dornieden [2017] hat im Rahmen seiner Arbeit Testdaten erstellt. Diese enthalten sieben

sich wiederholende Benutzerprofile, die vom Clustering erkannt werden sollen. Sie unterscheiden sich auf zwei verschiedene Arten voneinander. Unterschiedliche Benutzerprofile haben unterschiedliche Call-Graphen (siehe Abschnitt 2.4). Der Typ *Account Manager* ruft zum Beispiel Login und Accounteinstellungen auf, der *Browsing User* nutzt stattdessen die Suchfunktion und lässt sich einzelne Produkte anzeigen. Neben dem Call-Graphen gibt es noch die Call-Information. Diese charakterisiert einen bestimmten Aufruf. So kann der *Entry-Call jpetstore.actions.Catalog.viewCategory()* zum Beispiel mit der Call-Information FISH versehen sein, wenn der Benutzer sich diese Kategorie hat anzeigen lassen, oder CATS, wenn er mehr an diesen Tieren interessiert war. Das Erkennen eines Benutzerprofils in einem Verhaltensmodell setzt sich daher aus diesen zwei Aspekten zusammen: Dem Erkennen des Call-Graphen und der richtigen Call-Information.

Das erste Ziel der Parameterstudie lautet somit: Eine Parametrisierung für den BIRCH-Algorithmus zu finden, die möglichst viele der sieben Benutzerprofile erkennt. Zur Erfüllung des Ziels werden folgende Fragen gestellt:

1. Für welche Clusterabstandsmetrik werden am meisten Benutzerprofile erkannt?

2. Für welche Anzahl von Blättereinträgen werden am meisten Benutzerprofile erkannt?

Zur Beantwortung beider Fragen werden Metriken benötigt, die Auskunft über die erkannten Cluster geben.

1. Sum of Square Errors (SSE) für 7 Cluster. Die Summe der Fehlerquadrate misst den Abstand der Objekte eines Clusters von ihrem Mittelpunkt, dem erkannten Benutzerprofil. Je kleiner der Wert, desto näher liegen die Verhaltensmodellen in den Daten an dem erkannten Benutzerprofil. Kleine Werte für SSE sind daher ein Hinweis auf viele richtig erkannte Benutzerprofile.

2. Anzahl der richtig erkannten Benutzerprofile.

Die SSE lässt sich leicht am Ende der Refinement-Phase berechnen und dann ausgeben. Um die Anzahl der richtig erkannten Benutzerprofile berechnen zu können, müssen noch zwei Fragen beantwortet werden:

1. Wie berechnet sich die Ähnlichkeit von zwei Verhaltensmodellen?

2. Wie ähnlich muss ein gefundenes Benutzerverhalten einem der sieben vorgegebenen Benutzerprofile sein, um als erkannt zu gelten?

Zur Bestimmung der Ähnlichkeit werden zwei Verhaltensmodellen miteinander verglichen und dabei die Anzahl der Knoten und Kanten gezählt, die sie nicht gemein haben. Somit erkennt man, wie sehr sich die Call-Graphen voneinander unterscheiden. Dazu muss untersucht werden, wie ähnlich die Call-Information der verglichenen Benutzerprofile ist sind. Die gefundenen Benutzerprofile haben aufgrund des Clusterings nur einen numerischen Wert, der nicht direkt einer diskreten Call-Information zugeordnet werden kann.

Um sie dennoch vergleichen zu können, werden für die sieben korrekten Benutzerprofile die korrespondierenden numerischen Werte berechnet. Dieser Wert wird mit dem Wert des gefundenen Verhaltensmodell verglichen.

Um Grenzwerte zu finden, bei denen Benutzerprofile noch als erkannt gelten, werden zum Vergleich die Werte herangezogen, um die sich die sieben korrekten Benutzerprofile unterscheiden. Um hinsichtlich des Call-Graphen als erkannt zu gelten, sollte der Abstand der gefundenen Verhaltensmodell kleiner sein als die Hälfte des Minimalabstands zwischen den korrekten Benutzerprofilen. Für die Call-Information wird ebenfalls die Hälfte des Minimalabstands verwendet, diese aber nach unten durch den Wert 500 begrenzt. Ist der Abstand kleiner als 500, lässt sich auf Grund der Struktur der numerischen Repräsentation eine Call-Information eindeutig einer Produktgruppe zuordnen. Abbildung 5.1 zeigt die Obergrenzen der Abstände mit der ein Verhaltensmodell noch als erkannt gilt. Damit etwa ein gefundenes Verhaltensmodell als Benutzerprofil *Account Manager* erkannt wird, darf der Unterschied bei den Knoten und Kanten des Call Graphen in Summe nicht größer als 10 sein, für die Call-Information *addItemToCart* höchstens 500, *viewItem* höchstens 500, *categoryId* höchstens 1500 und *productId* nur 550 sein.

	Call Graph	Call Information			
Profil	Max Nodes/Edges	addItemToCart	viewItem	categoryId	productId
account-manager	10	500	500	1500	550
browsing-user	20	500	12056	6500	1980
cat-lover	0	13495	500	6500	1980
fish-lover	0	3049	500	2500	3050
new-customer	3	500	500	500	500
single-cat-buyer	0	505	500	500	505
single-reptile-buyer	0	500	500	500	500

Abbildung 5.1. Grenzwerte zur Erkennung von Benutzerprofilen

Ausgehend von Abbildung 5.1 kann man erwarten, dass die Typen *Account Manager* und *Browsing User* relativ häufig erkannt werden. Sie sind deutlich von den anderen Benutzerprofilen zu unterscheiden und haben daher auch großzügigere Grenzwerte. *Cat Lover* und *Fish Lover* haben den selben Call-Graphen und sind bereits in dieser Hinsicht schwer zu erkennen. Selbiges gilt für *Single Reptile* und *Single Cat*, die darüberhinaus auch noch sehr ähnlich zu *New Customer* sind. Ím Gegensatz zu *Fish Lover* und *Cat Lover* haben diese auch noch sehr ähnliche Call-Information, was die Erkennung weiter erschwert.

Das zweite Ziel dieses Abschnitts ist eine Parametrisierung zu finden, bei der die von der L-Method berechnete Clusteranzahl möglichst nahe an die korrekte Anzahl von sieben Clustern herankommt. Dies lässt sich durch zwei Fragen beantworten:

1. Wie viele Cluster werden erkannt, wenn man das Evaluationsmaß variiert?

2. Wie abhängig ist die Anzahl der Cluster von Parametrisierung des BIRCH-Algorithmus?

Als Metrik dient die von der L-Method berechnete Clusteranzahl. Als Evaluationsmaße müssen die in Abschnitt 3.4 vorgestellten *Weighted Average Cluster Radius Square* (WACRS), *Weighted Average Cluster Diameter Square* (WACDS) sowie die Summe der Fehlerquadrate (SSE) miteinander verglichen werden. Zur Durchführung wird für jede Kombination von L-Method Evaluationsmetrik, Clusterabstandsmetrik sowie der Anzahl von Blättereinträgen das Verfahren durchgeführt. Es ist nicht sinnvoll, jede Anzahl von Blatteinträgen auszuprobieren. Der Aufwand wäre enorm, ohne zusätzlichen Erkenntnisgewinn: Die Rebuilding Phase verkleinert den Baum in jedem Schritt um etwa die Hälfte der Einträge. Dies wird dadurch abgebildet, indem wir die volle Anzahl von Vektoren zulassen, etwa 150, gefolgt von 75, 35 und 18. Als Metriken werden die *average inter cluster distance*, als D2 bezeichnet, und D4, die *variance increase distance*, verwendet. Die Anzahl der Cluster sowie die SSE werden zu diesen Zwecken in eine Datei geschrieben. Die ausgegebenen Verhaltensmodelle werden mit den sieben Benutzerprofilen verglichen und dann auf die beschriebene Art und Weise als erkannt oder nicht erkannt gezählt.

5.2. Diskussion der Ergebnisse

Abbildung 5.2 zeigt das Resultat der Studie. Die Ergebnisse des Clusterings zeigen, dass die SSE für alle Anzahlen von Blatteinträgen über 18 sehr groß ist. Die Wahl der Clusterabstandsmetrik spielt im Vergleich eine untergeordnete Rolle. Für 150 und 35 Blatteinträge ist D4 jeweils eine Größenordnung besser als D2 und daher nach diesem Kriterium vorzuziehen. Die ideale Parametrisierung ist nach SSE also 18 Blatteinträge und Metrik D4.

Die Anzahl der gefundenen Benutzerprofile ergibt Folgendes: Bei 18 Blatteinträgen konnte stets ein Profil erkannt werden. Für alle anderen Parametrisierungen gab es keine Treffer. Die Wahl der Metrik hat keinen Einfluss auf die Anzahl der erkannten Benutzerprofile. Die Ergebnisse des Clusterings sind in sich stimmig und legen folgende Parametrisierung nahe: 18 Blatteinträge und aufgrund der geringeren SSE die Metrik D4.

Bei der L-Method ist die vorhergesagte Anzahl von Clustern für alle drei Evaluationsmaße fast genau gleich. Nur WACDS, D4, 35 unterscheidet sich mit einer Clusteranzahl von fünf von den anderen Evaluationsmaßen, für die bei ansonsten gleichen Parametern drei Cluster berechnet wurden. Ansonsten liegt die Anzahl je nach Clustering entweder bei 21, im Bereich sechs bis acht oder bei drei. Eine Clusteranzahl von 21 ist nicht sinnvoll. Sechs bis acht gefundene Cluster entsprechen dem erwarteten Ergebnis. Falls das Clustering sehr ähnliche Benutzerprofile wie Fish/CatLover zusammengefasst hat, kann auch eine Clusteranzahl von drei noch sinnvoll sein. Dass sich die Ergebnisse der Evaluationsmaße kaum unterscheiden, ist ein starkes Anzeichen dafür, dass sich die Maße zu ähnlich sind. Alle drei beruhen im Kern auf dem Abstand der Vektoren zu ihren Clustermittelpunkten. Interessant wäre zu untersuchen, welche Ergebnisse die L-Method mit Maßen berechnet, die nicht auf dem Abstand der Vektoren zu ihren Clustermittelpunkten basieren. Bewertet

Lmethod Metric	Cluster Metric	LeafEntries	#Clusters	SSE 7	#profiles
Diameter	D2	150	21	6.66E+10	0
Diameter	D2	75	6	5.95E+09	0
Diameter	D2	35	3	5.95E+09	0
Diameter	D2	18	3	3046	1
Diameter	D4	150	7	5.95E+09	0
Diameter	D4	75	8	5.95E+09	0
Diameter	D4	35	5	8.13E+08	0
Diameter	D4	18	3	3046	1
Radius	D2	150	21	6.66E+10	0
Radius	D2	75	6	5.95E+09	0
Radius	D2	35	3	5.95E+09	0
Radius	D2	18	3	3046	1
Radius	D4	150	7	5.95E+09	0
Radius	D4	75	8	5.95E+09	0
Radius	D4	35	3	8.13E+08	0
Radius	D4	18	3	3046	1
SSE	D2	150	21	6.66E+10	0
SSE	D2	75	6	5.95E+09	0
SSE	D2	35	3	5.95E+09	0
SSE	D2	18	3	3046	1
SSE	D4	150	7	5.95E+09	0
SSE	D4	75	8	5.95E+09	0
SSE	D4	35	3	8.13E+08	0
SSE	D4	18	3	3046	1

Abbildung 5.2. Clustering Ergebnisse in Abhängigkeit von Evaluationsmaß, Clusterabstandsmetrik und Anzahl von Blatteinträgen

man nur die Anzahl der gefundenen Cluster, wäre das Ergebnis wohl, dass die LMethod das Evaluationsmaß WACDS, die Metrik D4 und 150 Blatteinträge bevorzugt.

Vergleicht man das Ergebnis der L-Method jedoch mit dem des Clusterings, fällt auf, dass die L-Method genau dort die erwartete Anzahl vorhersagt, wo das Clustering nach beiden Metriken kein gutes Ergebnis erreicht. Dies zieht die Ergebnisse zur Parametrisierung der L-Method in Zweifel. Wenn das Clustering keine sinnvollen Cluster erkennt, kann die L-Method auch nur schwer eine korrekte Anzahl vorhersagen. Die Ergebnisse der Clusterings sind nicht gut genug um die L-Method abschließend zu bewerten. Sicher ist, dass das Zusammenspiel der L-Method mit den hier verwendeten Evaluationsmaßen und dem BIRCH-Algorithmus nicht funktioniert hat.

Kapitel 6

Evaluation

Nachdem im letzten Kapitel geeignete Parameter für die Anwendung des BIRCH-Algorithmus auf die JPetStore Testdaten gefunden wurden, werden nun die Clustering Ergebnisse des BIRCH-Algorithmus mit den bereits in iObserve vorhandenen Verfahren verglichen. So lässt sich beurteilen, ob das veränderte Clustering die Klassifizierung des Benutzerverhaltens verbessern konnte. Es gibt zwei bereits implementierte Verfahren: XMeans und das Expectation Maximization (EM) Clustering Verfahren.

6.1. Aufbau der Evaluation

Der Kontext der Evaluation ist der gleiche wie in Abschnitt 5.1: iObserve soll aus einer Reihe von Testdaten die typischen Benutzerprofile berechnen. Die Testdaten bestehen den aus Entry-Calls, die von sieben unterschiedlichen Benutzerprofile bei wiederholten Besuchen des JPetStores aufgezeichnet wurden (siehe Abschnitt 2.5). Alle drei Clustering Verfahren verwenden die selbe Schnittstelle, sodass auch wirklich nur das Clustering evaluiert wird. Ziel ist es, die Ergebnisse der drei Verfahren beim Bestimmen der Benutzerprofile aus den Testdaten zu vergleichen. Um dieses Ziel zu erreichen, müssen folgende Fragen beantwortet werden:

1. Wie gut ist das Clustering durch das EM-Verfahren?
2. Wie gut ist das Clustering durch das XMeans-Verfahren?
3. Wie gut ist das Clustering durch den BIRCH-Algorithmus?

Um die Qualität des jeweiligen Clusterings zu beurteilen, werden die folgenden Metriken verwendet:

1. Sum of Square Errors (SSE). Die Summe der Fehlerquadrate misst den Abstand der Objekte eines Clusters von ihrem Mittelpunkt, das dem Verhaltensmodell für diesen Cluster entspricht. Je kleiner der Wert, desto näher liegen die Benutzerprofile in den Daten an dem berechneten Verhaltensmodell. Kleine SSE sind daher ein Hinweis auf viele richtig erkannte Benutzerprofile.
2. Anzahl der richtig erkannten Benutzerprofile. Sieben erkannte Benutzerprofile sind möglich.

Die Berechnung der Anzahl wird wie in Abschnitt 5.1 beschrieben durchgeführt.

Der BIRCH-Algorithmus verwendet zu diesem Zweck die Parametrisierung aus Kapitel 5. Da die automatische Bestimmung der Clustergröße für diese Parameter keine vernünftigen Ergebnisse brachte, wird die Größe des korrekten Clusterings vorgegeben. XMeans ist durch die Werte *ExpClusterSize* und *Variance* parametrisierbar. Für alle Werte in dem Intervall (*ExpClusterSize - Variance*) bis (*ExpClusterSize + Variance*) werden Clusterings berechnet, aus denen XMeans dann eins auswählt. Für einen aussagekräftigen Vergleich werden die Ergebnisse von XMeans für unterschiedliche Parameter um den Wert von sieben erwarteten Benutzerprofile herum berechnet. Das EM-Clustering ist ohne Optionen zur Konfiguration implementiert und wird daher mit den Standardparametern aufgerufen. Dies bedeutet, dass das EM-Clustering selbständig die Anzahl der Cluster ermittelt und 100 Iterationen durchläuft.

6.2. Diskussion der Ergebnisse

Die Ergebnisse der Metriken sind in Abbildung 6.1 aufgeführt. Unter *Verfahren* sind die Clustering Algorithmen aufgelistet. Die Zahlen hinter den XMeans Einträgen sind die *ExpClusterSize* und *Variance* Parameter, mit denen das Clustering erstellt wurde. *Gefundene Cluster* ist die vom Verfahren berechnete Anzahl von Verhaltensmodellen. Bei BIRCH wurde diese vorgegeben. *SSE* ist die Sum of Square Errors für das jeweilige Verfahren. Die Anzahl der erkannten Benutzerprofile kann man in der gleichnamigen Spalte ablesen.

Das EM-Verfahren liegt in der Metrik *SSE* deutlich über den anderen Verfahren. Dementsprechend hoch ist der geometrische Abstand der gefundenen Benutzerverhalten von den Benutzerverhalten in den Testdaten. Ferner konnte es kein Benutzerprofil erkennen. Beides zusammen bedeutet eine schwache Erkennungsrate. Alle drei Parametrisierungen von XMeans weisen eine deutlich geringere *SSE* als das EM-Verfahren auf. Die gefundenen Benutzerverhalten sind also deutlich repräsentativer für die Benutzerverhalten in den Testdaten. Eine vollständige Erkennung, die einer SSE von null entsprechen würde, kann XMeans aber nicht erreichen. Das zeigt sich auch in der Anzahl der erkannten Benutzerprofile: Zwei der Parametrisierungen konnten eins der sieben Benutzerprofile, die in den Testdaten enthalten sind, richtig erkennen. Dies ist vergleichbar mit den Werten von BIRCH. Dessen *SSE* ist ähnlich und auch hier wurde ein Benutzerprofil erkannt.

Insgesamt bleiben zwei Erkenntnisse. BIRCH liefert mit abgestimmten Parametern ein vergleichbares Ergebnis zu dem von XMeans. Beide waren besser als das EM-Verfahren. Es bleibt aber festzuhalten, dass sowohl BIRCH als auch XMeans nur ein Benutzerverhalten vollständig richtig erkennen konnten. Insbesondere konnten die Ergebnisse aus [Dornieden 2017] nicht reproduziert werden, der immerhin fünf Benutzerprofile richtig zuordnen konnte. Zum Teil ist dies durch unterschiedliches Vorgehen bei der Evaluation zu erklären, aber er konnte in seiner Arbeit auch geringere *SSE* Werte erreichen.

Um Anhaltspunkte für die Gründe der geringen Erkennungsrate zu erhalten, werden die bei der Evaluation erzeugten Daten noch einmal untersucht. Das korrekt erkannte

Verfahren	Gefundene Cluster	SSE	Erkannte Benutzerprofile
EM Clustering	6	4.14E+10	0
XMEANS 6/3	4	3436.42	0
XMEANS 7/3	6	3106.51	1
XMEANS 8/3	7	3051.31	1
BIRCH	7	3046	1

Abbildung 6.1. Ergebnisse der unterschiedlichen Clustering Verfahren

Benutzerverhalten war das des *NewCustomer*.

Beim Aufstellen der Metriken in Abschnitt 5.1 wurden die sieben zu erkennenden Benutzerprofile miteinander verglichen. Dabei stellten sich *NewCustomer* als sehr ähnlich hinsichtlich Call-Graph und Call-Information zu den Benutzerprofilen *SingleCat* und *SingleReptile* heraus, was eine Erkennung eher hätte erschweren sollen. Diese Ähnlichkeit des Benutzerverhaltens hat sich scheinbar nicht in einem geringen geometrischen Abstand der zu clusternden Vektoren niedergeschlagen. BIRCH fasst gerade die Vektoren zusammen, die einen geringen geometrischen Abstand aufweisen.

Eine andere Erklärung könnte sein, dass die Dimension der Vektoren einfach zu groß ist um noch sinnvoll nach geometrischem Abstand zu clustern. [Beyer u. a. 1999b] legen in ihrer Arbeit dar, dass für Vektoren großer Dimensionen die Suche nach dem nächstgelegenen Vektor nicht immer den Vektor hervorbringt, die diesem auch am ähnlichsten ist.

Kapitel 7

Verwandte Arbeiten

Die meisten Arbeiten in diesem Bereich widmen sich einem ganzheitlichen Ansatz zur Klassifizierung von Benutzerverhalten, in dem das Clustering nur ein Aspekt darstellt. van Hoorn u. a. [2014] verfolgen dieses Ziel. Mit ihrem WESSBAS Ansatz Workloads aus Beobachtungsdaten erstellt. Das Benutzerverhalten wird hier durch Markov-Chains modelliert, die dann durch den XMeans Algorithmus geclustert werden.

Peter [2016] verwendet den WESSBAS Ansatz, um in iObserve das Palladio-Component Modell mit einem Workload basierend auf dem Benutzerverhalten füllen. Auch hier wird das Clustering mit XMeans durchgeführt.

Dornieden [2017] führt diese Entwicklung fort, indem er statt des WESSBAS Ansatzes eine neue Modellierung des Benutzerverhaltens einführt. Das Clustering wird aber auch hier durch XMeans realisiert. Der Autor weist bereits auf Probleme mit XMeans hin: Die gesetzten Ziele an die Erkennung von Benutzerprofilen konnte nicht erreicht werden. Die vorliegende Arbeit ist eine Weiterentwicklung dieser vorhergehenden Arbeiten. Statt XMeans wird ein hierarchischer Clustering Algorithmus eingesetzt und gibt Aufschlüsse über die Eignung unterschiedlicher Algorithmen. Auch für das Problem, dass XMeans einen Suchbereich für die richtige Anzahl von Clustern benötigt, präsentieren wir eine Lösung.

Kapitel 8

Fazit und Ausblick

8.1. Fazit

Die Klassifizierung des Benutzerverhaltens in iObserve wird durch ein Clustering der beobachteten User-Sessions realisiert. Die bislang zum Clustering eingesetzten Verfahren konnten die Ansprüche nicht gänzlich erfüllen. Das Unterscheiden von Benutzerverhalten gelang nicht in allen Fällen. Um die Klassifizierung in dieser Hinsicht weiterzuentwickeln, wurde ein Clustering-Verfahren aus dem Bereich des hierarchischen Clusterings ausgewählt. Um das langfristige Ziel einer Echtzeit-Analyse zu gewährleisten, fiel die Wahl auf den BIRCH-Algorithmus, der eine besonders geringe Laufzeitkomplexität aufweist. Dies wird durch eine Reduktion der Daten auf einen Clustering-Feature-Baum erreicht, so dass die Daten nur einmal eingelesen werden müssen.

Ein weiteres Problem in der bestehenden Klassifizierung ist die Bestimmung der Anzahl der Cluster in den Daten. Das XMeans Verfahren benötigt die Angabe eines Bereichs für die Clustergröße. Außerhalb von Testfällen ist es jedoch unklar, wie viele unterschiedliche Benutzerprofile zu erwarten sind. Zur Lösung dieses Problems wurde die L-Method zur Bestimmung der Clusteranzahl in einem hierarchischen Clustering ausgewählt.

Sowohl BIRCH als auch die L-Method wurden in iOberve implementiert. Die Implementierung fügt sich in die bestehende Analyse ein und verwendet die selbe Schnittstelle wie die bereits implementierten Verfahren. Außerdem sind die Parameter für BIRCH und die L-Method sind über die iObserve Konfiguration einstellbar.

Am Beispiel des JPetStores wurde die Implementierung getestet. Zunächst wurde eine günstige Parametrisierung für den BIRCH-Algorithmus und die L-Method gesucht. Hierbei zeigte sich bereits, dass der Algorithmus die in den Testdaten enthaltenen Benutzerprofile bis auf eine Ausnahme nicht erkennen konnte. Die L-Method lag mit der von ihr berechneten Clusteranzahl in der erwarteten Größenordnung. Sie konnte aber gerade für die Parameter, die beim Clustering vergleichsweise gute Resultate erzielten, keine richtige Vorhersage abgeben. Möglicherweise benötigt die L-Method auch bessere Clustering Ergebnisse, um die Clusteranzahl korrekt zu berechnen.

Schließlich wurden die Ergebnisse der BIRCH Clusterings mit denen der bereits vorhandenen Verfahren, XMeans und EM-Clustering, verglichen. Für die Testdaten des JPetStores konnte der BIRCH-Algorithmus mehr Benutzerprofile erkennen als das EM-Clustering. Auch die Summe der Fehlerquadrate deutete auf ein besseres Clustering hin. XMeans und

BIRCH lieferten vergleichbare Resultate, wenn man beide Verfahren mit der korrekten Anzahl von Clustern parametrisierte.

Kann ein hierarchisches Clustering Verfahren nun bessere Ergebnisse beim Klassifizieren von Benutzerverhalten liefern? Diese Frage kann aufgrund der erzielten Ergebnisse klar mit Nein beantwortet werden.

8.2. Ausblick

Die Klassifizerung des Benutzerverhaltens konnte durch ein hierarchisches Verfahren nicht verbessert werden. Es könnten aber noch weitere Clustering Ansätze verfolgt werden. Da bereits im kleinen JPetstore Dimensionen von über 250 erreicht werden, könnte man generell über Verfahren nachdenken, die sich auf hohe Dimensionalität spezialisieren. In der Literatur wird häufig vom Dimensionsfluch gesprochen, da in hohen Dimensionen viele übliche Verfahren wie XMeans oder eben hierarchisches Clustering keine guten Resultate mehr erzielen. Es könnte also lohnend sein, Ansätze wie Subspace Clustering [Berkhin 2006] zu verfolgen.

Möglicherweise liegen die Probleme auch nicht in den Clustering-Algorithmen, sondern in der Modellierung der Verhaltensmodelle. Aktuell bestehen die Verhaltensmodelle aus dem Call-Graphen und der Call-Information. Ein Benutzer, der zunächst *viewCategory=&categoryId=FISH* und dann *Cart.addItemToCart()=&productId=FI-SW-0* aufruft, wird der gleiche CallGraph zugeordnet wie einem Benutzer, der erst *viewCategory=&categoryId=CAT* und dann *Cart.addItemToCart()=&productId=FL-DSH-01* aufruft. Würden etwa *viewCategory=&categoryId=CAT* und *viewCategory=&categoryId=FISH* als unterschiedliche Knoten im Call-Graph modelliert, wären die Verhaltensmodelle möglicherweise besser voneinander zu unterscheiden. Dies würde allerdings die Dimension der zu clusternden Objekte weiter erhöhen. Ein solcher Ansatz müsste also mit einem Clustering-Verfahren kombiniert werden, das für hohe Dimensionen konzipiert ist.

Die Evaluation der L-Method in Kapitel 5 brachte kein eindeutiges Ergebnis. Um sie endgültig bewerten zu können, müsste die L-Method mit einem Algorithmus zusammen eingesetzt werden, der die Benutzerprofile in den Testdaten erkennen kann.

Auch der BIRCH-Algorithmus könnte weiter angepasst werden. Der Nearest-Neighbour Algorithmus in Phase drei von BIRCH könnte durch andere Algorithmen ersetzt werden. So ließen sich die Ergebnisse von BIRCH verbessern ohne die Vorteile des Algorithmus zu verlieren.

Literaturverzeichnis

[Amigo u. a. 2009] E. Amigo, J. Gonzalo, J. Artiles und F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information retrieval* 12.4 (2009), Seiten 461–486.

[Basili 1992] V. R. Basili. Software modeling and measurement: the Goal/Question/Metric paradigm. Technischer Bericht. 1992. (Siehe Seite 18)

[Becker u. a. 2009] S. Becker, H. Koziolek und R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82.1 (2009), Seiten 3–22. (Siehe Seite 6)

[Ben-Hur u. a. 2001] A. Ben-Hur, D. Horn, H. T. Siegelmann und V. Vapnik. Support vector clustering. *Journal of machine learning research* 2.Dec (2001), Seiten 125–137. (Siehe Seite 20)

[Berkhin 2006] P. Berkhin. A Survey of Clustering Data Mining Techniques. In: *Grouping Multidimensional Data: Recent Advances in Clustering*. Herausgegeben von J. Kogan, C. Nicholas und M. Teboulle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, Seiten 25–71. URL: https://doi.org/10.1007/3-540-28349-8_2. (Siehe Seiten 2, 19 und 44)

[Beyer u. a. 1999a] K. Beyer, J. Goldstein, R. Ramakrishnan und U. Shaft. When Is "Nearest Neighbor" Meaningful? In: *Database Theory — ICDT'99*. Herausgegeben von C. Beeri und P. Buneman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, Seiten 217–235. (Siehe Seite 19)

[Beyer u. a. 1999b] K. Beyer, J. Goldstein, R. Ramakrishnan und U. Shaft. When is "nearest neighbor" meaningful? In: *International conference on database theory*. Springer. 1999, Seiten 217–235. (Siehe Seite 39)

[Cherng und Lo 2001] J.-S. Cherng und M.-J. Lo. A hypergraph based clustering algorithm for spatial data sets. In: *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE. 2001, Seiten 83–90. (Siehe Seite 20)

[Contreras und Murtagh 2012] P. Contreras und F. Murtagh. Fast, linear time hierarchical clustering using the Baire metric. *Journal of Classification* 29.2 (2012), Seiten 118–143. (Siehe Seite 20)

[Dornieden 2017] C. Dornieden. Knowledge-Driven User Behavior Model Extraction for iObserve. Dissertation. Kiel University, 2017. (Siehe Seiten 1, 7, 21, 27, 31, 38 und 41)

[Estivill-Castro und Lee 2000] V. Estivill-Castro und I. Lee. Amoeba: Hierarchical clustering based on spatial proximity using delaunay diagram. In: *Proceedings of the 9th International Symposium on Spatial Data Handling. Beijing, China*. Citeseer. 2000, Seiten 1–16. (Siehe Seite 20)

[Geva 1999] A. B. Geva. Hierarchical unsupervised fuzzy clustering. *IEEE transactions on fuzzy systems* 7.6 (1999), Seiten 723–733. (Siehe Seite 20)

[Guha u. a. 1998] S. Guha, R. Rastogi und K. Shim. CURE: an efficient clustering algorithm for large databases. In: *ACM Sigmod Record*. Band 27. 2. ACM. 1998, Seiten 73–84. (Siehe Seite 20)

[Guha u. a. 2000] S. Guha, R. Rastogi und K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information systems* 25.5 (2000), Seiten 345–366. (Siehe Seite 20)

[Hasselbring u. a. 2013] W. Hasselbring, R. Heinrich, R. Jung, A. Metzger, K. Pohl, R. Reussner und E. Schmieders. iObserve: Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems. Research Report. Kiel, Germany: Christian-Albrechts-Universität Kiel, Okt. 2013. URL: http://eprints.uni-kiel.de/22077/. (Siehe Seiten 1 und 5)

[Jain 2010] A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31.8 (2010), Seiten 651–666. (Siehe Seite 8)

[Jung u. a. 2017] R. Jung, M. Adolf und C. Dornieden. Towards Extracting Realistic User Behavior Models. *Softwaretechnik-Trends* (2017). (Siehe Seite 8)

[Karypis u. a. 1999] G. Karypis, E.-H. Han und V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer* 32.8 (1999), Seiten 68–75. (Siehe Seite 20)

[Kaufman und Rousseeuw 1990] L. Kaufman und P. J. Rousseeuw. Finding groups in data. an introduction to cluster analysis. *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics, New York: Wiley, 1990* (1990). (Siehe Seite 20)

[Li und Biswas 2002] C. Li und G. Biswas. Unsupervised learning with mixed numeric and nominal data. *IEEE Transactions on Knowledge and Data Engineering* 14.4 (2002), Seiten 673–690. (Siehe Seite 20)

[Mark und Csaba 2007] K. Mark und L. Csaba. Analyzing customer behavior model graph (CBMG) using Markov chains. In: *Intelligent Engineering Systems, 2007. INES 2007. 11th International Conference on*. IEEE. 2007, Seiten 71–76. (Siehe Seite 7)

[Menasce und Almeida 2000] D. Menasce und V. Almeida. Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. 2000. (Siehe Seite 7)

[Mollineda und Vidal 2000] R. A. Mollineda und E. Vidal. A relative approach to hierarchical clustering. *Pattern Recognition and Applications* 56 (2000), Seiten 19–28. (Siehe Seite 20)

[Murtagh und Contreras 2012] F. Murtagh und P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1 (2012), Seiten 86–97. (Siehe Seite 20)

[Olson 1995] C. F. Olson. Parallel algorithms for hierarchical clustering. *Parallel computing* 21.8 (1995), Seiten 1313–1325. (Siehe Seite 16)

[Pelleg und Moore 2000] D. Pelleg, A. W. Moore u. a. X-means: Extending k-means with efficient estimation of the number of clusters. In: *Icml*. Band 1. 2000, Seiten 727–734. (Siehe Seite 1)

[Peter 2016] D. Peter. Observing and modeling workload characteristics of dynamic cloud applications. Dissertation. Department of Informatics: Institute for Program Structures und Data Organization (IPD), 2016. (Siehe Seite 41)

[Salvador und Chan 2004] S. Salvador und P. Chan. Determining the number of clusters/-segments in hierarchical clustering/segmentation algorithms. In: *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*. IEEE. 2004, Seiten 576–584. (Siehe Seiten 17, 18 und 22)

[Schmidt u. a. 1996] D. Schmidt, M. Stal, H. Rohnert und F. Buschmann. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. 1996. (Siehe Seite 6)

[Van Hoorn u. a. 2014] A. van Hoorn, C. Vögele, E. Schulz, W. Hasselbring und H. Krcmar. Automatic extraction of probabilistic workload specifications for load testing session-based application systems. In: *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*. ICST (Institute for Computer Sciences, Social-Informatics und Telecommunications Engineering). 2014, Seiten 139–146. (Siehe Seite 41)

[Wulf u. a. 2017] C. Wulf, W. Hasselbring und J. Ohlemacher. Parallel and Generic Pipe-and-Filter Architectures with TeeTime. In: *Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on*. IEEE. 2017, Seiten 290–293. (Siehe Seite 6)

[Xu und Wunsch 2005] R. Xu und D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks* 16.3 (2005), Seiten 645–678. (Siehe Seite 9)

[Zhang u. a. 1997] T. Zhang, R. Ramakrishnan und M. Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery* 1.2 (1997), Seiten 141–182. (Siehe Seiten 9, 10, 16, 20, 22, 23)