Mining Perennial Objects

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakulität für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Zaigham Faraz Siddiqui
geb. am 29 Mai 1982 in Abu Dhabi, UAE

Gutachter:

Eyke Hüllmeier	-	University of Marburg
João Gama	-	University of Porto
Myra Spiliopoulou	-	University of Magdeburg

Verteidigung am 13. Mai 2013, Magdeburg

Abstract

In many real world domains, data is composed of multiple interrelated streams. For example, an online retail store that keeps a customer warehouse, where multiple streams consists of stream of customers, products and the activities (that link a customer to the products he has purchased) constitute multiple streams. Unlike, in the conventional stream mining paradigm, where objects are observed, incorporated into the model and then forgotten, objects such as customers and products may not be forgotten, for they may be referenced later. When mining is performed over multiple streams, the stream, which is central to learning, is designated as *target* stream. The target stream consists of *perennial* objects that may not be forgotten, e.g., customers. It is fed by *other* streams *other* streams that may contain either perennial objects (e.g., products) or ephemeral objects (those that can be forgotten, e.g., stream of activities). Over time, more activities are recorded for the customers, while new ones may arrive. As activities accumulate, the customers may exhibit a change in their behaviour, e.g., a customer may stop being trustworthy.

Perennial objects put forward many unique challenges for mining algorithms, i.e., they 1) constitute a stream, 2) are multi-relational objects that are linked to objects from other streams, 3) they may not be forgotten, 4) may evolve over time, and 5) may change their class labels. The existing stream mining algorithms are designed for simpler objects that do not reappear and thus do not change their labels, a new solution is needed.

Our solution for mining perennial objects builds upon the conventional mining methods and extends the stream mining paradigm. The first component of our solution is a pre-processing method that combines multiple streams into a single stream. This method also incorporates a cache-based mechanism for handling perennial objects when resource constraints may force some objects to be discarded from the main memory. The objects to be discarded are spilled on the secondary storage and can be recalled when needed again.

An incremental version of K-Means, which we adapted to handle the dynamic nature of perennial objects, then operates over the propositionalised stream to discover homogeneous groups of perennial objects. We have also developed a classification algorithm, TrIP, that learns a tree over the labelled propositionalised stream. TrIP incorporates methods that can update the induced tree by tracking evolution of perennial objects and detecting concept drifts. We also enrich TrIP with classification rules that can capture and record complex patterns for transformation, during preprocessing phase.

We have evaluated our solution on real and synthetic datasets. For generating the synthetic dataset, we also developed a multi-stream generator, to create synthetic datasets. The experimental results uncovered important aspects about the nature of the perennial objects. One of the key findings of the evaluation was; the sizes of the caches maintained over perennial streams had a direct impact on quality of the models. The usage of complex patterns, based on classification rules, during the pre-processing phase, leads to concise and robust models.

Zussamenfassung

Daten aus vielen realen Domänen setzen sich aus mehreren verbundenen Datenströmen zusammen; zum Beispiel ein Online-Verkaufsgeschäft welches die Daten seiner Kunden speichert: hierbei existieren ein Kundenstrom, ein Strom der Produkte und ein Strom der Kundenaktivitäten welche den Kunden und die Produkte, die er gekauft hat, verbinden. Anders als bei dem konventionellen Streamminingparadigma, wo Objekte beobachtet, in das Modell integriert und dann vergessen werden, können Objekte wie Kunden und Produkte nicht vergessen werden, da auf sie referenziert wird.

Bei der Anwendung von mehreren Strömen, wird einer von diesen Strömen als Target Strom bezeichnet; dieser ist der Mittelpunkt des Lernens. Der Target Strom beinhaltet Perennial Objekte, welche immer wieder erscheinen und nicht vergessen werden dürfen, z.B. Kunden. Der Target Strom wird von anderen Strömen, welche Ephemeral (z. B. ein Strom von Aktivitäten) oder Perennial Objekte (z.B. Produkte) enthalten können, beeinflusst.

Ãœber die Zeit werden viele Aktivitäten von bestehenden Käufern gesammelt. Durch die gesammelten Informationen lassen sich Veränderungen im Verhalten der Kunden erkennen, beispielsweise kann das Verhalten eines Kunden dazu führen, das dieser nicht mehr als vertrauenswürdig angesehen wird. Im Zusammenhang mit Perennial Objekten ergeben sich viele neue Herausforderungen für Data Mining-Algorithmen, sie 1) bilden einen Strom, 2) sind Multirelationale Objekte, verbunden mit Objekten aus anderen Strömen, 3) werden nicht vergessen, 4) können im Laufe der Zeit evolvieren und 5) können ihre Klasse verändern. Die bestehenden Stream-Mining-Algorithmen sind für einfachere Objekte, die nicht wieder erscheinen und somit nicht ihre Label verändern, entwickelt.

Unser Ansatz für das Mining von Perennial Objekten orientiert sich an den konventionellen Data Mining Methoden. Der erste Schritt ist ein Vorverarbeitungsverfahren welcher mehrere Datenströme (mit Perennial Objekten) in einen einzelnen Strom transformiert. Dieses Verfahren beinhaltet auch einen Cache-Mechanismus zur Handhabung von Perennial Objekten, welcher notwendig ist wenn nicht ausreichdend große Rechner-Ressourcen zur Verfügung stehen d.h. Objekte welche nicht oft erscheinen, befinden sich nicht im Hauptspeicher. Sie werden dagegen im sekundären Speichersystem gehalten und können bei Bedarf wieder abgerufen werden. Eine inkrementelle Version von K-Means, die wir angepasst haben um die dynamische Natur der Perennial Objekten zu

behandeln, arbeitet auf einem propositionalisiertem Strom um homogene Gruppen von Perennial Objekte zu entdecken. Darüberhinaus haben wir einen Klassifikationsalgorithmus, TrIP, entwickelt welcher einen Entscheidungsbaum über den annotierten propositionalierten Strom lernt. TrIP umfasst Methoden, zum updaten des induzierten Baums, indem die Entwicklung der Perennial Objekte über die Zeit verfolgt wird und eventuelle Ã $f$nderungen im Konzept der Objekte erkannt werden. Wir haben TrIP mit klassifikations Regeln erweitert, die dazu dienen komplexe Muster für die Transformation innerhalb der Vorverarbeitungsphase zu erfassen.

Unsere Lösung haben wir auf echte und synthetische Datensätze ausgewertet. Um synthetische Datensätze zu erstellen, haben wir einen Multistreamgenerator MultiGen entwickelt. Einige der wichtigsten Ergebnisse der Evaluierung waren: die Größe des Caches hatte einen direkten Einfluss auf die Qualität der Modelle; die Verwendung von komplexen Mustern für die Klassifikationsregeln, während der Vorverarbeitung Phase, führt zu präziseren und robusten Modellen.

آنی و فانی تمام، معجزہ ہائے ہنر
کار جہان بے ثبات، کار جہان بے ثبات

اوّل و آخر فنا، باطن و ظاہر فنا
نقش کہن ہو کہ نو، منزل آخر فنا

ہے مگر اس نقش میں رنگ ثبات دوام
جس کو کیا ہو کسی مرد خدا نے تمام

مرد خدا کا عمل، عشق سے صاحب فروغ
عشق ہے اصل حیات، موت ہے اس پر حرام

تند و سبک سیر ہے گرچہ زمانے کی رو
عشق خود ایک سیل ہے سیل کو لیتا ہے تھام

عشق کی تقویم میں عصر رواں کے سوا
اور زمانے بھی ہیں جن کا نہیں کوئی نام

Frail and evanescent, all miracles of ingenuity,
Transient, all temporal attainments;
Ephemeral, all worldly accomplishments.

Annihilation is the end of all beginnings;
Annihilation is the end of all ends.
Extinction, the fate of everything;
Hidden or manifest, old or new.

Yet in this very scenario,
indelible is the stamp of permanence
On the deeds of the good and godly.

Deeds of the godly radiate with *Love*,
The essence of life,
which death is forbidden to touch.

Fast and free flows the tide of time,
But *Love* itself is a tide that stems all tides.

In the chronicle of *Love*,
there are times other than the past,
the present and the future;
Times for which no names have yet been coined.

Acknowledgments

I wouldn't have been able to complete this thesis without the support of a lot of sincere people that I have met over the years[1].

First of all, my *Doktormutter*, Prof. Myra Spiliopoulou. During the lengthy mind blocks, her suggestions and ideas always lit up the seemingly dark passages. She always showed me the brighter side of each mistake, which were many and happened quite often.
My reviewers, Prof. Eyke Hüllmeier and João Gama, for accepting to review my thesis. They have also been a source of inspiration, in person and through their works.

My colleagues from KMD, old and new, who have been like an extended family. Thanks to Henning and Rene for helping me settle down during the initial months. Thanks to Max, for German translations, crazy technical ideas (which somehow always seemed implementable), and support. Thanks to Georg and Pawell for their helpful comments regarding the writing of this thesis. Thanks to Fred, Frank, Stefan and Michael for helping me stay connected to the outside world. Thanks to Silke for helping with all the administrative work, travel arrangement and keeping me connected with the reality.

I would also like to thank my friends, Adnan, Ateeq, Azeem, Kirsten, Marcia, Nabeel, Rehan, Sohail, Sümmeyye, and Tahir, who have helped me at one time or the other during this process. My teachers, Sadiq Ali, Sadaf Alvi, Shahid Quereshi, Mazhar Abbas, and Nouman Bhai, also played an important part in getting me where I am.

Without the support of my parents and my siblings, who are the main reason for what I am and none of this would have been possible. Lastly, thanks to Cherish for providing much needed final push for the completion of this thesis.

Before embarking on his journey in search of *glory* and *the great adventure*, Don Quixote deemed it necessary to find his true love. He declared a peasant girl to be his beloved and with whose name he would kiss his sword before every battle. Unlike Don Quixote, I found mine towards the end of my first adventure into the great known. But its only a beginning, there are a lot of adventures to be experienced, there are a lot of battles to be won.

[1] Its hard to name every one. I apologise to all those, who I have failed to mention.

Contents

Contents xi

List of Figures xv

List of Tables xix

List of Algorithms xxi

1 Introduction 1
- 1.1 Thesis Motivation . . . 2
  - 1.1.1 Perennial Objects . . . 3
  - 1.1.2 Challenges . . . 3
- 1.2 List of Scientific Contributions . . . 6
- 1.3 Thesis Outline . . . 7

2 Related Work 9
- 2.1 Multi-Relational Mining . . . 9
  - 2.1.1 Inductive Logic Programming and Terminology . . . 9
  - 2.1.2 ILP and Relational Databases . . . 11
  - 2.1.3 Propositionalisation . . . 12
- 2.2 Stream Mining Paradigm . . . 15
  - 2.2.1 Stream Management . . . 16
  - 2.2.2 Stream Clustering . . . 18
  - 2.2.3 Stream Classification . . . 20
  - 2.2.4 Rule Mining over Streams . . . 22
  - 2.2.5 Relational Mining on Streams . . . 25
- 2.3 Summary . . . 27

3 Managing Perennial Objects 29
- 3.1 Building Perennial Objects . . . 29
- 3.2 Inceremental Propositionalisation Algorithm . . . 32
  - 3.2.1 Memory Management . . . 32
  - 3.2.2 Propagation of Target Identifiers . . . 40
  - 3.2.3 Transforming the Multi-Table Streams . . . 42
  - 3.2.4 Space Adjustment for Nominal Attributes . . . 44
- 3.3 Clustering Perennial Objects . . . 47
- 3.4 Summary . . . 49

4 Evaluating Incremental Propositionalisation 51
4.1 The Datasets . . . 51
4.1.1 Synthetic Dataset: Synthetic Small . . . 51
4.1.2 Synthetic Dataset: Zipf Dataset . . . 51
4.1.3 Synthetic Dataset: Ratings Dataset . . . 52
4.1.4 Movie Lens Dataset . . . 53
4.1.5 Financial Dataset . . . 53
4.1.6 Gazelle Dataset . . . 55
4.2 Evaluating Space Adjustment . . . 56
4.2.1 Evaluation Framework . . . 56
4.2.2 Experimental Results . . . 60
4.3 Comparing Different Caching Strategies . . . 62
4.3.1 Evaluation Framework . . . 62
4.3.2 Experimental Results . . . 64
4.4 Evaluating Incremental Propositionalisation . . . 78
4.4.1 Evaluation Framework . . . 79
4.4.2 Experimental Results . . . 80
4.5 Conclusion . . . 88

5 Classifying Perennial Objects 91
5.1 Tree Induction over a Stream of Perennial Objects . . . 92
5.1.1 Maintaining Sufficient Statistics . . . 94
5.1.2 Dealing with Outdated Perennial Objects . . . 95
5.1.3 Growing a Decision Tree on a Propositionalised Stream 96
5.1.4 Tree Ageing and Generation of Alternate Trees . . . 98
5.2 Experiments . . . 100
5.2.1 Datasets . . . 101
5.2.2 Evaluation Framework . . . 102
5.2.3 Experimental Results . . . 104
5.3 Conclusion . . . 109

6 Using Classification Rules for Mining Perennial Objects 111
6.1 Challenges of Rule Mining over Streams . . . 112
6.2 Overview . . . 113
6.3 Aligning the Ephemeral Streams . . . 114
6.4 Building a Lattice of Classification Rules . . . 115
6.4.1 Update Lattice with new objects . . . 116
6.4.2 Grow Lattice with new rules . . . 116
6.4.3 Update Lattice with old objects . . . 117
6.4.4 Shrink Lattice by removing useless rules . . . 118
6.5 Generating Features from the Rules' Lattice . . . 119

6.6 Enhancing TrIP with Rule-based Features 120
6.7 Experiments 121
6.7.1 Datasets 121
6.7.2 Evaluation Framework 122
6.7.3 Experimental Results 123
6.8 Conclusion 126

7 Multi-Relational Data Generator **129**
7.1 Introduction 129
7.2 Related Work 130
7.2.1 Generating a Static Concept for Recommendations . 130
7.2.2 Generating Social Data with Concept Drift 131
7.2.3 Generating Arbitrary Streams with Concept Drift . . 132
7.3 Multi-Gen for the Generation of Profiles and Transactions . . 134
7.3.1 Generation of Item Profiles and of Items 134
7.3.2 Generation and Transition of User profiles 136
7.3.3 Generation of Users 138
7.3.4 Algorithm: Multi-Gen 139
7.4 Experiments with Multi-Gen 142
7.4.1 Quantitative Results 143
7.4.2 Validation Measures 144
7.4.3 Results 145
7.5 Conclusion 145

8 Conclusion and Future Work **147**
8.1 Summary 147
8.2 Directions for Future Research 149

Bibliography **151**

A Additional Material **165**
A.1 Generating Zipf Distribution using Python 165
A.2 Generating Dataset with Multi-Gen 166
A.3 Experiment with Caching Strategies 166
A.3.1 Financial Dataset with an Adjusted Baseline 166

List of Figures

1.1 Example: A multi-relational schema of an online retail store 2
1.2 Perennial objects come from multiple-interrelated streams . 4

2.1 Example: Target and background predicates with labels . . . 10
2.2 Graphical representation of a multi-relational schema. 11
2.3 Example: Propositionalisation 13
2.4 Different types of time windows. 16

3.1 Example: Incremental Propositionalisation. 31
3.2 Cache maintenance. 34
3.3 Traversing multiple-interrelated streams. 36
3.4 A chain of relationships among different streams. 36
3.5 Transitive references. 37
3.6 Updating the SmallCache. 39
3.7 Conversion into 'star' schema. 41
3.8 Domain of nominal value changes over time. 46
3.9 Nominal feature vectors for hard adjustment. 47

4.1 Schema of the streams in Ratings dataset (generated by the Synthetic Generator). 53
4.2 Statitics for Movie Lens Dataset 54
4.3 Schema of the tables from Financial dataset. 54
4.4 Schema of the tables from Gazelle dataset. 55
4.5 Example: Numerical Value Discretization 56
4.6 Measures for nominal clustering: Continuous 57
4.7 Measures for nominal clustering: Continuous with gap . . . 58
4.8 Measures for nominal clustering: Continuous with var. gap 59
4.9 Measures for nominal clustering: Exponential 60
4.10 Synthetic Generator: Max Measure with n_{bins}=100 varying K. 61
4.11 Synthetic Generator: Jaccard with n_{bins}=100 varying K. . . . 61
4.12 Plots for SmallDataset: $N^{\mathcal{T}}$=5, M=10, β=1 65
4.13 Plots for SmallDataset: $N^{\mathcal{T}}$=5, M=10, β=2 66
4.14 Agg. Plots for SmallDataset: $N^{\mathcal{T}}$=5, M=10 and varying β . . 67
4.15 Agg. Plots for SmallDataset: M=10, β=2 and varying $N^{\mathcal{T}}$ . . 68
4.16 Agg. Plots for SmallDataset: $N^{\mathcal{T}}$=5, β=2 and varying M . . . 69
4.17 Plots for Zipf: $N^{\mathcal{T}}$=100, M=200, β=5 70
4.18 Agg. Plots for Zipf: $N^{\mathcal{T}}$=100, M=200 and varying β 71

4.19 Agg. Plots for Zipf: M=200, β=5 and varying $N^{\mathcal{T}}$ 72
4.20 Agg. Plots for Zipf: $N^{\mathcal{T}}$=100, β=5 and varying M 73
4.21 Plots for Movie Lens: $N^{\mathcal{T}}$=100, M=200, β=1 74
4.22 Plots for Movie Lens: $N^{\mathcal{T}}$=100, M=200, β=2 74
4.23 Plots for Movie Lens: $N^{\mathcal{T}}$=500, M=1000, β=2 75
4.24 Plots for Financial: $N^{\mathcal{T}}$=100, M=200, β=1 76
4.25 Plots for Financial: $N^{\mathcal{T}}$=100, M=200, β=2 77
4.26 Statistics for *Accounts* (perennial) in Financial dataset. 78
4.27 Gazelle Dataset: Jaccard with varying ω and N^j sizes. 81
4.28 Financial Dataset: Entropy for $\omega = 30$ with varying K. 82
4.29 Financial Dataset: Entropy for $K = 9$ $\rho_{min} = 0.7$ with varying ω. 84
4.30 Financial Dataset: Entropy for $K = 9$ with varying ω, ρ_{min}. . 85
4.31 Ratings Dataset: Entropy and Jaccard for K=5, $N^{\mathcal{T}}$=50. . . . 86
4.32 Ratings Dataset: Entropy and Jaccard for K=5, $N^{\mathcal{T}}$=100. . . . 86
4.33 Ratings Dataset: Entropy and Jaccard for K=5, $N^{\mathcal{T}}$=150. . . . 87
4.34 Ratings Dataset: Entropy and Jaccard for K=5, $N^{\mathcal{T}}$=600. . . . 88

5.1 Example: Tree losing support as objects change their label . 98
5.2 Plots for Marriage Dataset with ω=5 and ω=7 104
5.3 Ratings Datset: AUC for the strategies with varying $N^{\mathcal{T}}$ . . . 106
5.4 Performance of different strategies over financial dataset (left) without IDs (right) with IDs 108

6.1 Example: Profiling students based on their grades 111
6.2 Example: Propositionalised students' profiles. 112
6.3 Example: A set of discovered itemsets. 113
6.4 Example: Maximally frequent itemsets and closed itemsets. 113
6.5 AUC plots for Ratings 6500r and Ratings 1500r datasets. . . 123
6.6 AUC and Tree size plots for Drift 150u dataset. 124
6.7 AUC and Tree size plots for Drift 1000u dataset 125
6.8 AUC and Tree size plots for Financial dataset. 126

7.1 Example: A list of item profiles. 136
7.2 Example: A list of user profiles. 137
7.3 Example: Graph for the transition of user profiles. 138
7.4 Components of Multi-Gen. 141
7.5 Statistics for of the MovieLens dataset. 142
7.6 Comparing Multi-Gen dataset and MovieLens dataset 143
7.7 Log-Log plot from the Figure 7.6 143

7.8 Effect of changing various parameters of the Zipf distribution on the user's rating behaviour (left) variable s, (middle) variable v and (right) variable R 144
7.9 Effect of changing various parameters of the Zipf distribution on the how items get rated (left) variable s, (middle) variable v and (right) variable R 144
7.10 Effect of changing the parameter n^u on the no of ratings distribution for user and item 145
7.11 Effect of changing the parameter n^i on the no of ratings distribution for user and item 146

A.1 Plots with an Adjusted Baseline for Financial Dataset: $N^{\mathcal{T}}$=50, M=100, β=1 166
A.2 Plots with an Adjusted Baseline for Financial Dataset: $N^{\mathcal{T}}$=50, M=100, β=2 168
A.3 Plots with an Adjusted Baseline for Financial Dataset: $N^{\mathcal{T}}$=100, M=200, β=2 168
A.4 Plots for Financial Dataset: $N^{\mathcal{T}}$=100, M=200, β=5 169
A.5 Plots for Financial Dataset: $N^{\mathcal{T}}$=150, M=300, β=5 169

List of Tables

3.1 Terms and symbols 30

4.1 Dataset statistics from Financial dataset. 55
4.2 Dataset statistics from Gazelle dataset. 56
4.3 Experimental settings for clustering of nominal values 57
4.4 Description of caching strategies 62
4.5 Parameters for caching strategies 63
4.6 Evaluation measures for caching strategies 64
4.7 Experimental settings for Gazelle Dataset. 79
4.8 Experimental settings for Financial Dataset. 79

5.1 Terms and symbols 92
5.2 Strategies on Marriage Dataset 102
5.3 Strategies on Ratings Dataset 102
5.4 Strategies on Financial dataset 103

6.1 Terms and symbols 114
6.2 Description of Synthetic Datasets 121
6.3 Evaluation Strategies 122
6.4 Experimental settings for synthetic and real datasets. 122

7.1 Parameters of Multi-Gen. 135

A.1 Parameters for various dataset generated from Multi-Gen. . 167

List of Algorithms

1 Cache Update 35
2 Target Id Propagation 40
3 Incremental Propositionalisation 43
4 Nominal Space Adjustment 45
5 Incremental K-Means 48

6 TrIP: Tree Induction over Perennial Objects 93
7 TrIP: Tree Induction over Perennial Objects *(Core)* 93
8 Recursively Forget 95
9 Present Example *(to TrIP)* 96
10 Grow & Adapt Subtree 97

11 Align Ephemeral Streams 116
12 CRMPES: Classification Rules for a stream of Perennial Objects . 117
13 Feature Generation 120
14 CRMPES with TrIP 121

15 Multi-Gen 140

CHAPTER 1
Introduction

In traditional stream mining there is only a single stream of incoming objects. As the objects enter the horizon of observation, information is extracted from them, the model is adapted according to this new information and then the objects are discarded. This learning paradigm has been motivated by the fact that it is practically impossible and pragmatically unnecessary to remember each observed object. However, many application domains contain multiple streams that are interrelated to each other. One of them is the *target* stream, on which learning is performed. For this mining task we introduce the term 'multi-relational stream mining'.

In multi-relational stream mining, the target stream consists of *perennial* objects that appear repeatedly, hence, may not be forgotten. The target stream is fed by *other* streams that may contain *ephemeral* objects (i.e., they contain information of temporary nature, thus can be forgotten once the information has been incorporated in the model) or perennial objects as well.

Perennial objects occur in more applications than one might think at first. Consider a hospital that maintains records for patients with a chronic disease, a company that keeps a customer warehouse, or a group of scientists that study a system of at least two stars, one of which are visible while the others are perceived only through the seemingly aperiodic effects they cause on the plasma, luminosity and trajectory of the visible one. The patients, the customers and the stars (of yet unknown number) are perennial objects, for which activities are recorded. As more activities get recorded, the more is captured in the model of the objects. At the same time, new objects may arrive: new patients with the same disease are registered at the hospital, new customers are recorded in the warehouse, while the observations on further star systems are recorded and can be exploited for model learning. The perennial objects constitute themselves a stream. Capacity limitations may force the deletion of old activities and even of old objects (e.g., customers that have attrited since long or patients that deceased long ago), but their properties must be remembered by the model for the case that similar objects might show up.

As a concrete example, consider an online retail store as shown in Figure 1.1, where customers visit the store's website and buy the products

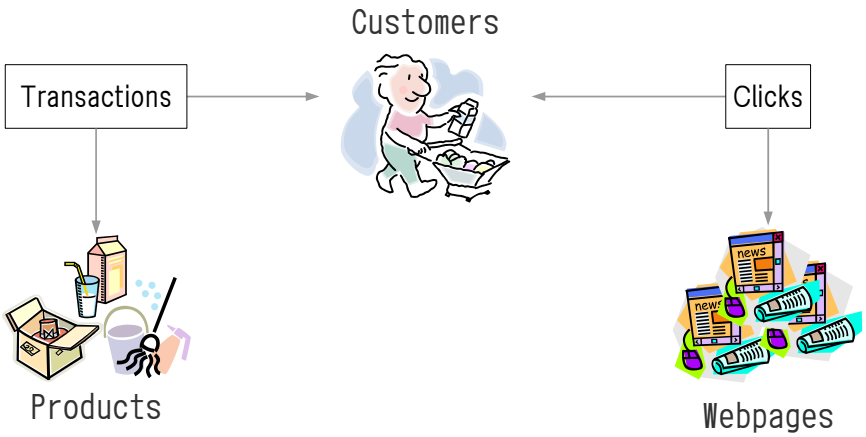

Figure 1.1: Example: A multi-relational schema of an online retail store. The target stream is 'Customers'. The streams 'Products' and 'Webpages' as well as 'Customers', contain perennial objects, whereas the streams 'Transactions' and 'Clicks' contain ephemeral objects

that are offered there. The stream of customers constitute target stream and contains perennial objects: a customer cannot be forgotten or discarded even if she is not currently active. Of the streams that feed the target stream, 'Clicks' and 'Transactions' contain ephemeral objects. These streams only reflect the temporal aspect of the customers and the objects from them (i.e., clicks and transactions) can be discarded once they get old. While the streams 'Products' and 'Webpages' also contain perennial objects. Similarly to the stream of customers, the objects from the streams Products' and 'Webpages' cannot be forgotten as well.

1.1 Thesis Motivation

The focus of this thesis is on mining „perennial objects from the target stream". In the following sections we elaborate on the characteristics of the perennial objects and the implications that they introduce for the algorithms that operate on them.

1.1.1 Perennial Objects

In this section, we give a formal definition of perennial objects and elaborate on their individual aspects, separately, in the subsequent sub-sections.

Perennial objects:

1. are members of a stream $\mathcal{T}$.
2. are referenced from objects of one or more further streams $S_1, \ldots, S_j$.
3. may appear at multiple timepoints. For example, consider a perennial object x and its two instances $x(t), x(t')$. This means that at timepoints t, t', there exist some non empty sets of objects $Y = matches(x, t)$[1], $Y' = matches(x, t')$ that arrive in a stream S_j and references x.
4. do not change their intrinsic properties[2], but their label may change over time.

For the ease of notation, we will use the term *perennial stream* for a stream that contains perennial objects and *ephemeral stream* for a stream that contains ephemeral objects in the remainder of this thesis.

1.1.2 Challenges

Perennial objects are complex Perennial objects are relational by nature: they are composed of simpler ephemeral objects, e.g., customers at an online retail store, patients at a hospital, stars in a constellation, etc. and are associated to further streams that contain objects of different kinds, transactions, symptoms, etc. Depending on the type of relationship, i.e., 1-to-1, 1-to-m or m-to-1, with the other streams, a perennial object can be associated with one or more objects.

Conventional stream mining algorithms are designed to work primarily with a *single* stream of incoming objects (see Figure 1.2). The objects in the stream are mostly ephemeral in nature, i.e., objects arrive, get incorporated into the model and are forgotten as they get outdated. As a

[1] $matches(x, t)$ is a simple query function that returns all the objects that reference x from one or more further streams $S_1, \ldots, S_j$ at timepoint t.

[2] In the customer example, intrinsic properties may be *customer id*, *name* and *gender*. In some applications, the only known intrinsic properties is customer id. Presently, we do not allow volatile properties like *age*; such properties can be often mapped to unchangeable ones like *birth year*.

result, these algorithms do not assume any explicit dependency between the objects that arrive in the stream. For example, two transactions that may belong to a same customer are assumed or get treated as two distinct objects. The dependencies between the objects in the ephemeral stream of transactions get established only when considering the stream of perennial objects - the customers.

For the reasons discussed above, mining these perennial objects presents an enormous challenge because perennial objects violate both of the assumptions, i.e., there exist more than one streams, and objects within any of these streams may be interdependent.

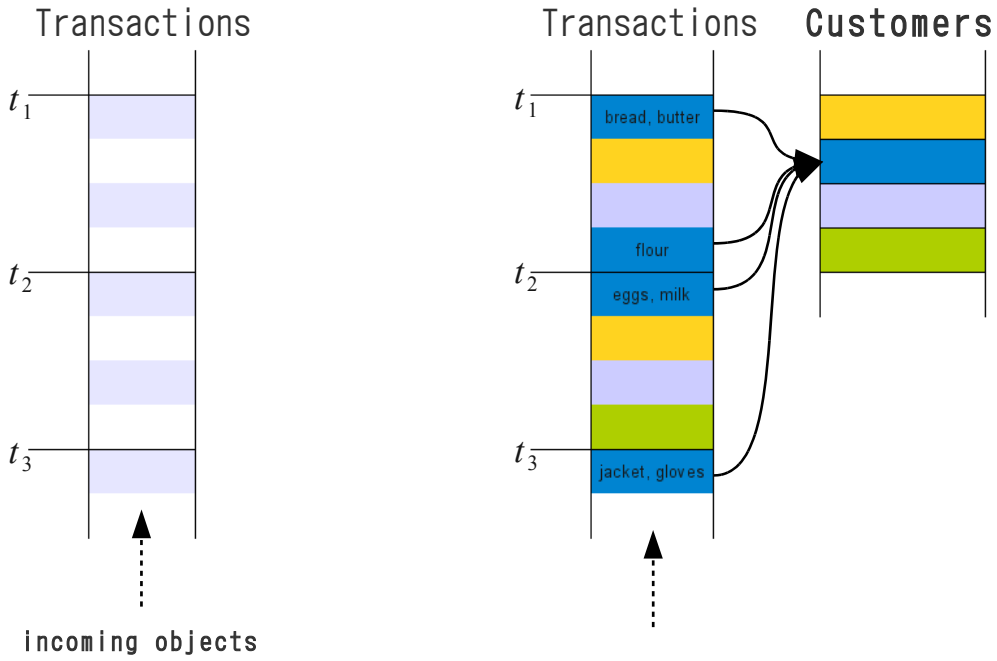

Figure 1.2: Stream of transactions (left) without the perennial stream of customers, i.e., conventional setting (right) with perennial stream of customers. Different colours in the stream of customers depict different customer objects, while the colours in the stream of transactions refer/tag a transaction to the customer that performed that particular transaction.

Perennial objects are dynamic The target stream, which is also a perennial stream, is fed by objects that arrive in the other streams. It is usually a slow stream, i.e., a small number of new objects arrive at each timepoint. The main load, however, comes from the ephemeral stream(s), which are

usually in a 1-to-m relationship with the target stream. As stated earlier, perennial objects from the target stream may appear at multiple timepoints. This means that as the multi-relational stream progresses, new objects that arrive in the ephemeral stream(s), reference the perennial objects from the target stream. These newer objects that arrive in the ephemeral stream make more information available for the perennial objects from the target stream. This new information may either reaffirm properties of the perennial objects or change them. In other words, perennial objects (from the target stream) are dynamic in nature.

For example, consider the *blue* customer in the Figure 1.2 that appears at multiple timepoints. The customer has performed two transactions at timepoint t_1, that are `[bread & butter]` and `[flour]`. As he performs the transaction `[eggs & milk]` at timepoint t_2, there are more data available on him that reaffirm his preference for dairy products. His subsequent transaction, i.e., `[jacket & gloves]` at timepoint t_3 reports a deviation from his current behaviour, i.e., he has also developed a preference for winter products.

We propose a new stream mining framework which assumes a data stream to be generated by a dynamic process and consist of objects that are dynamic and independent of each other.

More formally, consider an instance $x(t)$ of a perennial object $x \in \mathcal{T}$ at timepoint t. The instance $x(t)$ is referenced by a non-empty set of objects $Y = matach(x, t)$ from a stream S_j. At a later timepoint t', when new objects arrive in stream S_j that reference x, the set of objects referencing x would change to $Y' = matches(x, t)$. Hence, the information on x would change its instance to $x(t')$. For any two instances $x(t), x(t')$ of x where, $t \neq t'$, $x(t) \neq x(t')$. For any x with more than one instance, the one with the higher t index replaces the old occurrence, i.e. if $t' > t$, $x(t')$ replaces $x(t)$.

Perennial objects can change their labels In conventional stream classification, each object that arrives in the stream[3] is associated with a *static* class label, i.e., label remains unchanged. For example, a customer would remain trustworthy (or untrustworthy) or an Alzheimer's patient would be exhibiting pacing (or not-pacing). Perennial objects on the other hand, cause a concept drift of a very particular nature. As their properties change over time (c.f. Section 1.1.2), they can also change their class labels. For example, a customer who was trustworthy earlier, may stop being trustworthy if he misses a payment or an Alzheimer's patient may start to exhibit

[3]this stream is ephemeral in nature

pacing from his earlier state of not pacing.

More formally, consider an instance $x(t)$ of a perennial object $x \in \mathcal{T}$ at timepoint t, where x is associated with a label l. As new information appears for x in the other streams at timepoint t', this additional information would transform the old instance $x(t)$ to $x(t')$ and it may come to be associated with a new label l'. For any two occurrences of the labels l, l' at t, t' for x, the one with the higher timepoint replaces the old occurrence, i.e. if $t' > t$, l' replaces l for x.

1.2 List of Scientific Contributions

With this thesis we aim to develop methods that can discover patterns over streams that are connected to each other and contain perennial and ephemeral (transactions, activities, etc.) objects. To achieve this goal, we studied methods from the field of multi-relational mining, conventional stream clustering, classification and rule mining methods. This thesis makes the following scientific contributions:

1. Introduces perennial objects and presents a framework for mining perennial objects (Chapter 3).

2. Presents an *incremental* propositionalisation algorithm (Chapter 3) that combines the stream of perennial objects and the streams of ephemeral objects (transactions, activities and etc.) associated to them into a multi-table stream upon which a learner can be applied.

3. Presents a clustering approach for mining over a propositionalised stream (Chapter 3). The algorithm is a variant of K-Means, extended to deal with perennial objects and able to incorporate any newer objects as they arrive.

4. Presents a new approach for classification over a propositionalised stream of perennial objects (Chapter 5). The classifier is based on CVFDT algorithm of Hulten et al. [Hulten et al., 2001]. It can accommodate the objects as they change their definition and their class labels.

5. Presents extensions to the *incremental* propositionalisation algorithm (Chapter 6). These extensions aim at creating new attributes based on association/classification rules. These attributes capture information otherwise missed by the propositionalisation algorithm.

6. Presents a multi-relational data generator (Chapter 7). We developed the generator for two reasons. First, there is only a limited number of publicly available multi-relational datasets. Second, the generator offers the possibility of making controlled experiments, which is especially useful in our context. It makes it possible to see how the propositionalisation and mining behaves if the underlying concept changes.

1.3 Thesis Outline

In the current chapter, we have provided the problem definition. We have elaborated on what perennial objects are and what are the challenges they pose when mining is to be performed on a stream consisting of perennial objects. We have also discussed the basics that are relevant for the understanding of the contribution of this thesis.

In Chapter 2, we provide a detailed survey of the current state-of-the-art methods in KDD. As multi-relational mining on streams is relatively a new field of study, the related work mostly comes from static relational mining, stream data management and conventional stream mining.

In Chapter 3, we present the incremental propositionalisation algorithm. The approach is based on the *static algorithm* RelAggs by Krögel [Kroegel, 2003]. We accompany this algorithm by a strategy that updates the caches and windows before model adaptation and heuristically minimizes information loss with respect to model learning. Chapter 3 also presents a clustering algorithm based on K-Means that operates over the propositionalised stream. The evaluation of the incremental propositionalisation algorithm is presented in Chapter 4, where we not only evaluate the core algorithm, but also evaluate its constituent parts. Parts of these works were published in the proceedings of the *Scientific and Statistical Database Management Conference 2009* [Siddiqui and Spiliopoulou, 2009a] and *Discovery Science 2009* [Siddiqui and Spiliopoulou, 2009b].

In Chapter 5, we present a classification algorithm for inducing a tree over a stream of perennial objects. It is based on CVFDT and utilises *Hoeffding Bound* for splitting the nodes [Hulten et al., 2001]. Additionally, each node has an age counter associated, which is used to determine its recency and validity. Parts of this work have been published in the proceedings of the *Scientific and Statistical Database Management Conference 2010* [Siddiqui and Spiliopoulou, 2010].

Chapter 6 presents attribute generation approaches to enhance the propositionalisation algorithm. These extensions aim at efficiently han-

dling the attributes whose domain is unknown apriori (i.e., categorical or nominal attributes) and capturing information, i.e., attribute interaction and ordering, which gets lost during the propositionalisation process. Parts of this work have been published in the proceedings of the *International Symposium on Rule Learning 2011* [Siddiqui and Spiliopoulou, 2011].

We continue with the detailed presentation of the multi-relational data generator, which was used in the experiments of Chapter 3, Chapter 5, and Chapter 6. This generator aims to simulate the activities of a user who rates products/items. As the stream progresses, the users' preferences change as well. Parts of this work have been published in the *2nd Workshop on Mining Ubiquitous and Social Environments* at ECML PKDD 2012 [Siddiqui et al., 2011].

In Chapter 8, we conclude with a summary and possible directions for future research.

CHAPTER 2
Related Work

Stream mining over multiple-interrelated streams of perennial objects is a new problem. Most of the related work comes from the domain of learning on multiple tables on static data, as well as mining and management of data in data streams, including windowing, object referencing, stream sampling, etc. We also discuss relevant work from the domain of supervised and unsupervised stream based learning, i.e., clustering, classification, and association and classification rule mining. These work provide the context for the various extensions proposed for our base algorithm.

This chapter is organized as follows. Section 2.1 is on relational mining on static data. Section 2.2 is on stream mining. Under Section 2.2, we discuss management of the streaming data in Section 2.2.1, stream clustering in Section 2.2.2, stream classification in Section 2.2.3, rule mining over streams in Section 2.2.4 and multi-relational stream mining in Section 2.2.5.

2.1 Multi-Relational Mining

Learning over multi-table data requires knowledge on the semantic relationships among the tables. These semantics are typically stored in the database schema. Multi-relational mining methods exploit this schema to deduce the order in which the tables/relations must be processed to build the patterns. Most methods of such type are based on Inductive Logic programming (ILP) [Muggleton and Raedt, 1994].

Since the core of our work emanates from multi-relational data mining, we very briefly explain notations and basics of multi-relational mining, quoting mainly the work of Dzeroski [Dzeroski, 2003], so as to establish a link before we proceed.

2.1.1 Inductive Logic Programming and Terminology

The central concept of ILP is that of *logic programs* that consist of *clauses*. Clauses can be perceived as first-order rules, whose conclusion (or consequent) part is termed as the *head* of a clause and condition (or antecedent) part is termed as the *body* of a clause.

Target *training examples*	*labels*	**Background Knowledge**	
daughter(mary,ann)	+	parent (ann,mary)	female(ann)
daughter(eve,tom)	+	parent (ann,tom)	female(marry)
daughter(tom,ann)	-	parent (tom,eve)	female(eve)
daughter(eve,ann)	-	parent (tom,ian)	
daughter(ann,tom)	-		

Figure 2.1: ILP Example: Target and background predicates with labels (taken from Dzeroski [2003])

For example, consider the following clause:

$$\texttt{son(X,Y)} \leftarrow \texttt{parent(Y,X)} \wedge \texttt{male(X)}$$

The predicate `son(X,Y)` is the head while, the body is made of two predicates, i.e., `parent(Y,X)` and `male(X)` [Dzeroski, 2003]. It reads, if `X` is male (`male(X)`) *and* `X` has a parent `Y` (`parent(Y,X)`), then `X` is a son of `Y`.

ILP learning starts with an extensional definition of the so-called *target predicate* (that forms the head of the clause) and learns a more concise intentional definition (body or the conditional part), exploiting given background knowledge in the form of other predicates. In the example given in Figure 2.1, the task is to learn the intentional definition of the target concept `daughter(X,Y)` (it reads, a person `X` is daughter of a person `Y`), in terms of background examples. ILP would take all of these examples and by utilising the labels given for each of the target examples, it would come up with the following clause:

$$\texttt{daughter(X,Y)} \leftarrow \texttt{parent(Y,X)} \wedge \texttt{female(X)}$$

Some of the methods that use ILP include multi-relational association rules [Dehaspe and Toivonen, 2001, 1999] multi-relational decision trees [Blockeel and Raedt, 1998; Kramer and Widmer, 2001] and distance measures [Emde and Wettschereck, 1996] that can be used for clustering [Kirsten et al., 2001].

For the purpose of mining over a perennial stream, multi-relational learning techniques have two caveats. First, they are computationally very expensive. Because of multiple-joins of varying length that exist between different tables, the search space is exponential and multi-relational mining algorithms suffer in terms of efficiency. It is often the case that even testing a given relational pattern for validity is computationally expensive

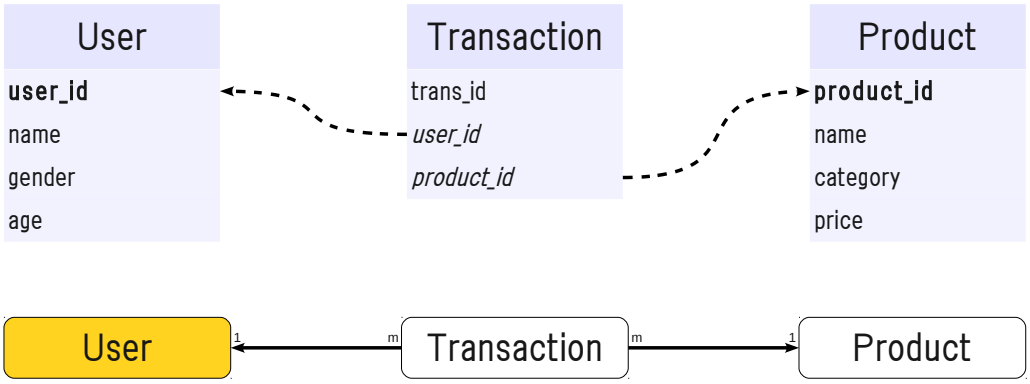

Figure 2.2: Graphical representation of multi-relational schema. (Top) a sample database schema comprising of three tables. The **primary keys** are in bold face, while *foreign keys* are italicised. The edges are drawn from foreign key to primary key. (Bottom) Graph based representation of the sample schema. The target stream 'User' is depicted with a shaded box.

[Dzeroski, 2003]. Second, multi-relational learning methods are dedicated methods. Since there is a wealth of stream mining algorithms, it seems reasonable to design a methodology that can exploit any of them rather than devising new dedicated stream mining methods.

2.1.2 ILP and Relational Databases

In the context of relational databases, a database table corresponds to a *predicate* in logic programming, where the attributes of the tables are the *arguments* of the predicate and the tuples in the table are the *ground facts* or training examples. Accordingly, we have the notion of *target predicate* that corresponds to the *target table*. In the context of this thesis, the stream of perennial objects, which is central to the learning is designated as target stream. The notion of *background predicate* is synonymous with the further streams that fed the target stream.

In this thesis, the schema of multi-relational streams is depicted using graphs. Each node in the graph represents a database relation or a stream. The edges between the nodes represent the relationships that exist between the streams. Relationships in conventional databases are expressed through foreign key relationships. The edges point from the streams that contain the foreign keys to the streams that contain the primary keys. The node that depicts the target stream is shaded, whereas nodes that depict other streams are shown as transparent boxes. An example of

this convention is shown in Figure 2.2[1].

2.1.3 Propositionalisation

The *propositionalisation* methods described below alleviate this second caveat for static data. They exploit the schema to transform the multi-table data into a single table which preserves the original semantics *and* is appropriate for conventional data mining.

The propositionalisation methods use the background knowledge predicates to derive a set of clauses for the target predicate [Dzeroski and Lavrač, 1993; Lavrač and Flach, 2001; Knobbe et al., 2001; Kroegel, 2003; Perlich and Provost, 2006; Anderson and Pfahringer, 2009; Alfred, 2011]. The target predicate is the head of each clause, while the body contains background predicates that must conform to a given declarative or language bias. The clause set forms the basis for propositionalisation.

In the context of databases, a tuple from the target table $\mathcal{T}$ must be expanded with all relevant information before being processed. The main challenge to be solved by propositionalisation methods is that data spanned across multiple tables are correlated and may stand in 1-to-n or m-to-n relationships. Thus, the intuitive approach of joining all tables into a single one is inappropriate. For example, consider the data in Fig. 2.3(a) and assume that `Customer` is the target table: a customer tuple must be joined with all transactions of this customer. There are three customers, who have performed four, two and one transactions. If a natural join is computed over this schema, the resulting table will have 7 tuples (see Fig. 2.3(b)). Data mining algorithms assume that the rows of a table are independent of each other. This assumption gets violated after a join is computed over the tables. There are tuples in the new table that are not independent of each others as they belong to the same customers. Propositionalisation methods perform more than a simple joined based transformation. They perform join and aggregation in order to transform data from multiple tables data into a single table by turning rows to columns (see Fig. 2.3(c)).

Our work is based on the static multi-table propositionalisation method RelAggs [Kroegel, 2003]. In the subsequent sections, first, we briefly explain RelAggs and then continue our discussion on other propositionalisation methods.

[1]This convention has been adapted from RelAggsby Kroegel [Kroegel, 2003], which is the basis propositionalisation algorithm that we build upon.

Original Schema

Transaction

TID	CID	PID
1	2	1
2	2	1
3	2	1
4	2	2
5	1	2
6	1	1
7	3	1

Customer

CID	Name	Married	Income
1	David	S	20000
2	Tom	M	65000
3	Mark	S	45000

Product

PID	Price	Category
1	100	Book
2	50	DVD

Natural Join

TID	CID	NAME	Married	Income	PID	Price	Category
1	2	Tom	M	65000	1	100	Book
2	2	Tom	M	65000	1	100	Book
3	2	Tom	M	65000	1	100	Book
4	2	Tom	M	65000	2	50	DVD
5	1	David	S	20000	2	50	DVD
6	1	David	S	20000	1	100	Book
7	3	Mark	S	45000	1	100	Book

Propositionalised Table

CID	NAME	Married	Income	SUM_P	AVG_P	MIN_P	MAX_P	Count_C_Book	Count_C_DVD
2	Tom	M	65000	350	87.5	50	100	3	1
1	David	S	20000	150	75	50	100	1	1
3	Mark	S	45000	100	100	100	100	1	0

Figure 2.3: First tuples of (a) a multi-table stream on customers, transactions and products, (b) natural join of the `Customer` data with `Transaction` and `Product` information and (c) propositionalised version of the same target `Customer`

RelAggs for the Propositionalisation of Static Data

The propositionalisation algorithm RelAggs starts similarly to ILP methods. It derives a set of clauses from the database schema using foreign key relationships [Krogel and Wrobel, 2002] between the target table and background knowledge table.

A natural join covering all the tables in each remaining clause is separately computed. If the table S that feeds the target table $\mathcal{T}$ stands in 1-to-1 or 1-to-m relationship with the target table $\mathcal{T}$, its tuples are simply concatenated to the target tuples with help of the foreign key. If S stands in m-to-1 relationship with $\mathcal{T}$, then for each attribute $A \in S$, a group of new features $\varphi(A)$[2] are created to accommodate the aggregated contents of S. If A is numeric, it is expanded into four attributes that accommodate the *average*, *sum*, *maximum* and *minimum* of values seen thus far. If A is nominal, it requires much more space, since each distinct value gets converted in a new attribute, which stores the count for the number of times the respective nominal value was observed. It also generates an additional feature *count* once for each table. This feature records the number of tuples on which the aggregation was performed.

[2] $\varphi(A)$ is a function defined in [Kroegel, 2003] that generates new aggregate features for the attribute A

Other Propositionalisation Approaches

The early propositionalisation approach LINUS was developed for logic programming and was a part of an ILP system [Dzeroski and Lavrač, 1993]. Later it was extended to include the handling of determinate clauses [Lavrač and Flach, 2001]. The methods of Knobbe et al. [Knobbe et al., 2001], Perlich and Provost [Perlich and Provost, 2003, 2006] and as well of Kroegel [Kroegel, 2003] are examples of propositionalisation approaches on databases. These approaches transform the tables from a relational database into a single table with the help of the built-in functionalities of databases for joining the relation and column aggregation.

Later propositionalisation approaches [Alfred, 2011; Anderson and Pfahringer, 2008, 2007, 2009; Alfred, 2008a; Lachiche, 2005; Alfred, 2009; Alfred and Kazakov, 2007; Alfred, 2008b] adopt a more complex and sophisticated approach towards relational transformation. Anderson et al. [Anderson and Pfahringer, 2007, 2008, 2009] make use of *Relational Random Rules* (RRR) to propositionalise the relational data. Their method for the rule discovery is similar to FOIL[3] as it also finds first-order rules. Unlike FOIL, which is exhaustive in terms rule discovery, RRR adds the conjunctive literals randomly. To validate the quality and the coverage of the discovered rules they use *stochastic discrimination* [Kleinberg, 2000]. This rule set is then used as the basis for their propositionalisation method RRR-P. RRR-P first checks the rule set for the sufficiency of coverage for both the negative and the positive classes, if not, it calls RRR to generate further rules. RRR-P generates boolean attributes that are based on the rules from the rule set. Each of the generated attribute records `True`, if the respective rule is true for a given target example or `False`, otherwise.

The methods of Alfred et al. [Alfred and Kazakov, 2007; Alfred, 2008b,a, 2011, 2009] are similar to the RRR-P method in terms that the propositionalisation process is driven by *pattern-based* aggregation. However, their notion of pattern is more elaborate than the one used by Anderson et al. [Anderson and Pfahringer, 2007, 2008]. Their core method is called *Dynamic Aggregation of Relational Attributes* (DARA). The patterns that they discover are based on what they term as *cross-aggregation*. The cross-aggregation is a combination of the *vertical aggregation*[4] and the *horizontal aggregation*[5]. Their method of finding patterns is explosive in terms

[3]FOIL is method from ILP for Rule Generation

[4]A vertical aggregation is similar to the column-based aggregation as used in RelAggs[Kroegel, 2003] and Relational Concept Classes [Knobbe et al., 2001].

[5]A horizontal aggregation is similar to the rule-based aggregation used in RRR [Anderson and Pfahringer, 2007, 2008, 2009].

of number of seed attributes that it generates. To avoid the curse of dimensionality, they compute the weight of each pattern-based attribute. This computation is straight forward and is based on pf-irf score, where pf is the pattern frequency and irf is the inverse record frequency. The semantics of pf-irf are similar to that of tf-idf scores commonly used in the field of information retrieval for document clustering.

In general, the propositionalisation overhead grows exponentially with the number of attributes that are created to accommodate the rows of tables into a single propositionalised table [Dzeroski and Lavrač, 1993; Lavrač and Flach, 2001; Knobbe et al., 2001; Alfred, 2011; Lachiche, 2005; Alfred, 2009; Alfred and Kazakov, 2007; Alfred, 2008b]. The method of Anderson et al. [Anderson and Pfahringer, 2008, 2007, 2009; Alfred, 2008a] uses first-order rules whose expressiveness enables it to limit the number of generated attributes. However, the complexity required for the validation of the discovered rules is exponential (noted in [Dzeroski, 2003] as well) and is a significant drawback. On the other hand, the algorithm RelAggs [Kroegel, 2003] achieves propositionalisation that is linear to the number of generated attributes. This is important for our stream clustering task, so we opt for RelAggs as the basis of our stream propositionalisation.

2.2 Stream Mining Paradigm

Streaming data or *data streams* are not a new phenomenon but we have only recently started to investigate them. Data streams are often characterised by high arrival rate, massive size and the notion that they are dynamic, e.g., in domains such as wireless network, sensor networks, science and finance where data is produced continuously. Algorithms that operate on data streams have to take into account that 1) data arrive continuously, 2) data is open ended and potentially infinite and cannot be accommodated in the main memory, 3) data objects arrive in a specific order (with system having no control to alter the ordering), and 4) the distribution function that generates data is dynamic and can change over time.

These requirements put forward massive challenges for the algorithms that operate over such data. Streaming algorithms must learn the model from the incoming data and update it in an online fashion. Many algorithms tackle the problem of open-ended nature of data by storing a fraction of (most recent) data in memory buffers or the so called *time windows*. Depending on the nature of the data, time windows can be *landmark win-*

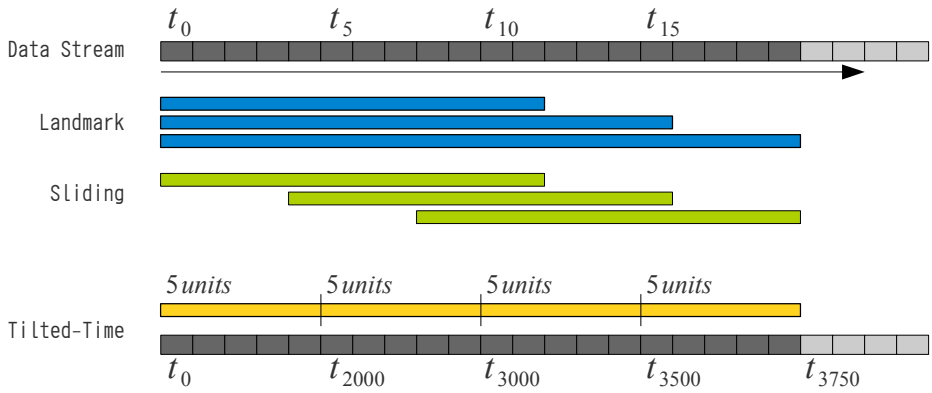

Figure 2.4: Different types of time windows.

dows, *sliding windows*[6] or *tilted-time windows* (see Figure 2.4). A *landmark window* defined over a stream of incoming data is accumulative, i.e., as new data arrive, it is added to the window and the size of the window grows. A *sliding window* holds a limited amount of data. As new data arrive, it gets added to the window and at the same time, the old data is forgotten. A *tilted-time windows* stores the complete data at multiple level of granularity. The most recent data is stored at the highest resolution, i.e., every point is stored, while as the data gets older, it is stored in increasingly concise summaries.

2.2.1 Stream Management

Sampling and window size adaptation are investigated in the context of concept drift detection and adaptation. The simplest approach is to use a sliding window of fixed length w. However, adaptation is needed if the speed of the stream changes or other forms of concept drift occur. The methods of [Gama et al., 2004; Widmer and Kubat, 1996] discard individual objects on the basis of their usefulness in terms of learning the concept. ADWIN by Bifet et al. uses the notion of adaptive windowing where the window grows during seemingly static scenarios and shrinks when the data start to change [Bifet and Gavaldà, 2007]. Unlike other approaches it is assumption-free in the sense that it adapts to the environment. The only parameter is the confidence bound δ that indicates how confident users want to be in the output provided by the algorithm.

[6] can be of a fixed size or can change depending on the nature of the distribution in the data

Most literature on sampling deals with sampling over static data and individual streams, which is different from the problem that we address. Chuang et al. [Chuang et al., 2009] point out that sampling over streams has its basis on the two sampling methods, *reservoir sampling* [Vitter, 1985] and *sequential random sampling* [Vitter, 1987]. Chaudhuri et al. point out that it is inefficient to perform sampling on the output of query result [Chaudhuri et al., 1999]. For a given query tree T, their method does not sample at the root but rather pushes the sampling operator down to the leaves.

Babcock et al.'s sampling algorithm *priority sampling* has been designed for the sliding window and generates the sample sequentially [Babcock et al., 2002]. However, the sampling is random and is done *after* the join and on the result of the join between the two streams than over streams individually. Johnson et al. propose a new *stream sample operator* that provides approximate answers [Johnson et al., 2005]. Or in other words, their focus is more on aggregation aspect of sampling. Chaung et al. proposes a novel sampling algorithm, *feature-preserved sampling* that can maintain a quality sample, sequentially, for a given sample rate p [Chuang et al., 2009]. They do so by preserving the distribution characteristics for each attribute in the sample, as in the population.

Stream join algorithms incorporate a mechanism that discards old tuples. `PROB` retains those tuples that are referenced many times [Das et al., 2003], while `LIFE` discards the oldest ones [Das et al., 2003]. `AGE` assumes that the benefit of storing a tuple is a function of its age, modelled as the time that it remains within the sliding window [Srivastava and Widom, 2004]. Law and Zaniolo presents an algorithm for load shedding of tuples across a join involving more than two streams [Law and Zaniolo, 2007]. Xie et al. distinguish between joins involving two streams and those involving a stream and a static table [Xie et al., 2005]. The first operation (i.e., a join involing two streams) is termed *joining*, while the second one (i.e., a join involving a stream and a static table) *caching*.

Our window management algorithm presented in [Siddiqui and Spiliopoulou, 2009a] borrows the formulation from Xie et al. [Xie et al., 2005]. Our method distinguishes between tuples that can be forgotten and those that cannot be forgotten. For the ones that can be forgotten it uses a sliding window and for the ones that cannot be forgotten it uses caching as used by Xie et al. [Xie et al., 2005]. It can perform load shedding across multiple streams by utilising transitive references [Siddiqui and Spiliopoulou, 2009a].

2.2.2 Stream Clustering

Clustering on data streams is an active research area. Most stream based clustering algorithms process and then discard tuples. The method of Bradley et al. [Bradley et al., 1998] is based on identifying regions that must be maintained in memory and regions that are discardable. The algorithm maintains a buffer of incoming tuples. K-Means[7] is applied over the filled buffer until convergence. The discardable regions are identified and are replaced with a compressed tuple placed at the mean of a discardable region and is weighted with the cardinality of the discarded region. The acquired space in the buffer is filled with next batch of tuples.

The clustering method presented by Farnstorm et al. [Farnstrom et al., 2000] is based on the clustering method of [Bradley et al., 1998]. They point out that data compression is an expensive strategy and does not necessarily improve clustering. They remove all the expensive computations needed to identify the regions that can be discarded and cannot be discarded. Instead, their method treats each cluster as a discardable region. Similarly to Bradley et al., each discarded region is replaced with the region's mean and weighted with the cardinality of the discarded region.

Callaghan et al. maintain a buffer to store points into batches of m points [O'Callaghan et al., 2001]. After the clustering on the buffer, it retains the K centres along with statistics and discards the contents of the buffer. The buffer is refilled with new data points and the process is repeated with all the points. Guha et al. maintain at each moment the m most recent tuples and K medians that stand for $K \times m$ tuples seen in the past [Guha et al., 2003].

Aggarwal et al. [Aggarwal et al., 2003] pointed out that one pass algorithm over a data stream can be dominated out dated data. They go on to suggest that the exploration of the stream over different time windows can provide the users with a much deeper understanding. They present a micro-clustering approach, CluStream, that maintains clustering in multiple snapshots of pyramidal time frame. TECNO-Stream method of Nasraoui et al. [Nasraoui et al., 2003] for clustering streaming is inspired from the working of human immune systems. At the core of their method are the DWB-cells, which are very similar to the micro-clusters of Aggarwal et al. [Aggarwal et al., 2003]. DWB-cells operate in an online step to approximate the incoming data. Similarly to Aggarwal et al. [Aggarwal et al., 2003], the final clustering model is generated by applying a K-means algo-

[7] For the initial timepoint t_0, k centres are randomly initialised. At every subsequent timepoint t_i, k centres are initialised with the centres from the previous timepoint, i.e., t_{i-1}

rithm over these DWB-cells. DenStream method of Cao et al. [Cao et al., 2006] also uses the concept of micro-clusters to approximate the fast streaming data. However, their micro-clustering structure is used as the basis to learn a density-based clustering model [Ester et al., 1996].

All previous works learn a model on a single stream of objects that are seen once and then forgotten. There are also studies on combining multiple streams to learn a model, as discussed below. However, they still consider objects (texts of news for example) that are seen once and then are forgotten, rather than multiple streams of ephemeral objects that *add* information to perennial objects.

Beringer and Hüllermeier present an elaborate method that aims at clustering multiple data streams [Beringer and Huellermeier, 2006]: for each individual stream they store data points into m blocks of v points each that define a window w of $m \times v$ most recent tuples. Before launching k-means based incremental clustering, a pre-processing step incrementally computes the distance between the streams using Discrete Fourier Transformation. To track data evolution they also include a fuzzy-based approach for the dynamic updating of the optimal number of clusters.

The framework Clustering on Demand (COD) by Dai et al. [Dai et al., 2006] clusters multiple data streams by performing an online and offline step. The online step is used for statistics collection and summary generation from the raw incoming data. They describe two methods for summary generation; a wavelet-based and regression-based. The summaries that they maintain are in pyramidal representation, where each level denotes a different level of approximation. The offline step operates over these summaries for the generation of a clustering model. Their pyramidal summary structure, makes it possible to learn a clustering model with adaptive window sizes.

Another method for clustering over multiple streams is COMET-CORE by Yeh et al. [Yeh et al., 2007]. Like the methods of Beringer and Hüllermeier [Beringer and Huellermeier, 2006], and Dai et al. [Dai et al., 2006], COMET-CORE also stores a summarised representation of the incoming streams by using *liner approximation*[8]. They also use the linear approximation method to identify events that occur in the individual streams. Only on the basis of these events they adjust the clustering model by splitting and merging clusters.

Differently from the methods discussed thus far, the method of Wang et al. [Wang et al., 2000] proposes a probabilistic framework to mine similar topics from multiple *asynchronous* text streams. The main idea in their

[8] „The incoming streams are smoothed as sequences of end points" [Yeh et al., 2007]

approach is to utilise the information on a certain topic from the multiple text streams to build a combined reinforced model.

The methods that operate on multiple streams are not appropriate for a stream of perennial objects, because they assume a fixed schema. Mining methods that do deal with relational data are discussed in Section 2.2.5.

2.2.3 Stream Classification

One of the first works to address the issue of incremental tree induction was the work of Schlimmer and Granger [Schlimmer and Granger, 1986]. Their proposed method ID4 is an extension to the ID3 method of Quinlan [Quinlan, 1983]. It processes each example (tuple/object with its label) as it arrives and learns the tree incrementally. All statistics needed to compute the entropy of each attribute and choose the one with lowest entropy, X_a, are stored at the nodes. At a later timepoint, if X_a no longer has the lowest entropy, it is replaced by the one with lowest entropy. In doing so ID4 also discards all the sub-trees below the node X_a. If the choice of decision attributes changes often during training, then sub-trees will be discarded repeatedly. This makes ID4 sensitive to the ordering of incoming examples, rendering certain concepts unlearnable by the algorithm.

Utgoff proposed two methods for incremental construction of decision trees, ID5 [Utgoff, 1988] and ID5R [Utgoff, 1989]. The basic idea of keeping the statistics at the nodes is similar to [Schlimmer and Granger, 1986]. However, instead of keeping concise statistics, they store complete examples. The monitoring of entropy $E(X_a)$ for a candidate attribute X_a is as in [Schlimmer and Granger, 1986]. However, when another attribute X_b exhibits the lowest entropy, ID5 re-structures the tree only to ensure that X_b becomes the root; unlike the algorithm in [Schlimmer and Granger, 1986], it keeps the original sub-trees without changing them. ID5R [Utgoff, 1989] shifts X_b at the root, but then proceeds recursively with re-structuring below the root node. Due to this, ID5 is unable to guarantee that the tree would be similar to that of an ID3 algorithm given the same examples, while ID5R does.

All of the above algorithms induce a decision tree by a greedy search mechanism that requires restructuring the tree, if the selection for a specific split attribute X_a (the "split decision") needs to be revised. The method of Gratch induces a decision that is significantly different from the greedy approaches [Gratch, 1996]. Gratch notes that the optimal split decision cannot be achieved without a finite sample, so the proposed method " selects an attribute that is within ϵ of the best with probability $1 - \delta$, tak-

ing as many examples as are sufficient to ensure a decision of this [given] quality."

More recently, Domingos and Hulten presented their incremental decision tree induction method VFDT [Domingos and Hulten, 2000]. They note (with Catlett [Catlett, 1991]) that a small subset of training examples is sufficient to select the best split attribute for a given node. They use the so-called *Hoeffding bound* to determine the number of training examples that are required. More formally, consider a real-valued random variable r whose range is R. Suppose we have made n independent observations of this variable and computed their mean $\bar{r}$. The Hoeffding bound states that, with probability $1-\delta$, the true mean of r is in $[\bar{r}-\epsilon, \bar{r}+\epsilon]$, where

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{2.1}$$

At each timepoint, VFDT applies the Hoeffding bound upon the difference ΔG between the information gain of the best split attribute $G(X_a)$ and of the second best attribute $G(X_b)$. In particular, assume that the two attributes deliver (asymptotically) the same gain, i.e., they are interchangeable. The true mean of their difference would then be zero and ΔG would be less than $0+\epsilon$, where ϵ is computed as in Eq.2.1 with n being the number of examples seen thus far. Hence, if ΔG is found to be larger than $0+\epsilon$, then the best split attribute X_a is significantly better than the second one, and VFDT uses it as a new leaf node, growing the tree upon it. If ΔG is less than ϵ, then VFDT postpones the split until a best split is found that is significantly better than the second best.

Hulten et al. extended the VFDT to deal with concept drift: the CVFDT updates the tree as soon as the learned concept starts to change [Hulten et al., 2001]. They also introduce a notion of window that stores only the most recent objects and removes the old ones, i.e., those that fall out of the window.

The algorithm VFDT$_C$ of Gama et al. makes use of a minimal set of statistics (they call them *sufficient statistics*[9]) calculated at the leaves [Gama et al., 2003]. VFDT$_C$ is an extension of the original VFDT [Domingos and Hulten, 2000] that employs Naïve Bayes at each leaf node to assign the node's members to the classes with help of these statistics. They show that they achieve better performance and can also recognize concept drift

[9]Sufficient statistics store the information about the distribution of class labels with respect to each attribute-value pairs. This information is maintained for each node in the tree. For example, (att=**forecast**, val=*rain*) $\rightarrow$ [+ve=90, −ve=10]. It says, that of the 100 examples that satisfy (**forecast**=*rain*), 90 belong to the positive class while 10 belong to the negative class.

by detecting performance degradation. Their proposed system can also handle numerical attributes.

Methods that learn a classifier on evolving relational data, like the SRPT algorithm of McGovern et al. McGovern et al. [2008], are presented in Section 2.2.5.

All aforementioned algorithms operate on a single stream of data and under a fixed schema. As we explain in Chapter 1, a multi-relational stream of perennial objects implies combining the individual streams at each timepoint; this combination of streams incurs a change in the schema. Furthermore, the stream classifiers assume that objects are seen only once, whereas perennial objects are seen more than once and experience changes - including changes in their label.

2.2.4 Rule Mining over Streams

Conventional classification rules have the form

$$A \ldots B \rightarrow l$$

where the antecedent $A, \ldots, B$ are any combination of (attribute,value)-pairs and the consequent part l is a class label.

Among the first works on association rules for classification were CBA [Liu et al., 1998] and ADT [Wang et al., 2000]. These methods discover rules and then apply a selection procedure to choose the most predictive ones.

Rule mining algorithms, in general, suffer from the problem of combinatorial explosion of the discovered patterns. The main challenge is how to effectively mine, store and find interesting rules. Data streams with their inherent properties and requirements bring newer challenges in the rule mining domain. Yu and Chi [Yu and Chi, 2009] in their survey on rule mining algorithms, distinguish between *exact* [Chi et al., 2004; Veloso et al., 2002] and *approximate* [Hidber, 1999] rule miners over streams. In the remainder of this sub-section, we discuss the relevant work from rule mining with respect to above categorisation, i.e., of Yu and Chi [Yu and Chi, 2009].

The methods *IRIL* [Aydin and Güvenir, 2004] and *ICRIL* [Aydin and Güvenir, 2005] by Aydin and Güvenir, deal with classification rules over streaming data. The objective of the proposed algorithms is to model interestingness of the rules and not to build a classifier. The notion of interestingness used by the algorithm is subjective and requires user interaction. The algorithms assign an application-specific label to the rules, e.g., they label rules on whether they are interesting to the user.

The work of Gupta et al. [Gupta et al., 2005] uses concept analysis for discovering classification rules incrementally. The rules are defined as concepts and are stored in one lattice per class. Each concept has an *extent* A (number of tuples supporting it) and an *intent* B (attributes common to A). For discovering rules, the algorithm simply traverses the lattice and returns the rules with the label as consequent. As new tuples are presented to lattice, it updates the existing concept in the lattice and if some concept is missing, it is added to lattice. The algorithm can only handle binary attributes. Moreover, it is not entirely clear how would the algorithm behave if some concept exists in more than one lattice, i.e., in lattice of other classes.

The method of Ferrer et al. FACIL [Ferrer-Troyano et al., 2005] is a classification rule mining algorithm for numerical data streams that focuses on processing border examples. FACIL iterates over the examples multiple times to discover rules by storing border-examples. As new examples are added, FACIL computes the rule purity (i.e., the extent to which a rules contains examples from a single class). When the purity threshold for a certain rule goes above a user-defined threshold κ, the examples stored inside are used to split the rule into purer rules. As new test examples arrives, they get classified by consistent rules[10] on the basis of coverage, whereas inconsistent rules classify the examples using the nearest neighbour classification.

Methods of Cheun et al. *FUP* [Cheung et al., 1996] and *FUP*$_2$ [Cheung et al., 1997] are batch update algorithms. *FUP* only considers insertion of new transactions while FUP_2 considers deletion as well. The algorithms are similar to the Apriori algorithm of Agrawal and Srikant [Agrawal and Srikant, 1994]. FUP and FUP_2 discover itemsets L over a window of incoming transactions D. As a new transactions batch D^+ arrives, it incrementally updates L over the new window $D \cup D^+$. Only *FUP*$_2$ considers the deletion.

The algorithm of Thomas et al. is similar to *FUP*$_2$ [Thomas et al., 1997]. Additional to the set of frequent itemsets L, it also keeps a so called *negative border* $NBd(L)$. A *negative border* stores all those itemset that were part of candidates for frequent itemset but their support were lower than the threshold. As a result, whenever an itemset from negative border becomes frequent, unlike FUP_2, the algorithm of Thomas et al. has to mine the updated window, i.e., $D \cup D^+$, at most once. Thus, achieving a performance improvement at the cost of slightly more memory.

ZigZag algorithm of Veloso et al. [Veloso et al., 2002] uses D^+ to up-

[10] the ones that contain examples from a single class only

date the support of the itemsets but not for finding new frequent itemsets. To avoid combinatorial explosion, it maintains only the *maximally frequent itemsets* which results in information loss and a database scan is required to discover the support of all frequent itemsets.

Moment maintains only the *closed frequent itemsets* over a sliding window [Chi et al., 2006]; for these itemsets there is no information loss. The authors use a tree-based data structure, *closed enumeration tree* (CET), to store selected itemsets over the sliding window. The selected itemsets must either be frequent themselves or their parents must be frequent. The nodes inside the tree are characterised as *infrequent*, *unpromising*, *intermediate* and *closed* based on their support count and if there support count is same as their child or parent. They define a boundary between closed nodes and rest of the nodes. With the arrival of transactions, the support counts of the nodes get updated and the boundary, defined between the closed nodes and others, gets updated. The new itemsets are captured through these boundary movements.

Unlike the methods that have been discussed previously, the method of Hidber, CARMA [Hidber, 1999], is a one-pass approximate algorithm that uses a lattice L, to store the frequent itemsets. As a new transaction x arrives, CARMA adjusts the support of all the relevant itemsets in L. If some itemset u appears x but is not in the lattice, i.e., $u \notin L$, u is added to the lattice if and only if all the subsets of u are already present in the lattice and meet the minimum support requirement. Since a newly created itemset may have missed some previously inserted tuples (an itemset cannot be added before its subset), CARMA keeps a count on the number of transactions that might have been missed and maintains support intervals for each itemset in the form $[s_l, s_u]$. If it is possible to retrieve the transactions again, CARMA can calculate the exact support of u.

The Very Fast Decision Rules (VFDR) of Gama and Kosina [Gama and Kosina, 2011] has some conceptual similarities to our classification rule miner CRMPES (Chapter 6, [Siddiqui and Spiliopoulou, 2011]), which we use as preprocessor to our relational stream classifier TrIP(Chapter 5). This is a non-adaptive, single-pass method that is inspired from the VFDT method of Domingos and Hulten [Domingos and Hulten, 2000]. Similarly to VFDT, it also maintains information in the form of sufficient statistics (cf. Section 2.2.3). Similarly to VFDT whose selection for the best attribute for splitting a leaf node is based on the hoeffding bound [Catlett, 1991], VFDR's decision for choosing the best literal for rule expansion is also based on the hoeffding bound.

Our method for discovering the classification rules for perennial streams, CRMPES (cf. Chapter 6, [Siddiqui and Spiliopoulou, 2011]) is a

single-pass algorithm that is inspired from CARMA [Hidber, 1999]. It stores the rules in the form of a concept lattice and each element in the lattice stores the sufficient statistics for lattice expansion [Domingos and Hulten, 2000]. For expanding the lattice with more rules it also makes use of the hoeffding bounding. To manage the number of discovered rules, it stores only the closed itemsets as in Moment [Chi et al., 2006].

2.2.5 Relational Mining on Streams

The focus in this section is specifically on the multi-relational methods for streams. The discussion on research advances mostly focus on classification, summarisation and frequent pattern mining.

Relevant to our idea of stream mining upon multi-relational objects is the SRPT algorithm proposed by McGovern et al. [McGovern et al., 2008]: the algorithm operates upon spatio-temporal data of numerical nature (e.g., time-series of meteorological phenomena like wind up-drafts), accompanied by attributes that summarise the time-series or check conditions upon them (e.g., whether the observed mean of a specific time-series exceeds a specific threshold or whether some explicitly defined event has been observed in the most recent timepoints)[11]. The SRPT induces a tree using the greedy heuristic approach with a chi-squared (χ^2) test. The objective of SRPT is to *probabilistically* assign each object (e.g., a storm) composed of multiple time-series to a set of predefined classes (e.g., the classes "positive", "negative" and "maybe" for storms). Since the time-series are read incrementally[12], SRPT can be interpreted as a stream mining algorithm; since it operates on multiple streams, it is relational. Dissimilarly to our problem specification (c.f. Chapter 1), though, SRPT does not deal with nominal attributes. Since the products purchased by a customer or the medical treatments of a patient cannot be transfered to time-series, and since SRPT assumes only a priori defined nominal *meta*-data, SRPT does not seem to transfer to the problem we want to solve.

SRRF (Spatio-temporal Relational Random Forests) [Supinie et al., 2009] is an extension to their earlier work on SRPT [McGovern et al., 2008]. The approach is inspired from the random forests [Breiman, 2001] for static

[11] There is not much text in [McGovern et al., 2008] on what art of summarisation information should be used in the general case, how it is to be computed and updated efficiently, nor about the space it consumes. The authors mention possible examples of meta-information for an example application, though.

[12] It is not really incremental but it can take the order and time information into account. Moreover, there are no methods provided for updating or altering the tree if some concept drift occurs.

data, where different C4.5 [Quinlan, 1986] are trained on different subsets of the data with a different attribute set (selected randomly). Each subset of the data is created through sampling and individual SRPT in SRRF uses a greedy approach to induce the decision trees over the subset. The trees are left unpruned.

In their introductory work on learning regression over evolving multi-relational streams [Ikonomovska and Dzeroski, 2011], Ikonomovska and Dzeroski have identified two potential solutions for relational regression on evolving streams that are worth exploring further. First, is the synchronisation of the streams. In multiple-interrelated streams, target objects (the perennial objects from the target stream) and the objects that reference them from other streams, do not all arrive at once and it is imperative that the objects are stored for some amount of time. In this regard they have proposed the use of sliding windows of varying sizes over each individual stream [Ikonomovska et al., 2011] and only use most recent data. Their second proposal deals with the computation of summaries or sketches over the multi-relational streams. These summaries can then be efficiently queried for building the regression model.

While work exists on summarising individual streams for approximate query answering, summarisation of evolving relational streams is still a new field. Unlike the relational stream mining methods discussed above, work of Csernel et al. focus on the summarisation of the multi-relational streams [Csernel et al., 2007]. Unlike propositionalisation, this method does not summarise the streams for a propositional learner to be applied, but simply generates a summary of the streams. In doing so, though, it preserves as much information on the entities and their relationships as possible. For this purpose, they use micro-cluster [Aggarwal et al., 2003] and blooms filters[13] [Bloom, 1970] to summarise the content of the individual entity stream. For summarising the relationship information between different streams, they use cross-table summary structures. Their method only considers the case where new data gets added to the individual streams, i.e., they do not forget old information.

Discovery of relational patterns on streams is studied in [Fumarola et al., 2009; Ceci et al., 2009]. Fumarola et al. [Fumarola et al., 2009] present a sliding window algorithm for mining relational frequent patterns from data streams. The relational patterns are discovered for each slide (unit of sliding window). The relational patterns are stored using SE-Tree [Rymon, 1993] and, once the patterns have been mined the slide is discarded. Instead for maintaining a separate SE-Tree for each slide, each node (or

[13]Bloom filters are based on hash-coding

pattern) in the SE-Tree keeps a $w-$sized sliding vector $s_v(w)$ that stores the support of all patterns in each slide of the window separately. To determine if a pattern P is frequent, an approximate support is computed using the local support values $s_v(w)$.

The work of Ceci et al. [Ceci et al., 2009] searches for novel relational pattern over a sliding window. According to [Ceci et al., 2009] a pattern P defines novelty if it has approximately the same support for all timepoints within a window $[t_i, t_{i+w})$, except for the last timepoint t_{i+w}, i.e.,

$$supp_{t_i}(p) \approx \ldots \approx supp_{t_{i+w-1}}(p) \neq supp_{t_{i+w}}(p)$$

Their approach is made up of two steps. During the first step relational patterns are discovered. In the second step only those are retained that are consistent with the novelty criterion defined above.

2.3 Summary

In this chapter we have provided a concise and targeted survey of the relevant works that set the contribution of this thesis within state-of-the-art. We have discussed notable work from relational mining, especially, the work on propositionalisation of relational databases. We have also discussed important work from the domain of stream mining, which includes management of streaming data, stream clustering, stream classification and rule discovery from data streams. Additionally, this chapter also serves as a motivation and provides basics that are essential for understanding the contribution of this thesis, i.e., for solutions on managing and mining over perennial objects, which are discussed in the subsequent chapters.

CHAPTER 3

Managing Perennial Objects

This chapter forms the core of this thesis. We start with an example in Section 3.1, where we briefly outline our solution for handling perennial objects from multiple-interrelated streams. In Section 3.2, we describe the incremental propositionalisation algorithm and the sub-procedures that are essential for the algorithm. In Section 3.3, we present an incremental clustering algorithm that operates over a propositionalised stream of perennial objects. The terms and symbols that are used in this chapter are given in Table 3.1.

3.1 Building Perennial Objects

Perennial objects are complex and dynamic (as already discussed in Sections 1.1.2), that are fed with objects from the neighbouring interrelated streams. The algorithm first proceeds by determining a *target* stream $\mathcal{T}$, which is central to learning. For example, stream of *customers* at an online retail store and/or stream of *patients* at a hospital forms the target stream. Depending on the type of relationship of the target stream with the other streams $\langle S^1, \ldots S^j \rangle$, the algorithm concatenates the objects from other streams directly (1-to-1 or m-to-1) or after the aggregation/summarisation (1-to-m) of the objects from other streams.

The incremental propositionalisation algorithm operates at discrete timepoints $\langle t_1, \ldots, t_i, \ldots \rangle$. Unlike the stream mining paradigm of Guha et al. [Guha et al., 2003], where t_i is the arrival time of an object x_i, in our scenario, t_i is defined as the end of some period, e.g. a week or year: this is closer to the intuition of building models at regular intervals. In the remaining of the section, the incremental propositionalisation algorithm is outlined informally, with the help of a running example. We explain it in detail in Section 3.2.

The core idea of propositionalisation over static data is [Siddiqui and Spiliopoulou, 2009a; Kroegel, 2003]:

Table 3.1: List of used terms and symbols.

Symbol	Notation
t_1, t_i	initial and current timepoint, (model is learned at discrete timepoints).
Schema and Streams	
$\mathcal{T}$	Target stream of perennials: central to learning.
$\mathcal{S}$=$\langle S^0 \dots S^J \rangle$	A set of multiple-interrelated streams, where S^0 is always the target stream, i.e., $\mathcal{T} = S^0$.
$\mathcal{X}^B$	Base schema of multiple-interrelated streams.
$\mathcal{X}^*$	*A star schema* of $\mathcal{X}^B$ with $\mathcal{T}$ at its centre.
$\mathcal{P}_i$	Propositionalised perennial stream at t_i.
Memory Management	
C_i^j, W_i^j	cache *xor* window associated with S^j at t_i. $C_i^{\mathcal{T}}$ is the cache associated with $\mathcal{T}$. Alternatively, C_i^S is the cache associated with S. Alternate notation is used for other symbols as well, e.g., N^S, $Q_i^S(x)$, etc.
Y_i^j	Y_i^j stands for C_i^j xor W_i^j.
ω	Length of W^j defined over ephemeral streams.
$\mathcal{N}$=$\langle N^0 \dots N^J \rangle$	Set of cache sizes. N^j is the size of C^j over S^j. If S^j is ephemeral, then $N^j = \emptyset$. $N^{\mathcal{T}}$ is the size of $C^{\mathcal{T}}$.
Cache Mechanism	
$Q_i^j(x)$	# references for a perennial object $x \in S^j$ at t_i.
$\hat{Q}^j(x)$	composite counter for maintaining references for $x \in S^j$.
M^j, M_{soft}^j	Hard and soft limit on # counters $\hat{Q}$ for S^j.
$\beta \in [1, \infty)$	Rate of decay. Used in conjunction with $\beta^{-\Delta t}$.
Space Adjustment for Nominal Attributes	
A	An attribute from stream S^j.
$domain(A, t_i)$	Domain of an attribute A (nominal) at t_i.
r_A	# count columns reserved for A (nominal).
Incremental Clustering	
K	Number of clusters to be discovered.
ζ_i	Clustering model learned over $\mathcal{P}_i$ at t_i.
ρ_{min}	User-defined threshold for change detection.

„Each object x of the target table (or stream) $\mathcal{T}$ is expanded by joining it with all objects that refer to it (via external target identifiers). If x joins with more than one object, i.e., with a set of objects $matches(x) \leftarrow \{y_1, y_2, \dots, y_n\}$, then these ob-

jects must be summarised into a single propositionalised object. This is done by adding new summary attributes/features to the schema of $\mathcal{T}$ for each distinct attribute that appears in the set $matches(x)$."

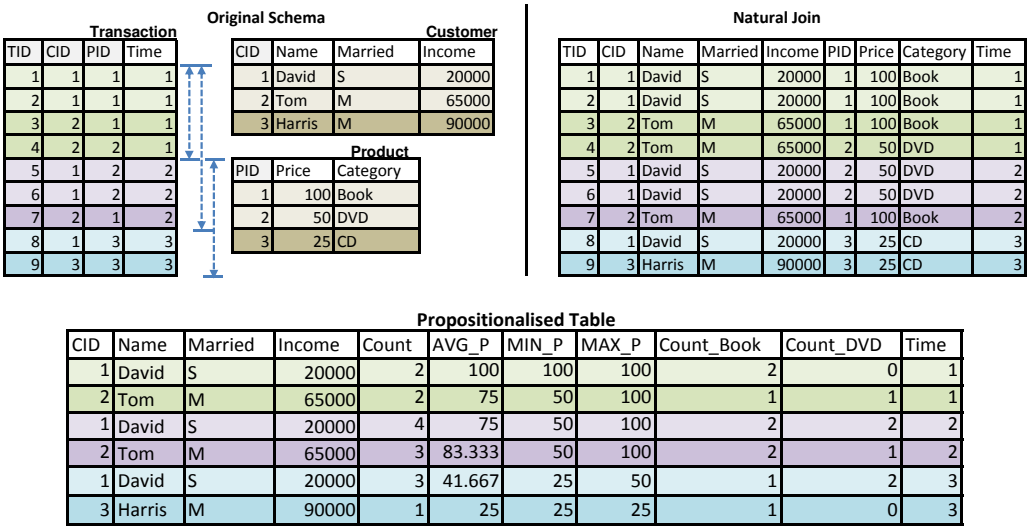

Figure 3.1: Incremental Propositionalisation of a perennial stream (a) consisting of target stream *Customer*, which is fed by objects from the streams *Transaction* and *Products* (blue arrows depict the sliding-window as it moves across timepoints while lighter shades in the Customer and Product stream represents the stream cache), (b) n-way join of the target stream with other streams and (c) propositionalised stream.

The task of the incremental propositionalisation algorithm is to transform multiple interrelated streams consisting of perennial objects, into a single stream, a *propositionalised stream*. In Figure 3.1 we see multiple interrelated streams arriving at different speeds. The 'Customer' stream is associated with the 'Product' stream and the 'Transaction' stream. We observe the Customer stream as the *target stream*, in the sense that this is the stream on which we want to perform mining, exploiting also the information from the other streams that feed it. It is obvious that the number of products purchased by a customer changes over time and so is the information accumulated about each customer's product preferences and regularity of purchases. It is also obvious that a customer object must be joined with all transactions of this account, and be thus kept for as long as such transactions are expected; customer objects are perennial (and so

are products). On the other hand, the transactions themselves may be discarded immediately after being read; they are ephemeral.

The incremental propositionalisation algorithm assigns a *cache* and a secondary storage for long-term maintenance to each stream of perennial objects, and a *sliding window* to each stream of ephemeral objects. In the example of Figure 3.1, a sliding window of two time units (depicted by dotted arrows) is specified for the 'Transaction' stream and caches are defined for the 'Customer' and the 'Product' stream; each of the caches can accommodate two objects.

Propositionalisation is performed incrementally at each timepoint t_i on the contents of the caches and windows. It starts with a semi-join between the contents of the cache for the target perennial stream $\mathcal{T}$ and the cache/window of each stream S^j associated to $\mathcal{T}$. Hence, for each stream S^j that is in 1-to-m or m-to-n relationship to $\mathcal{T}$, each object $x \in \mathcal{T}$ is associated with the set of matching objects $matches(x) \subset S^j$.

Then, the incremental propositionalisation algorithm summarises the objects in this set into a single *sub-object*. To summarise the values of each numerical attribute A in $matches(x)$, the algorithm generates four attributes: the *min*, *max*, *sum*, and *average* of A's values seen in $matches(x)$. To summarise each nominal attribute A, we generate as many $columns(r_A)$ for A as there are distinct values in $\bigcup_x matches(x)$ at t_1. The domain of A may change after t_1, in the sense that previously unseen values emerge, while old values are no more referenced. If the domain grows larger than a user provided threshold r_A, then values are grouped into $K = r_A$ clusters. At the end of the propositionalisation phase, each object of the (perennial) target stream $\mathcal{T}$ is expanded by summarised attribute values from each stream S^j associated with it.

3.2 Inceremental Propositionalisation Algorithm

The incremental propositionalisation algorithm is explained in detail in the following sections. First, we describe each individual component and then explain the final algorithm at the end.

3.2.1 Memory Management

The incremental propositional algorithm operates at discrete timepoints. Because streams are by definition infinite, only a small portion of the stre-

aming data, arriving at timepoint t_i, can be maintained in the main memory. This data is used to built the model ζ_i at t_i. In this section, the memory management strategies for handling perennial and ephemeral objects are discussed, which explain how we update memory structures to prepare for adapting the model ζ_{i+1} at t_{i+1}.

A perennial object is fed by objects from neighbouring streams which can be ephemeral or perennial. In the running example (cf. Figure 1.1), the streams 'Customers', 'Products' and 'Webpages' are perennial by nature and the objects from these streams may not be forgotten. On the other hand, the stream of 'Transactions' and 'Clicks' are ephemeral; as individual transaction (or click) objects get older, they have less and less influence on the model ζ_i at t_i. These different properties require different mechanisms of memory management. We employ a caching strategy coupled with the secondary storage for perennial streams, and a sliding window of fixed-size for ephemeral streams. The sizes of the individual caches defined over perennial streams may vary from one stream to the other. The size of the sliding window ω is uniform for all ephemeral streams.

Cache Management For Perennial Objects

We associate each stream of perennial objects S^j (same for target stream $\mathcal{T}$) that may not be forgotten with a fixed-size cache C^{S^j}. Because of the finite size of the cache, only a limited number of the objects can be maintained, while the rest are kept in the secondary storage. The contents of the cache $C_i^{S^j}$ get updated at each discrete timepoint t_i, as new data arrive. The objects to be stored inside the cache are chosen on the basis of their *importance*. The importance of an object for the caching purpose can be calculated in various ways, e.g., most frequently referenced, most recently referenced.

If an object that is not already in the cache, is referenced by an object from another stream, then this object must be fetched from the secondary storage; then, some objects must be put back to secondary storage to make space. The process of cache maintenance is depicted in Figure 3.2. In the example, the size of the cache is three and the objects to be cached are chosen on basis of how frequently they are referenced. At timepoint t_i, the cached objects are X, P and Z. With the arrival of new data at a later timepoint t_{i+1}, the reference count for object D becomes greater. Based on its score, object D is brought into the cache while object Z is spilled onto the secondary storage.

The mechanisms determining which objects should be retained in the cache are presented in the following.

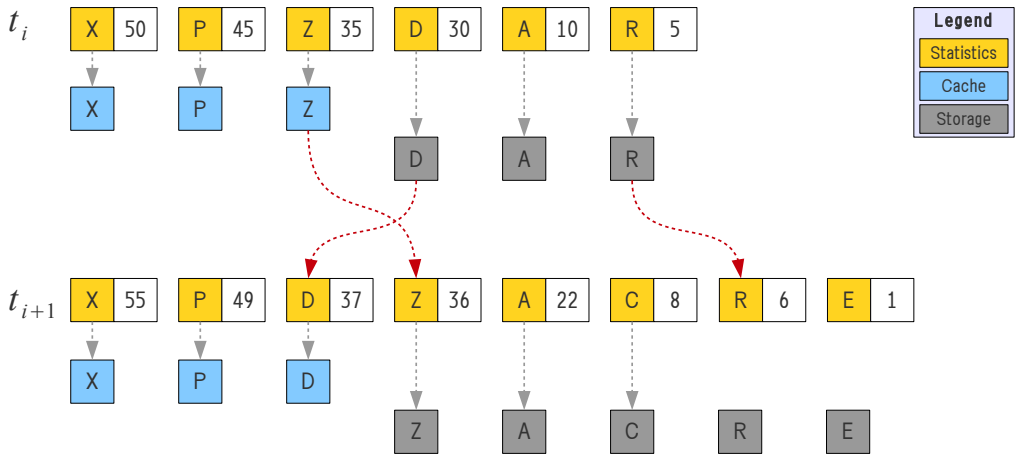

Figure 3.2: Cache maintenance for perennial objects. The cache maintenance involves keeping track of the perennial objects as they arrive in the streams and then keeping the ones that are important; an example function is $G()$ (cf. Equation 3.1); alternate importance functions are discussed below.

Cache Mechanism 1 The basic cache update mechanism is presented in Algorithm 1: CacheUpdate. It operates over each stream (including the target stream $\mathcal{T}$) that has a cache associated with it, at the end of each timepoint (Lines 2 & 5). For an object $x \in S$ that gets referenced during t_i, the algorithm counts the number of times x was referenced by an object y from another stream S' (such that $S \xrightarrow{1-to-m} S'$). CacheUpdate calculates the number of times an object x was referenced and found in the cache C_i^S, denoted as $H(x)$ (the hits), and the number of times x was referenced and not found in C_i^S, denoted as $M(x)$ (the misses). As can be seen in Line 4, the importance of an object x at t_i is computed as the number of references to it, thereby assigning a higher weight $(1 + \epsilon)$ to hits, i.e., to the objects already residing in the cache.

$$G^S(x) = \sum_{u=i-\omega}^{i} Q_u^S(x) \times decay(t_u, t_i) \tag{3.1}$$

After the calculation of hits and misses for all the perennial objects, CacheUpdate updates the caches for all streams. The computation of importance $G(x)$ (cf. Equation 3.1) to be achieved by caching an object $x \in S$ is computed by considering the number of references to x within the whole sliding window of length ω (Lines 10-12). To reduce the influence of old

statistics, we use the decay function depicted in Equation 3.2[1], which lowers the weight of old data exponentially. The referenced objects are sorted (Line 13) and the top $N^{\mathcal{T}}$ positions are brought into the cache $C^{\mathcal{T}}$ (Line 14).

Algorithm 1: CacheUpdate

Input : $t_i, \mathcal{S}, \mathcal{N}, \omega, \beta, \epsilon$

1 $\mathcal{T} \leftarrow S^0 \in \mathcal{S}$

2 **foreach** *stream* $S \in \mathcal{S}$ *with* $N^S > 0$ **do** /* Each S with a cache */

3 **foreach** *referenced tuple* $x \in S$ **do**

4 $Q_i^S(x) = (1+\epsilon)\mathtt{H}(x) + \mathtt{M}(x)$ /* Count references */

5 **foreach** *stream* $S \in \mathcal{S}$ *with* $N^S > 0$ **do** /* Each S with a cache */

6 **foreach** *referenced tuple* $x \in S$ **do**

7 **if** $S \neq \mathcal{T}$ **then**

8 $Q_i^S(x)$+= $\mathtt{H}(x) + (1+\epsilon)\mathtt{M}(x)$

9 $Q_i^S(x)$+= $PropInfo(x, S', t_i)$ [2]

10 $G^S(x) = 0$

11 **foreach** $u = i - \omega$ **to** i **do** /* Compute importance */

12 $G^S(x) += Q_u^S(x) \times decay(t_u, t_i)$

13 sort G^S [desc]

14 $C_i^S \leftarrow$ top-N^S objects of G^S

The computation of the importance $G()$ (Lines 10-12) aims to maximize the number of objects contributing to the model. It thus favours objects that are expected to produce more output when they are joined with the neighbouring streams. This corresponds to the MAX-Subset error measure proposed in [Das et al., 2003] for *sliding-window joins*.

$$decay(t, t') = \begin{cases} 1 & \Delta t = 0 \\ \beta^{-\Delta t} & otherwise \end{cases} \tag{3.2}$$

The updating of the caches starts at the target stream $\mathcal{T}$ and then to the streams that are in direct relationship with $\mathcal{T}$ and so on. The traversal order of the streams is depicted in Figure 3.3. The Lines 6-9 are invoked for all the streams that feed the target stream but are ignored when the cache $C^{\mathcal{T}}$ is to be updated. Cache updating is an iterative process, as depicted

[1] In Equation 3.2 $\Delta t = |t - t'|$.

[2] S' is in relationship with S such that S' is further away from $\mathcal{T}$ than S.

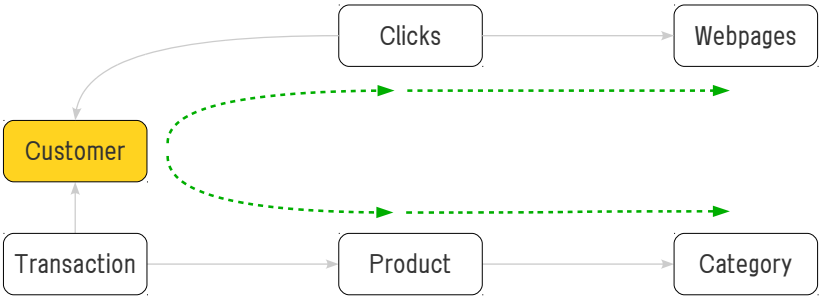

Figure 3.3: The dotted line depicts the traversal order of the streams. The traversal always starts from the target streams (depicted as shaded box).

in CacheUpdate: a cache is updated by removing objects from it and inserting new ones; this affects the hits $H(x)$ and misses $M(x)$ for objects referenced by them.

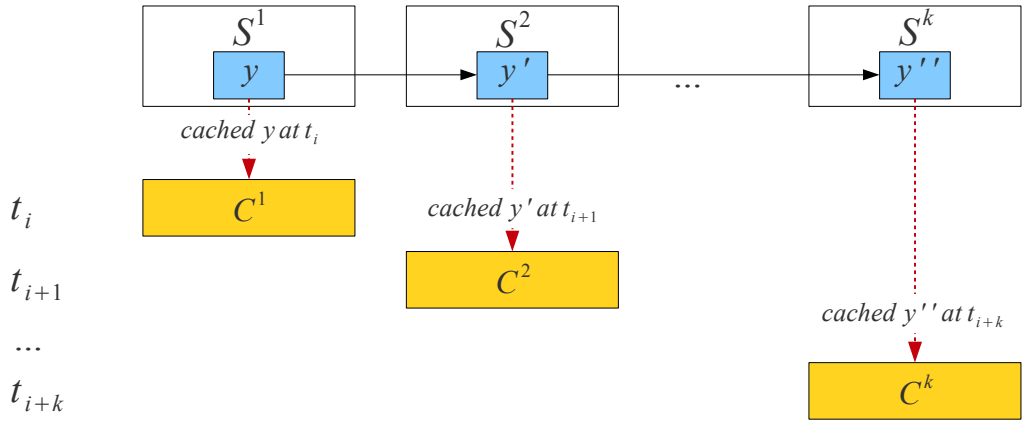

Figure 3.4: A chain of relationships among different streams.

To explain the effects and side-effects of cache updating, we first consider three streams S^1, S^2 and S^3 from $\mathcal{X}^B$ that use caches and constitute a chain of relationships $S^1 \underline{m{-}to{-}1} S^2$ and $S^2 \underline{m{-}to{-}1} S^3$ (c.f. Figure 3.4). At timepoint t_i, let an object $y \in S^1$ reference an object $y' \in S^2$, i.e., $y \to y'$. Assume that during cache updating at t_i, object y' is fetched and cached in C^2_{i+1}. If $y' \to y''$ for some $y'' \in S^3$, then y' contains a possibly dangling reference; the need to fetch y'' became known after y' was cached. Indeed, y'' can only be fetched to the cache C^3_{i+2} at the timepoint t_{i+2}. For a chain of $\underline{m{-}to{-}1}$ relations $S^1, S^2, \ldots, S^k$ of arbitrary, it will take k timepoints to amend a cache miss for an object $x \in S^1$ that propagates through to S^k. In other words, only after k timepoints, it would be possible to reconstruct

the complete perennial object x, assuming (optimistically) that all objects needed are still cached at t_{i+k}.

To load all objects with their references, the caches are updated iteratively. Once the first cache has been updated, we calculate the references to the objects of the remaining streams anew for the next cache update (Line 8). This time we assign a higher weight $(1+\epsilon)$ to the misses, making referenced objects outside the cache more likely candidates for caching.

Line 9 of the CacheUpdate algorithm deals with a problem of transitivity across a chain of relationships between individual streams. We first explain the notion of transitivity and then explain the mechanism working behind Line 9.

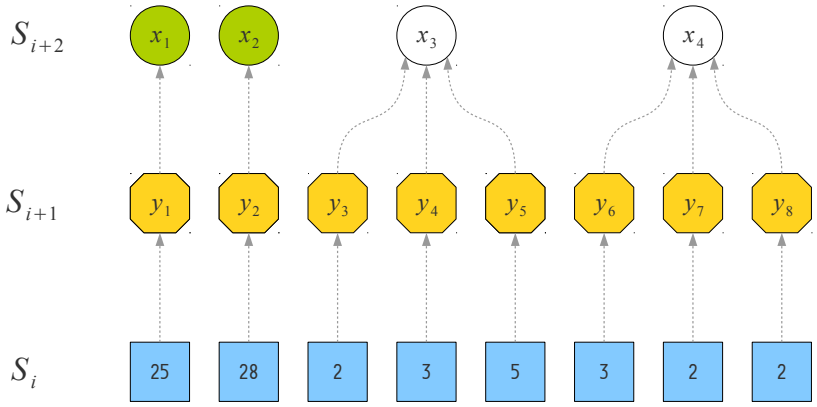

Figure 3.5: Transitive references.

Transitive References In a relational database, streams that are not directly connected can reference each other indirectly. Assume three streams S^1, S^2 and S^3, such that S^1 and S^3 are not directly related to each other. However, S^1, S^2 and S^2, S^3 are in relationships with each other. Then, streams S^1 and S^3 are said to have a *transitive relationship*. This notation of transitivity extends to the individual objects within the streams as well. For example, for objects $(x_1, x_2, x_3) \in (S^1, S^2, S^3)$, respectively, x_1 makes a transitive reference to x_3, if $x_1 \rightarrow x_2$ and $x_2 \rightarrow x_3$. The CacheUpdate maintains the objects from individual streams using only the direct references. We propose an improvement to CacheUpdate that utilises the transitive references as well.

Consider the example in Figure 3.5, where all streams are associated with caches and each stream is in m-to-1 relationship with the next one, as shown in Figure 3.4. Assume that the object $x_3, x_4 \in S^j$ are referenced

more from S^{j-1} than any other object, so they would be preferred over objects x_1, x_2. This is consistent with the MAX-subset error measure that promotes objects resulting in larger output. Now consider the 3-way join between S^j, S^{j-1}, S^{j-2}. Objects x_3 and x_4 would be again preferred over x_1 and x_2, and this will be inconsistent under the MAX-subset error measure [Das et al., 2003], because x_1, x_2 are going to produce more output objects (c.f. Figure 3.5). The reason for the inconsistency are the objects referenced by $x_1, x_2, \ldots, x_4$. Object referencing is transitive; we exploit this property in that we add transitive references to the hits (respectively, misses) of a object x (Line 9). The responsible function $PropInfo()$ is depicted in Equation 3.3.

$$PropInfo(x, S^u, t_i) = \sum_{y \in S^u \wedge x \rightarrow y} \frac{Q_i^u(y) + PropInfo(y, S^{u'}, t_i)}{D} \tag{3.3}$$

This function takes as input an object x and a stream S^u from a chain of streams, such that $S^u \underline{\underline{m-to-1}} S^{u'}$. It computes the references of x to S^u and invokes itself again for each object y referenced by x. At each invocation, it uses an information decay parameter D. This parameter ensures that the impact of those additionally counted references drops as we navigate down the chain of inter-linked streams.

After the execution of Lines 8 & 9 for the streams other than $\mathcal{T}$, the caches are updated similarly to that of $C^{\mathcal{T}}$.

CacheUpdate in its basic variant is very resource intensive. It requires an infinite amount of space to store the cache statistics for the perennial objects. Its space complexity is $O(n \times t)$, where n are the total number of users seen in the perennial stream and t is the total number of timepoints observed, which is possibly infinite. In the next section, we present an improvement for the CacheUpdate called SmallCache. The improvements in SmallCache have been inspired from the *space saving* algorithm of Metwally et al. [Metwally et al., 2005, 2006].

Small Cache The space saving algorithm [Metwally et al., 2005, 2006] computes the top-N items from a data stream under limited space. It is a very simple and straightforward algorithm. It is a counter-based algorithm, where counters are maintained for a limited number of individual objects. For an object, the algorithm maintains an observed count ctr^o and an upper bound on the number of missed instances ctr^b. Incoming objects update the counters and fill up the space reserved for the counters (at most M counters can be maintained in the main memory). If an object x that

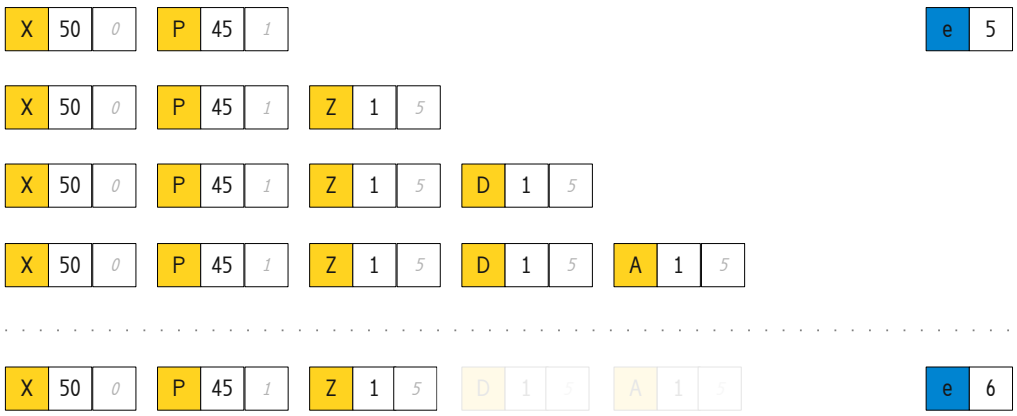

Figure 3.6: Updating of the small cache. The right most block stores the upper bound for the missed instances. When the counters have to be updated, the minimum upper bound also gets updated.

does not have a counter arrives, and the counter space is already filled, the space saving algorithm selects the counter ctr_y associated with an object y that has the smallest count and replaces its counters with counters for x: $ctr^o_x = 1$ and the upper bound is approximated as $ctr^b_x = ctr^o_y + ctr^b_y$.

Similarly, for an object x from stream S^j, SmallCache maintains a composite counter $\hat{Q}^j(x)$, i.e., $\hat{Q}^j(x)$ consists of $\hat{Q}^j_o(x)$ for the observed counts and $\hat{Q}^j_b(x)$ for maintaining an upper bound on the number of missed objects. The counter $\hat{Q}^j_o(x)$ is updated incrementally as new objects arrive. At each new timepoint, the counters are also subjected to decay β^{-1}.[3]

SmallCache maintains only a fraction of the observed objects in the main memory. Unlike the space saving algorithm, where at most M counters can be maintained, SmallCache puts a *soft* limit on the number of counters that can be maintained. Soft limit implies that SmallCache has fractionally more space while updating the counters than its storage space. For example, for a hard counter limit of M, as a new object arrives, the algorithm adds one more counter making the total number counter to be $M+1$. The algorithm keeps on adding new counters until it reaches M_{soft}. When M_{soft} is reached, it stores the count value for the counter value with minimum number of hits, then proceeds to remove all the excess counters and retains only the top-M counters. This process is depicted in Figure 3.6. Doing this also helps in keeping the bound on missed instances

[3] β^{-1} is the same decay function as $\beta^{-\Delta t}$ in Equation 3.2. Since the computations for SmallCache are incremental and are adjusted from one timepoint to the next, $\Delta t = 1$.

Algorithm 2: TargetIdPropagation

Input : $\mathcal{X}^B, \mathcal{S}, C^{\mathcal{T}}, Y^1 \dots Y^J$
Result: $\mathcal{X}^*$

1 **if** $\mathcal{X}^*$ *doesn't exist* **then**
2 Initialise $\mathcal{X}^*$
3 $tree$ = `GenerateSchemaTree(`$\mathcal{X}^B$, $\mathcal{T}$, $S^1 \dots S^J$`)`
4 **foreach** S, $S' \in tree$ **do** `/* ` S' `is child and` S `is parent */`
5 **if** S' *doesn't contain targetId* **then**
6 **if** $star(S')$ *doesn't exist* **then**
7 Create $star(S')$
8 `INSERT INTO` ($star(S')$)
`SELECT` $Y.targetId, Y'.*$
`FROM` Y, Y'
9 Update $\mathcal{X}^* \leftarrow star(S')$
10 Return $\mathcal{X}^*$

lower, which in space saving algorithm [Metwally et al., 2006] increases at a very rapid rate.

3.2.2 Propagation of Target Identifiers

After the cache updating, the objects in the caches and windows of each individual stream are combined into a single propositionalised stream $\mathcal{P}$. This process is done by first performing a join on the caches, or sliding windows, of the streams that are related to each other, and then aggregating the objects that are maintained in these windows and caches. Executing a join over streams S is a complex operation. The complexity increases in relation to the depth of join. That is, for a join between streams $S^1 \bowtie S^2 \bowtie \dots S^m$, the complexity is $O(n^m)$, where N is the maximum of the number of objects inside the cache/window of each individual stream. In a streaming environment where incoming data are infinite and arrive at fast speed, it is essential for the algorithm to have a low computational complexity.

In order to achieve low complexity, Krögel proposed to replace complex multi-relational joins with binary joins [Kroegel, 2003] and devised a procedure, TargetIdPropagation. The process of aggregation is anchored on the perennial objects from the target stream $\mathcal{T}$. A stream that is in direct

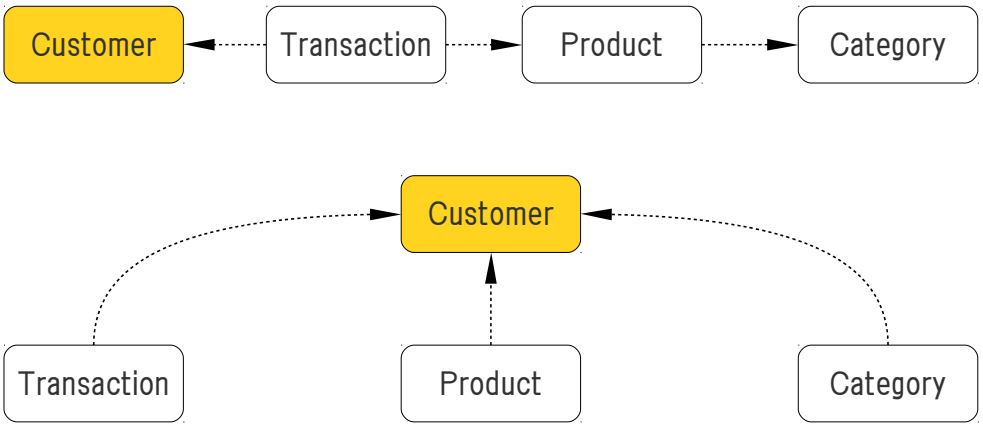

Figure 3.7: Schema transformation: (top) Base schema into (bottom) 'a star schema'.

relationship[4] with the target stream $\mathcal{T}$, has the target identifier as a foreign key, and in turn has the information as to on which attribute the aggregation should be performed. For a stream S', that is not directly linked with $\mathcal{T}$, i.e. $\mathcal{T} \bowtie \ldots \bowtie S'$, this information is missing and aggregation can only be achieved via a deep join. The procedure makes this information explicit for the streams, such as S', by propagating the target identifier from $\mathcal{T}$ to the streams with which $\mathcal{T}$ is not directly connected. Once the target identifier information has been propagated, every stream can be joined directly with the target stream, thus, avoiding deep joins. This process changes the schema into a star like formation and is referred as *a star schema*[5], $\mathcal{X}^*$. In Figure 3.7 we see the base schema at the top and at the bottom of the same figure its transformation into 'a star schema'. The target stream 'Customer' is at its centre, and all other streams are directly connected to it.

We have modified the procedure for target identifier propagation of Krögel for the streaming case Kroegel [2003]. The algorithm for this task is depicted in the Algorithm 2: TargetIdPropagation. It takes as input all the streams and the base schema $\mathcal{X}^B$, which has the information on relationships between each individual stream. The algorithm first transforms

[4] i.e., $S \overset{1-to-1}{\longrightarrow} \mathcal{T}$, $S \overset{1-to-m}{\longrightarrow} \mathcal{T}$, or $S \overset{m-to-1}{\longrightarrow} \mathcal{T}$

[5] In data warehousing, a similar notion, *star schema*, is used, albeit with a different meaning. In 'a star schema' of Krögel [Kroegel, 2003], each table is directly connected to the central table, where as the tables in the star schema of data warehousing are recursively folded with onto their parent tables until they are at a maximum depth of 1 from the central table.

the base schema $\mathcal{X}^B$ into a tree that has $\mathcal{T}$ at its root and the streams referencing it as its children (Line 3). The tree edges are the foreign key relationships, across which the identifiers of $\mathcal{T}$ are propagated towards the leaf nodes. The traversal order is same as depicted in Figure 3.3.

Initially, S = $\mathcal{T}$ with S' as $\mathcal{T}$'s leftmost child (Y, Y' are their caches xor windows, respectively). If the corresponding star table $star(S')$ of $S' \in \mathcal{X}^B$ doesn't exist in $\mathcal{X}^*$, first it gets created (Line 7). The target identifier propagation is then performed using the insert query operation on the contents of Y, Y' (caches xor windows for S, S', respectively) in Line 8.

3.2.3 Transforming the Multi-Table Streams

To aggregate all objects into the propositionalised stream $\mathcal{P}_i$ at timepoint t_i, after the target identifier propagation, we propose Algorithm 3: IncrementalPropositionalisation. For Algorithm 3, we extend the propositionalisation algorithm RelAggs (c.f. Section 2.1.3) for static data [Kroegel, 2003]. We reference the Algorithm 3 as we explain the process in the following.

At the current timepoint t_i, the contents of $\mathcal{T}$ in the cache $C_i^{\mathcal{T}}$ must be "propositionalised" to accommodate the information of each cache or window $Y_i^{S^j}$ of stream S^j. This process requires the computation of the joins between cache or window $Y_i^{S^j}$. To reduce the overhead of the complex join operations, IncrementalPropositionalisation starts by propagating the target identifier information to all streams (Line 2).

After the target id propagation, the algorithm iterates over all streams that feed the target stream. It performs a semi-join between the cache $C_i^{\mathcal{T}}$ of the target stream $\mathcal{T}$ and the cache or window $Y_i^{S^j}$ of each stream S^j that is in 1-to-1 or 1-to-m relationship with $\mathcal{T}$. Accordingly, it expands or concatenates $x \in C_i^{\mathcal{T}}$ with the matching object $y \in Y_i^{S^j}$, i.e., $x := x + +y$ (Line 5). At some later timepoint, if x is already expanded with some (old) object of $Y_i^{S^j}$, this old expanded object is replaced by the new one. This initially unintuitive approach is best explained by an example: consider a stream of 'customers' (the target stream) and a stream of their 'addresses (home)'. The streams have 1-to-1 relationship with each other, i.e., a customer can only live at one address. When a customer changes his address, his new address must replace the previous one.

For each S^j that is in 1-to-m relationship with the target stream $\mathcal{T}$ (Line 6), IncrementalPropositionalisation associates each object $x \in C_i^{\mathcal{T}}$ with the set of matching objects $matches(x) \subseteq Y_i^{S^j}$. The objects in this set must be summarised into a single *propositionalised object*.

For the summarisation of the values of each numerical attribute A

Algorithm 3: IncrementalPropositionalisation

Input : $\mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon$
Output: Propositionalised stream $\mathcal{P}_i$

1 $\mathcal{T} \leftarrow S^0 \in \mathcal{S}$
2 $\mathcal{X}^* \leftarrow$`TargetIdPropagation(` $\mathcal{X}^B, \mathcal{S}, C^{\mathcal{T}}, Y^1 \dots Y^J$`)`
3 **for** $j \leftarrow 1$ **to** $|\mathcal{S}|$ **do**
4 **if** $\mathcal{T}$ *and* S^j *are in* ***1-to-1*** *or* ***m-to-1*** *relationship* **then**
5 $\mathcal{P}_i \leftarrow$ Concatenate objects of $Y^{S_i^j}$ and $Y^{\mathcal{T}_i}$
6 **else if** $\mathcal{T}$ *and* S^j *are in* ***1-to-m*** *relationship* **then**
7 **foreach** *attribute* $A \in Y_i^j$ **do**
8 **if** A *is numerical* **then**
9 $agg_A \leftarrow$ Compute avg(A), sum(A), min(A), max(A)
10 $\mathcal{P}_i \leftarrow$ Add computed aggregates agg_A
11 **else if** A *is nominal* **then**
12 $cnt_A \leftarrow$ Compute counts for known nom values in A
13 $\mathcal{P}_i \leftarrow$ Concatenate (adjust cnt_A in r_A columns)
14 `CacheUpdate(` $t_i, \mathcal{S}, \mathcal{N}, \omega, \beta, \epsilon$`)`
15 `NominalSpaceAdjustment(` $t_i, \mathcal{S}, r_A$`)`
16 Return $\mathcal{P}_i$

among the $matches(x)$, the algorithm generates four attributes and accommodates in them the *min*, *max*, *sum* and *average* of the A values seen in $matches(x)$ (Line 9) and adds this aggregated information to the propositionalised stream $\mathcal{P}_i$ (Line 10). In the example in Figure 3.1 the customer *David* has purchased two products; the attribute *price* is numeric and the values in attribute *price* for the bought products are summarised in the columns *maxP, minP, avgP, sumP*. Differently from example above about customers and their addresses, as the customer does more transactions, the four generated attributes are *updated* for each object x.

For summarisation of each nominal attribute, *IncrementalPropositionalisation* counts the observed frequency of each distinct nominal value in $matches(x)$ (Line 12). *IncrementalPropositionalisation* generates r_A amount of columns for a nominal attribute A in the propositionalised stream $\mathcal{P}_i$. The observed frequency of the distinct nominal values are recorded in these generated columns (Line 13). In the example in Figure 3.1, the at-

tribute *category* in 'Product' stream takes the values *Book* and *DVD*. Two count columns are assigned for it, where the observed frequency of the values *Book* and *DVD* gets recorded.

In the static scenario, or at timepoint t_1, this is trivial, as one knows the number of distinct nominal values for A and the required number of count columns to be generated. In a streaming scenario, for $t_i, i > 1$ the number of values for A to be accommodated may increase, e.g., if the company decides to introduce the categories *eBook* and *CD*. The schema of $\mathcal{P}_i$ can neither be expanded arbitrarily (memory is finite) nor can each value ever seen be retained perpetually in it (memory must be used efficiently). Hence, the algorithm must adjust the available space to the demand. For this, we propose two space adjustment methods for accommodating previously unseen nominal values in the schema of $\mathcal{P}_{i+1}$, for the next timepoint t_{i+1}. We explain these space adjustment methods in Section 3.2.4.

Once the multi-relational stream of perennial objects has been propositionalised, the algorithm prepares for the next timepoint. It invokes Algorithm 1 to update all caches on basis of references observed at the current timepoint t_i for the perennial objects (Line 14). Before returning the propositionalised stream $\mathcal{P}_i$, it invokes Algorithm 4 to update the space reserved for the nominal attributes (Line 15).

3.2.4 Space Adjustment for Nominal Attributes

Consider a nominal attribute A from a stream S^j that was assigned r_A number of columns at the initial timepoint t_1. Each of these columns records the number of times a particular nominal value was observed. At timepoint t_1, the cardinality of its domain $domain(A, t_1)$ was less than or equal to the number of assigned columns r_A, i.e., $domain(A, t_1) \leq r_A$. Assume that the domain of A changes at some later timepoint t_i, such that $|domain(A, t_1)| < |domain(A, t_i)|$. This means that there are newer values that have been observed and need to be adjusted in the schema of the propositionalised stream $\mathcal{P}_i$, i.e., they need to be assigned columns in order for their counts to be recorded in $\mathcal{P}_{i+1}$ for the next timepoint.

The procedure of adjustment is depicted in Algorithm 4: NominalSpaceAdjustment. It is invoked from the Algorithm 3 (Line 15), at the end of each timepoint. It monitors all the nominal attributes from the streams that are in $1 - to - m$ relationship with the target stream $\mathcal{T}$. If no new nominal values have been observed for an attribute A, it performs no adjustment (Line 5). The arrival of previously unseen nominal values in the domain of a nominal attribute A at timepoint t_i can lead to two

Algorithm 4: NominalSpaceAdjustment

Input : $t_i, \mathcal{S}, r_A$

```
1  T ← S^0 ∈ S
2  foreach stream S ∈ S such that S ≠ T do
3      if T 1-to-m S then
4          foreach nominal attribute A ∈ S do
5              if domain(A, t_i) ⊆ domain(A, t_{i-1}) then
6                  No adjustment
7              else
8                  if |domain(A, t_i)| ≤ r_A then
9                      Perform soft adjustment.
10                 else
11                     Perform hard adjustment.
```

cases, either 1) the cardinality of the domain is still less than or equal to the number of assigned columns, i.e., $|domain(A, t_i)| \leq r_A$ (Line 8), or 2) the cardinality of the domain is greater than the number of assigned columns, i.e., $|domain(A, t_i)| > r_A$ (Line 10). The algorithm handles the first case by using a simple procedure called *soft adjustment*, while the second case is handled by employing *hard adjustment*, which is a clustering-based procedure to combine nominal values into homogeneous groups.

Soft Adjustment

If $|domain(A, t_i)| \leq r_A$, it means that the cardinality of the domain has not yet exceeded the number of assigned columns r_A. This leads to cases:

1. If the domain of A at timepoint t_{i-1} is a subset of the domain at t_i, i.e., $domain(A, t_{i-1}) \subset domain(A, t_i)$, it implies that all nominal values from the previous timepoint are still active, and the new nominal values can be accommodated in the remaining columns $r_A - |domain(A, t_{i-1})|$ assigned for the nominal attribute A.

2. If $domain(A, t_{i-1}) \subset domain(A, t_i)$, then some positions must be freed to accommodate the values in $domain(A, t_i)$ by deleting the expired values from $domain(A, t_{i-1})$, and replacing them with the new nominal values from $domain(A, t_i)$. This means that the columns that

have been replaced, acquire new semantics in the propositionalised stream $\mathcal{P}_{i+1}$ at timepoint t_{i+1}.

The *soft adjustment* heuristic de-allocates[6] the columns assigned previously for $domain(A, t_{i-1}) \setminus domain(A, t_i)$ and allocates them for the newly observed nominal values, i.e., $domain(A, t_i) \setminus domain(A, t_{i-1})$.

Hard Adjustment

If the cardinality of the domain for the nominal attribute A exceeds the number of assigned columns, i.e., $|domain(A, t_i)| > r_A$, then we perform the hard adjustment to accommodate the previously unseen values.

Transaction Id	Quantity	Price	Category	Timepoint
1	10	500	Book	1
2	500	40	Book	1
3	8000	5	DVD	1
4	2000	10	DVD	1
5	700	50	**eBook**	2
6	300	20	**CD**	2

Figure 3.8: Domain of nominal value changes over time.

Consider the example shown in Figure 3.8. Assume that, at the initial timepoint t_1, two columns were assigned for the nominal attribute $A^{category}$ for only two nominal values were observed for it, i.e., $domain(A^{category}, t_1)$ = `[Book,DVD]`. At timepoint t_2, two previously unseen nominal values, i.e., `[eBook,CD]` were observed for $A^{category}$. Since new count columns for the new values `[eBook,CD]` cannot be added to the propositionalised stream $\mathcal{P}$, we cluster the values in $domain(A^{category}, t_2)$ into $K = r_{A^{category}}$ homogeneous groups. With the homogeneous grouping we imply that the nominal values that appear in the same cluster/group become indistinguishable and are accommodated in a single count column.

Clustering is based on semantic similarity among the objects. In order to establish the notion of similarity, or closeness, between the values of a nominal attribute, we use a simple heuristic that asserts that *two values are similar if the objects containing them are similar*. However, a nominal value may be contained in one or more objects, e.g., in Figure 3.8, the nominal value `Book` is contained in two transaction objects with ids 1 & 2. This implies that there exists a $1-to-m$ relationship between the nominal values

[6]Currently, the columns to be de-allocated are selected at random, but more sophisticated options are possible, e.g., eliminating the oldest values.

Category	Quantity				Price			
	Min	Max	Avg	Sum	Min	Max	Avg	Sum
Book	10	500	255	510	40	500	270	540
DVD	2000	8000	5000	10,000	5	10	7.5	15
eBook	700	700	700	700	50	50	50	50
CD	300	300	300	300	20	20	20	20

Figure 3.9: Feature vectors of nominal values for performing hard adjustment.

and objects they are contained in. Thus, similarly to the propositionalisation solution of summarising perennial objects into a single object, we aggregate the objects based on the nominal values they contain, into one large object. The resulting objects represent feature vectors for the nominal values. The feature vectors for the example in Figure 3.8 are shown in Figure 3.9. These feature vectors for the nominal values are then used to define similarity between the nominal values. We use Hierarchical Flat Clustering [Mierswa et al., 2006] with cosine similarity to cluster the nominal values into $K = r_A$ homogeneous groups.

3.3 Clustering Perennial Objects

In the previous section we presented the method for the update and maintenance of perennial objects. Ideally, we would employ any clustering algorithm on them. However, conventional algorithms for learning over streams, e.g., the methods of Guha et al. and Aggarwal et al. [Guha et al., 2003; Aggarwal et al., 2003], assume that the schema of the objects does not change. Hence, we propose a dedicated *IncrementalKMeans* for perennial objects from a multi-relational stream.

Our algorithm is inspired by the *OnlineKMeans* for clustering parallel data streams [Beringer and Huellermeier, 2006]. The pseudo-code of IncrementalKMeans is shown in Algorithm 5.

At timepoint t_i, as the data arrives in the perennial streams, it is first transformed from a multi-table stream into a single propositionalised stream $\mathcal{P}_i$ by invoking the Algorithm 3: IncrementalPropositionalisation. It is important to stress here that IncrementalPropositionalisation prefers objects that are frequently referenced over those that are not. The underlying assumption (enforced through MAX-subset measure for stream joins) is that the perennial objects that have more references for other streams are likely to carry more information. Hence, these objects are likely to be more

Algorithm 5: IncrementalKMeans

Input : $\langle\ \mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon\ \rangle$, $\langle\ K, \rho_{min}\ \rangle$

1 **for** $i = 1$ **to** `STREAM_END` **do**
2 $\quad \mathcal{P}_i \leftarrow$ `IncrementalPropositionalisation(`$t_i, \mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon$`)`
3 $\quad$ **if** ζ_{i-1} *is* $\emptyset$ **then**
4 $\quad\quad \zeta_1 \leftarrow$ Perform *clustering* with K centroids over $\mathcal{P}_1$
5 $\quad$ **else**
6 $\quad\quad \zeta_i \leftarrow$ Update the clustering model ζ_{i-1} using $\mathcal{P}_i$
7 $\quad \rho \leftarrow$ Compute jaccard similarity between ζ_i, ζ_{i-1}
8 $\quad$ **if** $\rho < \rho_{min}$ **then** `/* ρ is the similarity between ζi,ζi−1 */`
9 $\quad\quad \zeta_i \leftarrow$ Perform *re-clustering* with K centroids over $\mathcal{P}_i$

mature or *grown*. Thus, they are likely to result in a better and a stable clustering. The objects that have fewer references are treated as noise and are kept away until they mature.

At Line 3, the algorithm checks if there already exists the clustering model ζ_{i-1} from the previous timepoint. For the first iteration of the streaming loop[7], i.e., at timepoint t_1, the previous clustering model would be *null*. In this case, the algorithm would learn the initial clustering model ζ_1 over the first batch of the propositionalised perennial objects $\mathcal{P}_1$ with K centroids.

At each subsequent timepoint, there always exist a model from the previous timepoint. For learning the new clustering model ζ_i, IncrementalK-Means takes the clustering model ζ_{i-1} (i.e., vectors of the centroids and cluster membership information) from the previous timepoint t_{i-1} as the initialisation. The newly arrived data from the propositionalised stream $\mathcal{P}_i$ is then used to incremental update the model ζ_i for the current timepoint (Line 6).

After each iteration, the clustering model ζ_i at t_i is compared with the model ζ_{i-1} from the previous timepoint t_{i-1} (Line 7). For the comparison, we use the Jaccard Coefficient [Tan et al., 2004]. Jaccard Coefficient calculates the degree of similarity between the two models and returns a value close to 1 if the models are similar and close to zero if they are different.

$$JaccardCoefficient(\zeta, \zeta') = \frac{f_{11}}{f_{01} + f_{10} + f_{11}}$$

[7] Line 1 simulates the open-ended nature of stream

where f_{11} is number of objects that are in the same cluster in both ζ, ζ'; f_{01} is the number of objects that are in different clusters, in model ζ and in the same cluster in model ζ'; and f_{10} is the number of objects that are in same cluster in model ζ, and in different clusters in model ζ'.

The comparison is done to check if the clustering model learned over the new data, i.e., newly propositionalised perennial objects from $\mathcal{P}_i$, is still valid or not. If the clustering model across the consecutive timepoints differs more than the user-specified threshold, i.e., the calculated value of Jaccard Coefficient ρ is less than the user-provided threshold ρ_{min}, the current clustering is discarded and a *re-clustering* is performed over $\mathcal{P}_i$.

As we have already mentioned, our method is inspired by OnlineK-Means [Beringer and Huellermeier, 2006]. However, OnlineKMeans works with multiple univariate time-series, while our method considers only the latest version of each object, not the whole series. Moreover, our method is designed to work with complex perennial objects from multiple interrelated streams. OnlineKMeans is designed to work with a fixed number of time-series objects, while IncrementalKMeans allows for the addition of new and the deletion of the old objects. This addition is done smoothly by the IncrementalPropositionalisation as the caching mechanism employed by it prefers objects that are likely to hold more information, over those that do not (cf. Algorithm 3: Line 14). OnlineKMeans can dynamically update the number of clusters over time, whereas IncrementalKMeans relies on quality comparison of the clustering ζ_i at t_i with that of ζ_{i-1} at t_{i-1} and performs re-clustering when required.

3.4 Summary

In this chapter we presented the core ideas of our approach on building perennial objects. Unlike conventional mining, where old data gets discarded, perennial objects may not be forgotten. Keeping this restriction under consideration, we have developed caching techniques to maintain only the most important perennial objects inside the memory. The ones that are deemed not important, are spilled on to the secondary storage from where they can be recalled as needed.

We have also presented a propositionalisation-based method to build perennial objects incrementally, with the help of efficient joins and aggregation of the other streams. The propositionalisation method can also adapt to the changes in the domain of nominal attributes. The final contribution of this chapter is an incremental variant of K-Means that was developed to evaluate the method for building perennial objects. The in-

cremental K-means operates on the propositionalised stream of perennial objects and can handle their dynamic nature. The evaluation of these methods is presented in the following chapter (Chapter 4).

CHAPTER 4

Evaluating Incremental Propositionalisation

In this we are going to evaluate the algorithm IncrementalPropositionalisation. The complete IncrementalPropositionalisation is made of several components, i.e., algorithms for the *maintenance of the cache* over perennial objects, algorithms for the *space adjustment* of the previously unseen values of a nominal attribute, and the *transformation of multiple interrelated streams*. To evaluate the complete algorithm we have used IncrementalKMeans (cf. Section 3.3).

Each component's evaluation is done under a separate section, where we describe the objectives of the evaluation, explain the evaluation measures used and a discussion on the results. Before moving onto evaluation, we first describe the datasets that we have used.

4.1 The Datasets

In this section, we provide the description of the datasets that were used in the evaluation. The description includes the name of the dataset, its schema and cardinality, number of concept drifts and their timepoints, and the availability of the dataset.

4.1.1 Synthetic Dataset: Synthetic Small

The is a rather small dataset. This dataset simulates the multiple occurrences of perennials. There are a total of 26 objects (alphabets from a to z). The learning task is to approximate the top-$N^{\mathcal{T}}$ to be cached for the next timepoint. The objects arrive repeatedly across 40 timepoints, i.e. t_1 to t_{40}. The concept drifts occur at timepoints t_3, t_{17} and t_{35} timepoints.

4.1.2 Synthetic Dataset: Zipf Dataset

This is a simple synthetic dataset which was generated according to zipf distribution. This dataset *simulates* the real world environments where

observance frequency of objects in a stream often follows zipf distribution. For example, number of purchases of the customers, number of items rated by the user and etc., often follow zipf distribution. The learning in this dataset is to evaluate the caching mechanisms (cf. Section 3.2.1) on how well the proposed mechanisms manage the perennial objects inside limited space.

The dataset is generated using the **zipf** module in the *numpy*[1] package of Python language. This module accepts only one parameter for generating the data distribution. The *head* of the data generated by the distribution lies at 0.

For generating a synthetic data stream with drifts, we generate the data by varying the shape parameter and then we manually alter the generated data (for details cf. Appendix A.1) in order to impute concept drifts. The dataset consists of 4704 unique numbers (which serve as the identifiers of perennial objects) with a combined total of 400,000 occurrences. Average occurrence frequency of a number is approximately 87. The data is distributed across 40 timepoints, i.e., t_1 to t_{40}. The concept drifts occur at timepoints t_{11}, t_{21} and t_{31}.

4.1.3 Synthetic Dataset: Ratings Dataset

This is a multi-relational dataset which was generated using the generator for multiple interrelated streams, **Multi-Gen**. We describe the generated dataset here, the details on Multi-Gen can be found in Chapter 7).

The dataset incorporates the characteristics that are exhibited by a (target) stream of perennial objects and the streams that feed it. The dataset consists of three streams, 'Users', 'Items' and 'Ratings'. 'Users' and 'Items' are perennial streams while 'Ratings' is ephemeral. 'Users' also serve as the target stream. The schema of the generated streams is shown in Figure 4.1. The learning task in this dataset is to predict each user's profile, on basis of the items that she has rated thus far, at each timepoint.

The core idea of the dataset is as follows: Each user adhere to a certain user profile. A user profile determines the affinity of a user for an item. A generated rating for an item depends on the user's affinity for the item's cluster, as set in her profile. A user may change from one user profile to another.

The ratings dataset has around 600 users. The number of transactions made by the users, vary according to zipf distribution. There were five

[1] numerical python

user profiles and 20 item profiles. The data is distributed across 20 timepoints, i.e., t_1 to t_{20}. The user profiles under go drift at timepoint t_{11}.

For the evaluation of nominal space adjustment, we use a different version of the dataset from the generator called 'Ratings2 Dataset', where we use only the 'Ratings' streams. The data consists of 40 timepoints, i.e., t_1 to t_{40}. The concept drifts occur at timepoints t_{11}, t_{21} and t_{31}.

Full specification of the dataset, i.e., the parametric settings for the generator are given in Appendix A.2.

Figure 4.1: Schema of the streams in Ratings dataset (generated by the Synthetic Generator).

4.1.4 Movie Lens Dataset

The **hetrec2011-movielens-2k**[2] dataset contains data from an online movie website where users provide ratings to the movies based on their viewing experience. This is a multi-relational dataset. Our learning task in this dataset is to evaluate the caching mechanisms (cf. Section 3.2.1)[3].

The dataset contains 2113 users, approximately 10,000 movies and approximately 850,000 ratings provided by the users. The dataset contains the daily ratings data from *09/1997* to *01/2009*. We have distributed the dataset into 140 timepoints, where each timepoint corresponds to a month.

The dataset follows zipf distribution for both the number of ratings provided by the users and how many ratings are associated with a particular movie (see Figure 4.2). Additionally with the dataset the information about each movie (i.e., genre, cast, location, and etc.) is provided in separate tables.

4.1.5 Financial Dataset

The Financial dataset of the PKDD 1999 Challenge contains data on bank customers. This is a multi-relational dataset and the tables in the Financial dataset depict the activities of bank customers who have been granted

[2]http://www.grouplens.org/node/73

[3]The original learning task was to predict the ratings of the movies and recommend them to the users.

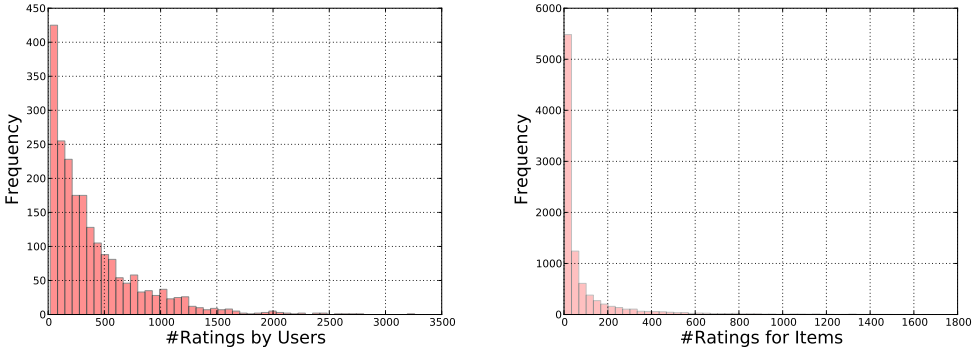

Figure 4.2: Statitics for Movie Lens Dataset (a) Ratings for users, (b) no of ratings for movies

a loan and are paying it back in the period *01/1993* to *12/1998*. The data are labelled and time-stamped. The schema of the tables from the financial dataset is shown in Figure 4.3, while basic statistics are depicted in Table 4.1.

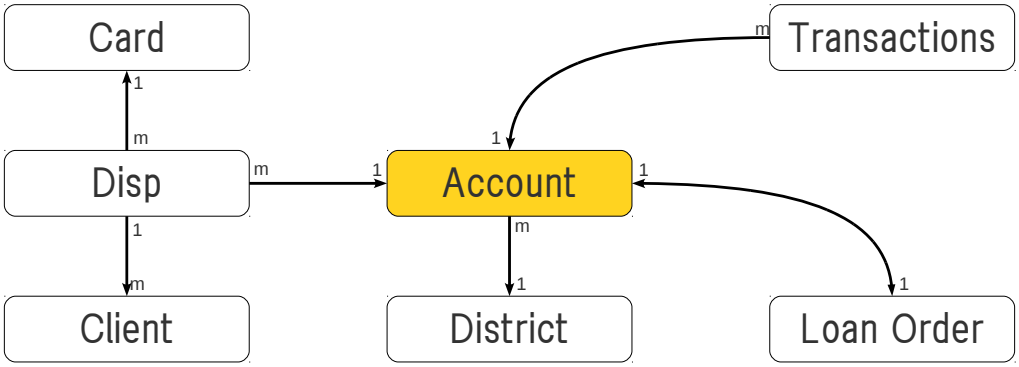

Figure 4.3: Schema of the tables from Financial dataset.

The learning task is to predict whether a customer (represented by her account) will default in paying back her loan. A loan is associated with an account which in turn may belong to one or more clients. The type of credit cards and orders made through each account are recorded, however the main load comes from the transaction stream. Already during PKDD Challenge classes A, C and B, D were merged into *loan-trusted* and *loan-risk*, respectively. We do the same.

This dataset puts forwards a difficult learning problem. The class distributions are not only very skewed to begin with; they also reflect the state

	Table/Stream	Cardinality	A & C	B & D
C	**Accounts**	682	606	76
	Districts	77		
	Clients	827		
	Disp	827		
	Orders	1513		
	Cards	170		
W	Transactions	191,556		

Table 4.1: Dataset statistics from Financial dataset. In the first column, C stands for streams associated with a cache, W stands for window, rest are used as static tables.

of accounts only when they have matured, i.e., class labels become applicable at a much later timepoint than when the objects were introduced.

4.1.6 Gazelle Dataset

The Gazelle dataset contains the data from an online retail store. It records the activities of the users as the browse the website. Each individual activity is request/visit to a webpage of a certain product. These requests are then grouped as sessions. Most the sessions in the dataset are done by anonymous users and only a fraction of users are the actual customers of the retail store. These activities are recorded over a period of 02-05/2000. In Table 4.2 we depict the tables of each gazelle dataset while its schema is given in Figure 4.4.

The learning task in this set is to separate session of users into groups of similar session. The original learning task in the dataset was to predict for each session whether a user would make a purchase in excess of 12 $.

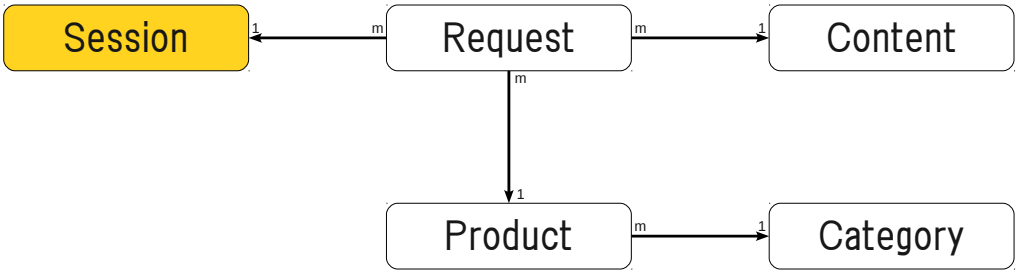

Figure 4.4: Schema of the tables from Gazelle dataset.

Table/Stream		Cardinality
W	**Sessions**	18113
	Customers	3336
W	Request	213,101
C	Products	376
C	Contents	77

Table 4.2: Dataset statistics from the Gazelle dataset. In the first column, C stands for streams associated with a cache, W stands for window, rest are used as static tables.

4.2 Evaluating Space Adjustment

In this section we evaluate Algorithm 4: *NominalSpaceAdjustment* (cf. Section 3.2.4) that accommodates previously unseen nominal values into predefined number of columns r_A. In order to do this, we first define a notion of similarity between the nominal values (cf. Section 3.2.4) and then nominal values are clustered to achieve groups of nominal values that will share a column. For clustering we use Hierarchical Flat Clustering where number of clusters to be discovered is set to $K = r_A$. We evaluate the algorithm on basis how well it can separate groups of similar nominal values. We describe the framework to evaluate the quality of discovered groups in the next section.

ID	Att1	Att2	Att3		ID	**Att1**	Att2	Att3
1	71	12	19	$\rightarrow$	1	**Bin35**	12	19
2	24	37	42	$\rightarrow$	2	**Bin12**	37	42
3	66	89	61	$\rightarrow$	3	**Bin33**	89	61
4	70	89	61	$\rightarrow$	4	**Bin35**	89	61

Figure 4.5: Numerical Value Discretization.

4.2.1 Evaluation Framework

We discretize the numerical attributes in our datasets into bins. The process of discretization is shown in Figure 4.5. This process converts the attributes into ordinal ones. For example, *Att1* with range (0,100] is discretized into 50 bins of length 2, i.e., [**Bin1**=0-2, **Bin2**=2-4, ..., **Bin50**=98-100]. We then perform propositionalisation and heirarchical clustering to cluster these nominal/ordinal bins.

Table 4.3: Experimental settings for clustering of noimnal values and discretization of the numerical attributes

Dataset	Parameters	Values
Ratings2 Dataset	number of bins	100
	$K = r_A$	25, 15, 10
	ω	1, 2, 4, 8

The experimental settings are given in the Table 4.3. Dataset is composed of varying of number of timepoints ($t \in [20, 30, 40]$) with drift occurring at timepoints that are multiple of 10, i.e., at $t_{10}, t_{20}, \ldots$ etc. One attribute from the dataset is discretised according to the specifications given in Table 4.3. We compare the strategies with a varying window size $\omega = 1, 2, 4, 8$.

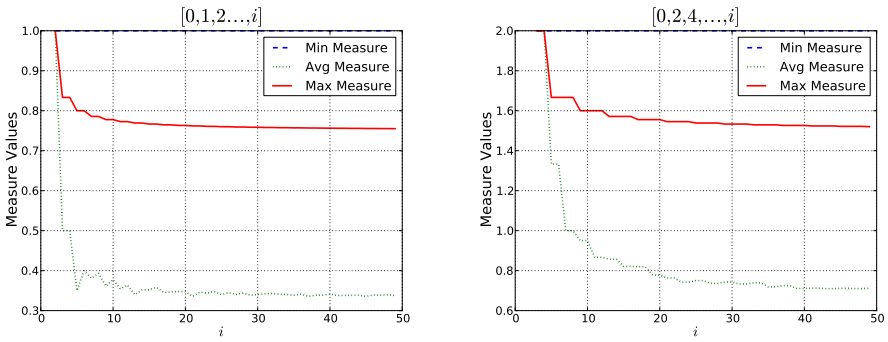

Figure 4.6: Various measures for evaluating the clusters of nominal values for a cluster containing indexes: (left) $\{0, 1, 2, \ldots, i\}$ and (right) $\{0, 2, 4, \ldots, i\}$

Evaluation Measures To evaluate the quality of the clustering of discretized numerical values we proceed as follows: We assume that if a cluster contains bins from consecutive intervals, it is cohesive, i.e., it contains discretised values that are similar to each other. Since an interval corresponds to a bin, we use the index number of the bins to check for proximity.

In order to compute the cohesiveness of an individual cluster we experimented with the following measures:

Let e, e' be two values such that $e \neq e'$. We define $dist(e, e')$ using some distance function.

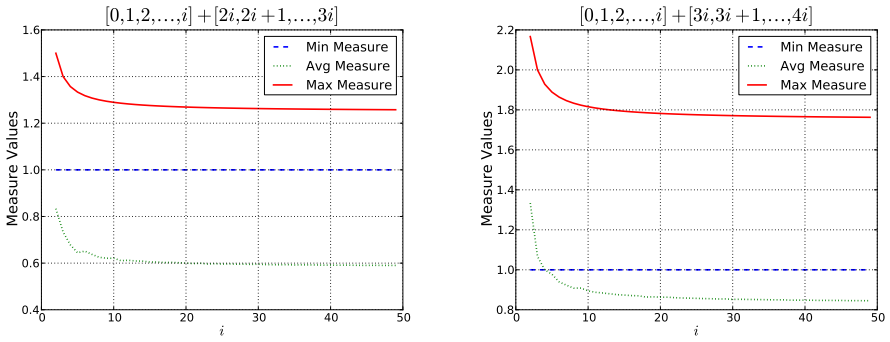

Figure 4.7: Various measures for evaluating the clusters of nominal values for a cluster containing indexes: (left) $\{0, 1, 2, \ldots, i\}$ & $\{2i, 2i + 1, \ldots, 3i\}$ and (right) $\{0, 1, 2, \ldots, i\}$ & $\{3i, 3i + 1, \ldots, 4i\}$

Minimum Measure: It computes average of minimum distance between all the bins inside the clusters.

$$MinMeasure\,(C) = \frac{\sum_{e \in C} \min_{e' \in C \setminus \{e\}} dist(e - e')}{|C|} \tag{4.1}$$

where C is the cluster of discretized bins, $e \in C$ and $e' \in C$ are the indexes of the bins from C such that $e \neq e'$.

Average Measure It computes average of average distance between all the bins inside the clusters.

$$AvgMeasure\,(C) = \frac{\sum_{e \in C} \frac{\sum_{e'} dist(e - e')}{|C|-1}}{|C|} \tag{4.2}$$

Maximum Measure It computes average of maximum distance between all the bins inside the clusters.

$$MaxMeasure\,(C) = \frac{\sum_{e \in C} \max_{e' \in C \setminus \{e\}} dist(e - e')}{|C|} \tag{4.3}$$

To get the aggregated measure for the whole clustering of different measures, we take the weighted average of the individual measures over all the clusters:

$$FinalXMeasure\,(\zeta) = \sum_{C \in \zeta} \frac{|C|}{|\zeta|} XMeasure\,(C) \tag{4.4}$$

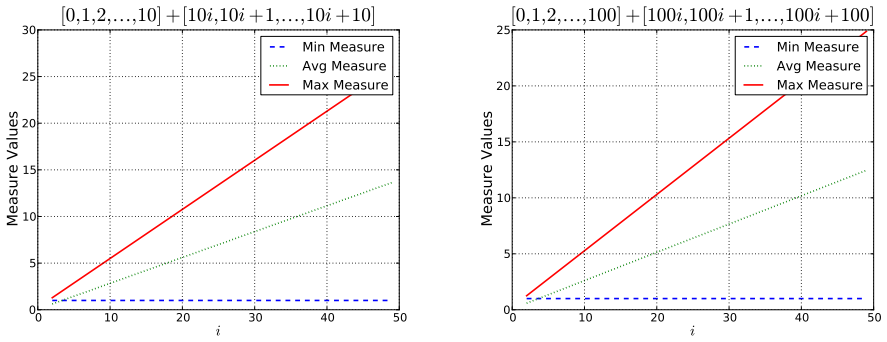

Figure 4.8: Various measures for evaluating the clusters of nominal values for a cluster containing indexes: (left) $[0, 1, 2, \ldots, 10]+[10i, 10i+1, \ldots, 10i+10]$ and (right) $[0, 1, 2, \ldots, 100] + [100i, 100i + 1, \ldots, 100i + 100]$

In Figures 4.6-4.9, we plot the values for each measure when the indexes of the bins inside the clusters vary in order to *select/choose* the appropriate measure from among them for evaluating the nominal value clusters.

In Figure 4.6, we show the plots for each measure for a cluster that contains bins with equal distance between them. When the distance between the bins is increased from one (see left of Figure 4.6) to two (see right of Figure 4.6), all measures report an increase. As the number of bins inside a cluster, the $MaxMeasure$ and $AvgMeasure$ measures converge. We expect the measure functions to monotonically converge, where as the $AvgMeasure$ does not show strictly monotonic behaviour.

In Figure 4.7, we show the plots for each measure for a cluster that contains continuous bins with a gap between them. We expect the measure functions to return larger values when the gap between the continuous bins is increased. However, $MinMeasure$ stays constant even when gap between the continuous values is increased from i on left of the Figure 4.7 to $2i$ on right of the Figure 4.7. This effect is more visible in Figure 4.8 where $MinMeasure$ remains unchanged when the gap between the values changes.

In the left of Figure 4.9, we show the plots for each measure for a cluster that contains bins with exponential indexing. All measures show increase as the number of bins are increased inside the cluster with $MaxMeasure$ being the most sensitive. In the left of Figure 4.9 the plots are show for bins with exponential indexing and the next consecutive bin. $MinMeasure$ is completely insensitive of the distance between the bins in such a cluster.

Based on the above experiments, we choose $MaxMeasure$ for evaluat-

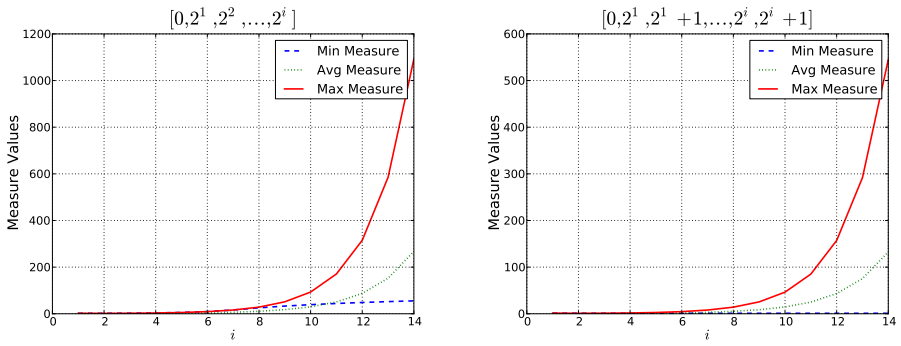

Figure 4.9: Various measures for evaluating the clusters of nominal values for a cluster containing indexes: (left) $\{0, 2^1, 2^2, \ldots, 2^i\}$ and (right) $\{0, 2^1, 2^1 + 1, \ldots, 2^i, 2^i + 1\}$

ing the quality of the clusters of numerical bins as its sensitive, stable and shows monotonic change.

We have conducted the experiments not only evaluate the quality of the clustering of the numerical bins, but also how stable are the groupings over time, i.e., whether the nominal bins constantly switch between the clusters or do they stabalise. If the clustering is not stable across time points, the discovered nominal groupings would be off little use as the propositionalised attributes that are constructed over these nominal clusters would change their definition too often to be of any utility. We use Jaccard Coefficient to check the stability of clusters of nominal bins across consecutive timepoints.

Jaccard Coefficient Jaccard Coefficient [Tan et al., 2004] assumes two clusterings ζ, ζ'. Let $f11$ be the number of records for which ζ, ζ' agree that they should be in the same cluster. Let $f10$ be the number of records that were put in the same cluster by ζ but in distinct clusters by ζ' (disagreement), and let $f01$ be the number of records put in the same cluster by ζ' but in distinct clusters by ζ (disagreement). Then, the Jaccard Coefficient is:

$$JaccardCoeff(\zeta, \zeta') = \frac{f11}{f11 + f01 + f10}$$

4.2.2 Experimental Results

In Figure 4.10, we plot the results of nominal clustering with varying window sizes $\omega = 1, 2, 4, 8$ over the synthetic dataset. The dataset consists of

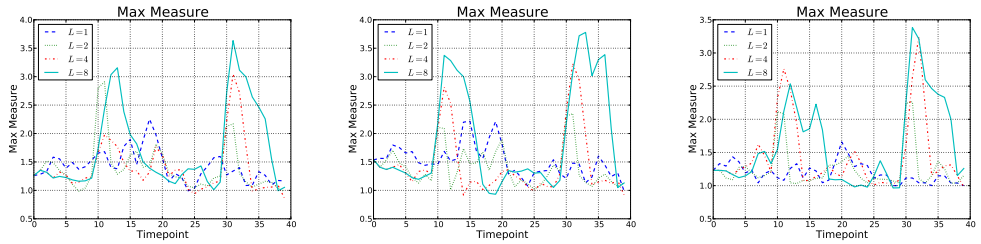

Figure 4.10: Synthetic Generator: Max Measure with n_{bins}=100 (left) K=10 (middle) K=15 and (right) K=25.

four attributes out of which we have discretised one attribute into 100 bins. The different sub-figures we have varied the number of nominal clusters $K = 10, 15, 25$.

As indicated by the plots for $MaxMeasure$, the clusters from all the experiments are generally compact, i.e., they contains bins, which are in close proximity of each other. When drift happens, all the strategies register a drop in compactness (indicated by peaks in $MaxMeasure$). After the drift the clustering quality for the smaller window sizes recovers first. The quality for the $\omega = 8$ recovers the last just in time for the next drift.

In the plots, only two drift points are visible, i.e., at timepoints t_{10} and t_{30}. All the strategies seem to missed the second drift which occurs at timepoint t_{30}. Only examining the dataset, we discovered that although there was indeed a drift at timepoint t_{20} but the clustering from the before the drift was valid for the new data, as well.

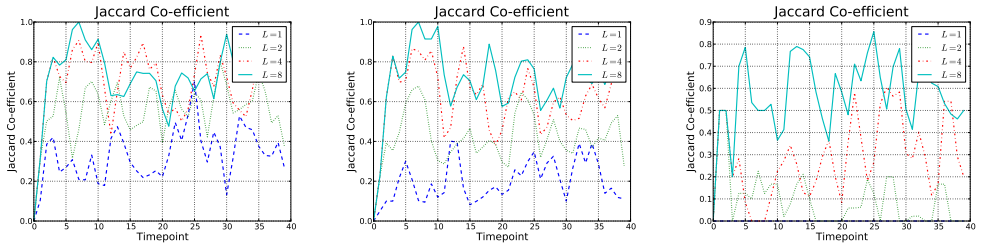

Figure 4.11: Synthetic Generator: Jaccard with n_{bins}=100 (left) K=10 (middle) K=15 and (right) K=25.

The clustering quality is marginally better for larger K values. One obvious reason can be the data compression. Compression is responsible in part for this. In a data streams we often come across skewed data distributions, where certain nominal values have higher frequency of occurrence than the others. We encounter something similar with our bins

where certain bins occur more often. When the bins are propositionalised, the ones with lower counts tend to act as noise. At larger values of, the flat hierarchical clustering isolates them into separate clusters and they do not disturb the quality that much. Smaller K values forces the clusterer to merge nominal bins together and while merging it merges these noisy values which are seemingly far apart.

In Figure 4.11 we plot the values from of Jaccard Coefficient. The strategies with larger window sizes are generally more stable than those that have a smaller window sizes. The stability of clustering decreases as the number of clusters is increased. When the Jaccard plots with $K = 25$ (see right of Figure 4.11) are juxtaposed with the $MaxMeasure$ plots (see right of Figure 4.10), it becomes apparent that the marginally better performance of the strategies with larger $K = 25$ are at a cost of stability.

4.3 Comparing Different Caching Strategies

In this section we evaluate the methods that keep a cache of perennial objects under limited space. We provide our experimental settings and the measures that we have used in the following.

Table 4.4: Description of caching strategies

Name	**Symbol**	**Description**
Baseline	**B**	infinite memory knows future
CacheUpdate	**R1**	infinite memory does not know future
SmallCache	**R2**	limited memory does not know future
SpaceSaving+	**SS**	limited memory does not know future

4.3.1 Evaluation Framework

To evaluate the different mechanisms (cf. Section 3.2.1) for caching perennial objects, we have devised an extensive framework. We evaluate the algorithms on how well they can manage memory and time limitations and maintain a good sample of objects from the perennial stream. In order to do that we have devised various strategies and compared them against each other. The strategies are described in Table 4.4.

Strategy *Baseline* is our reference strategy. It has infinite storage capacity and can anticipate which objects are going to arrive next. *CacheUpdate* is our cache update algorithm which operates without knowing the future (cf. Section 3.2.1). However, it has infinite resources and can store the exact number of references an object receives. *SmallCache* is an improvement on *CacheUpdate* strategy (cf. Section 3.2.1). It tries the perennial objects that will be referenced and caches them. *SpaceSaving+* is a variant of the "space saving" by Metwally et. al. [Metwally et al., 2006]. Its primary purpose is to keep approximate counts of objects seen thus far in the stream and maintain the top-N objects seen thus far. Unlike the original "space saving" algorithm, we apply decay on the counters maintained by SpaceSaving+ at the end of each timepoint.

Table 4.5: Parameters for caching strategies

Dataset	**Parameters**	**Values**
SytheticSmall	$N^{\mathcal{T}}$	[3, 5, 7]
	M	[6, 10, 14]
	β	[1, 1.1, 1.2, 1.5, 2, 5]
Zipf Dataset	$N^{\mathcal{T}}$	[50, 100, 200]
	M	[125, 200, 300]
	β	[1, 1.2, 1.5, 5]
MovieLens Dataset	$N^{\mathcal{T}}$	100, 500
	M	200, 100
	β	1, 2
Financial Dataset	$N^{\mathcal{T}}$	50, 100, 200
	M	100, 200, 400
	β	1.2, 2

In Table 4.5 we present the experimental settings for experiments that we have conducted for evaluating the cache mechanism. N is the size of the cache over the perennial stream. The parameter M is size of the counters maintained to approximate the N objects to be cached. This parameter is only applicable for the strategies SmallCache and SpaceSaving+, whereas the strategies Baseline and CacheUpdate have infinite resources. SmallCache uses an additional parameter M_{soft}. For all the experiments, $M_{soft} = 1.5 \times M$.

We have used the commonly used evaluation measures which are; accuracy, hits, runtime and memory. We list these measure in Table 4.6. The accuracy in our experiments means how well strategies can approximate the top-$N^{\mathcal{T}}$ objects for the next timepoint. The accuracy of the top-N sets

of objects is measured against the baseline because it has the knowledge of future and knows the exact top-N objects to be cached. Therefore, we do not plot the accuracy for the baseline. It is explicitly assumed to be 100%.

Table 4.6: Evaluation measures for caching strategies

Measure	**Description**
Accuracy *avg*(Accuracy)	The accuracy of top-N objects of a strategy wrt. to the Baseline For the aggregated experiments, we compute the average of the accuracies across all the timepoints.
Hits *sum*(Hits)	The number hits recorded by a strategy for the cached objects, i.e., top-N objects. For the aggregated experiments, we compute the sum of the hits across all timepoints, i.e., total number of hits recorded by a strategy in a single run of an experiment.
Runtime *sum*(Runtime)	The execution time of a strategy. For the aggregated experiments, we compute the sum of the runtimes across all the timepoints.
Memory *max*(Memory)	Units of memory used by a strategy. For the aggregated experiments, we compute the max memory consumed by a strategy at any in a single run.

In Table 4.5 regarding the experimental settings, the parametric values within parenthesis imply that we have also conducted aggregated experiments. For such an experiment, we vary the value of one parameter while others are kept constant. Unlike, the single run experiments, where we show the values of a measure as a line plotted against time on X-axis, the results of these experiments are depicted as bar charts. On X-axis, we show the different values for a parameter and the bars are the aggregate values of the individual measures across all timepoints (see Table 4.6 for aggregation specification for each measure). All aggregated experiments are aggregated over 10 runs for each parameter setting (unless stated otherwise). The randomisation in the experiments is achieved by shuffling the order of incoming objects with a timepoint.

4.3.2 Experimental Results

Here, we present the results of our experiments for each dataset. The aggregate runs for the dataset are presented after the individual runs.

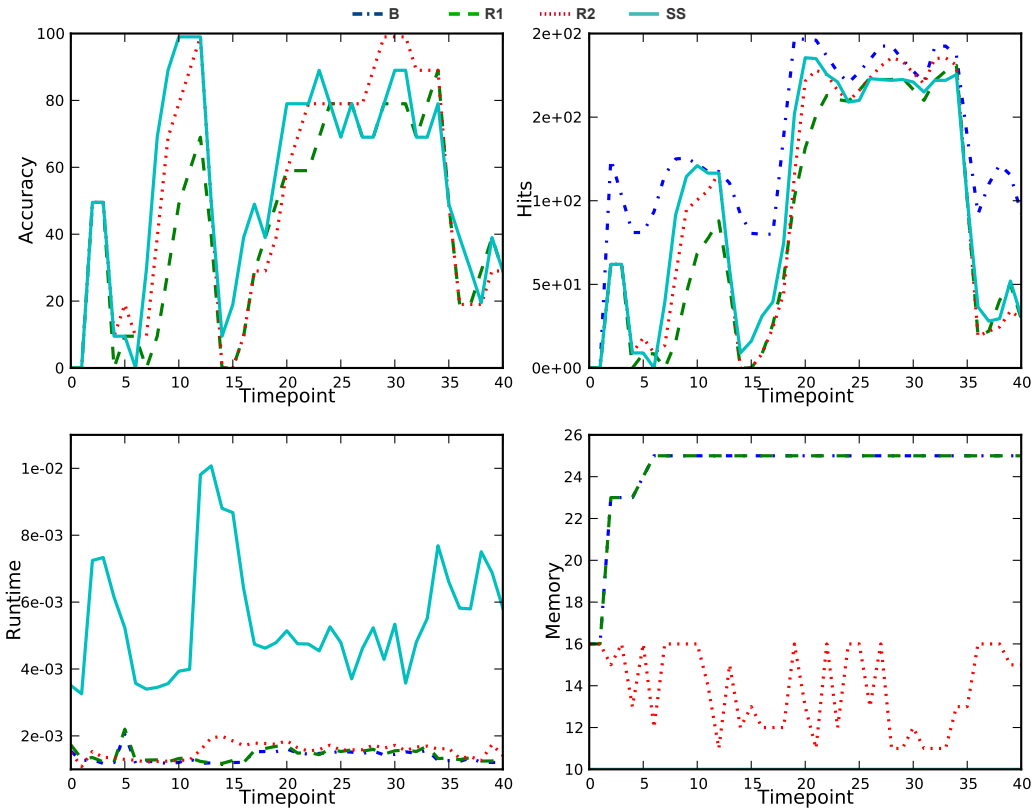

Figure 4.12: Plots for SmallDataset: $N^{\mathcal{T}}$=5, M=10, β=1

In Figure 4.12 we present the individual run over the SytheticSmall dataset with the parameters $N^{\mathcal{T}} = 5, M = 10, decay = 1$. Even when the decay is set to 1 (i.e., no forgetting), all the strategies are able to recover, albeit slowly (Hits count in Figure 4.12). For strategies R2 and SS recovery is relatively faster than that of R1. This is primarily due to the fact that R2 and SS maintain an approximation on the number of objects missed so far. Whenever a new object arrives, they associate the number of misses with the arriving object. In other words, they both are optimistic strategies that initialises a previously unseen object with an upper bound of the number of objects missed thus far. Therefore, when the drift occurs, new objects get slightly inflated counts and strategies recover. R1 on the other hand maintains only the exact counts. Whenever a new object arrives, unlike R2 and SS it is initialised with a count of zero. In order for the new object to brought into the cache (or among top-$N^{\mathcal{T}}$ objects) and has to wait until the accumulated counts of the new object become greater than the counts

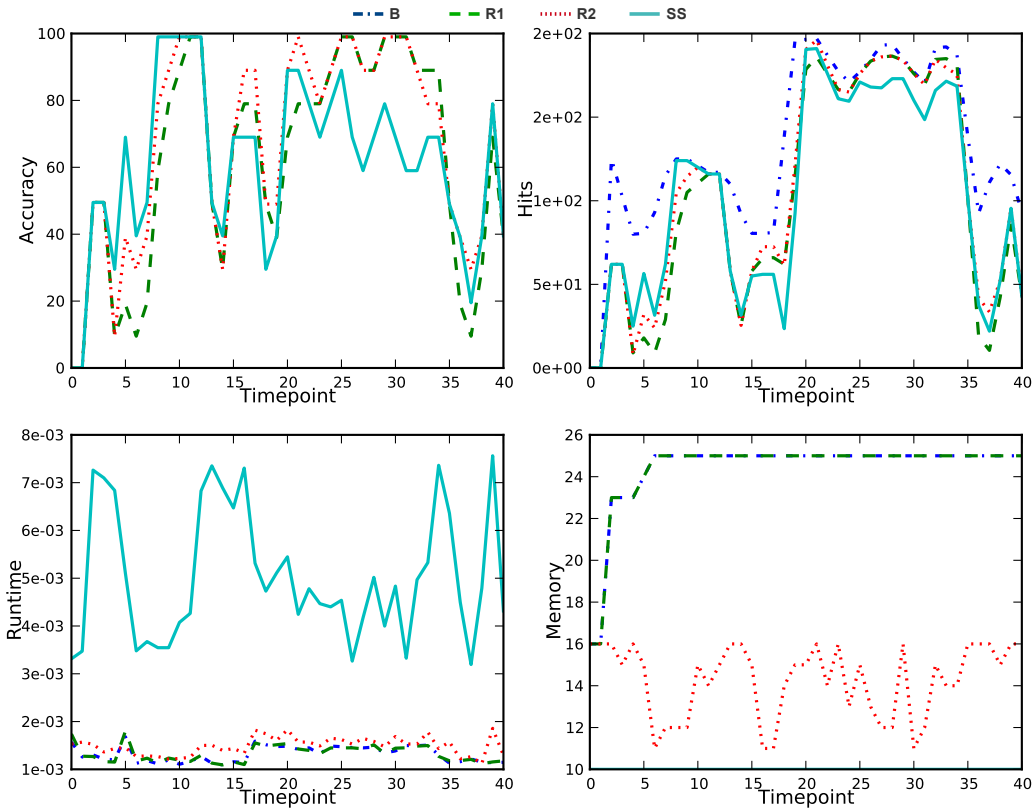

Figure 4.13: Plots for SmallDataset: $N^{\mathcal{T}}$=5, M=10, β=2

of those that are already in the cache[4]. It is the last to recover from the drifts at t_3 and t_{17}. For the last drift moment, no strategy is able to recover because there are not enough any more.

One peculiarity in Figure 4.12 is that although SS is the first one to recover after the drifts at t_3 and t_{20} (see Hits and Accuracy in Figure 4.12) but other strategies are able to catch up with it. R2 in particular, catches up with SS at t_10 and even surpasses it at t_{22}. The optimistic behaviour of SS works well when the data is volatile after the drift but during stable phases, it over estimates the counts for even incoming noise and thus suffers.

In Figure 4.13 we change the parameter value for decay to $\beta = 2$. We observe a speed-up in recovery time for all the strategies after the drift. This validates our earlier claim that the slowly recovery of R1was primarily due to its remembrance of old accumulated counts for the objects which

[4]R1 or CacheUpdate weigh the importance $G()$ of each perennial object based on the number of references it receives and then caches the top-$N^{\mathcal{T}}$ objects.

makes the recovery hard. Setting a higher value for the decay makes it possible for the strategies in general and R1 in particular to forget the historical counts quickly.

Another notable change is the noticeable reduction in the run times for the strategy SS. The strategy SS reports the reduction in the run times because in the presence of decay, the old objects get outdated quickly rather SS doing approximations to overcome their counts. For strategy R2, results in slightly less memory usage. However, we will explore this further for R2.

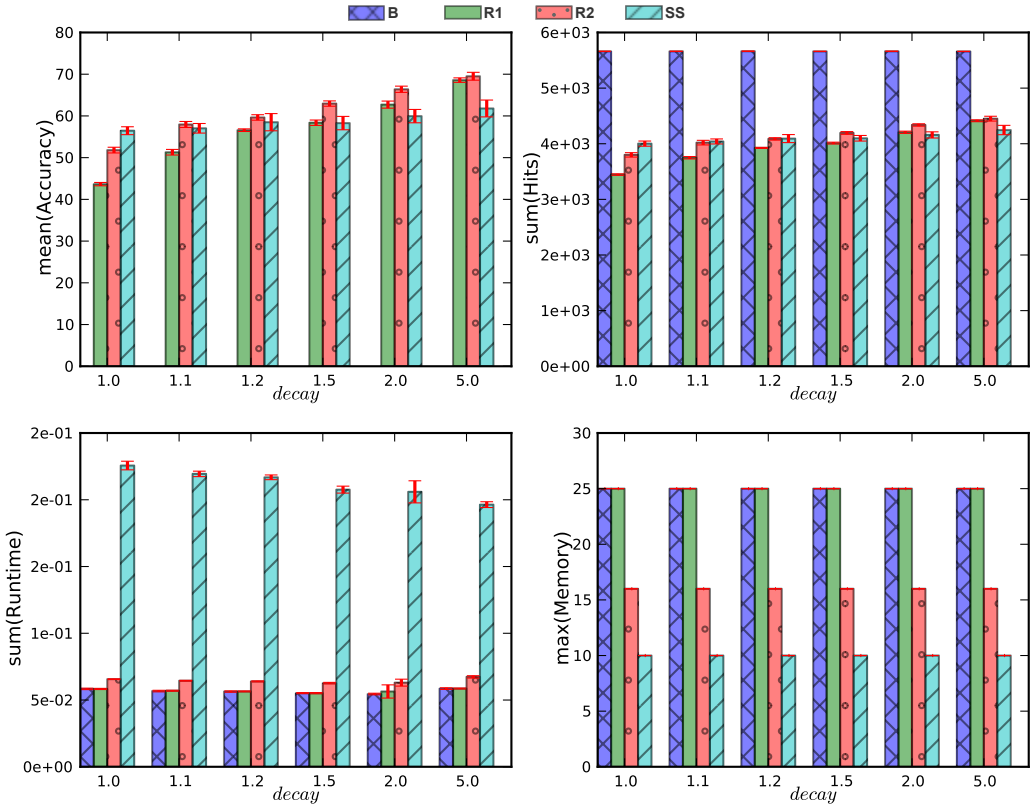

Figure 4.14: Agg. Plots for SmallDataset: $N^{\mathcal{T}}$=5, M=10 and varying β

In Figure 4.14 we present the aggregated experiments where we vary the decay parameter with values $\beta = 1, 1.1, 1.2, 1.5, 2, 5$. We can see that the average accuracy for SS strategy rises as we increase the decay till $\beta = 1.5$ and then becomes stable. The run times for the strategy SS also decreases with an increase in the decay. For R1 and R2 the mean accuracy keeps dropping as the decay is increased. On the other hand, the hits gradually increase as the decay is increased for all the strategies. The best hit

counts are obtained for strategy R2 with decay value of $\beta = 5.0$. R2 also reports the best average accuracy for the same decay value. Although R2 has slightly higher memory requirements, but it has the ability to adjust its space requirements and is suitable for scenarios where resource awareness is important. Its run times are several times lower compared to SS while their complexity is the same. R2 benefits from its ability to use the additional memory provided by M_{soft} (cf. Section 3.2.1), which greatly reduces its computational overhead.

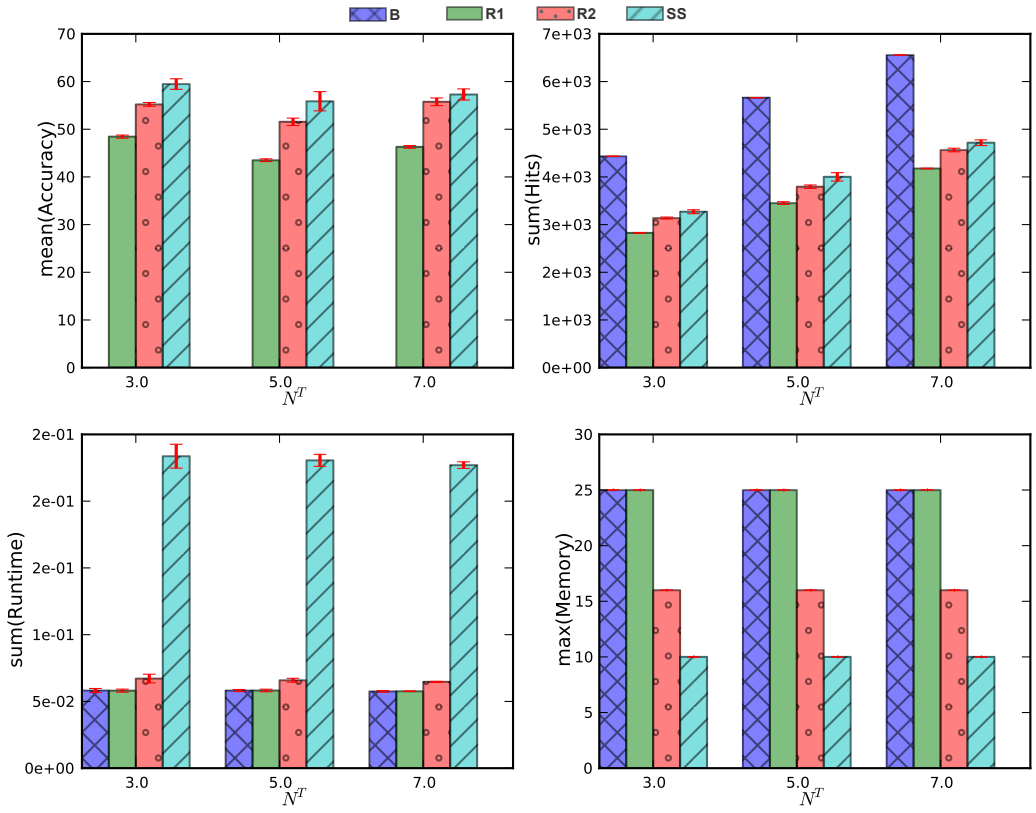

Figure 4.15: Agg. Plots for SmallDataset: M=10, β=2 and varying $N^{\mathcal{T}}$

In Figure 4.15 we plot the results for the experiments by varying the parameter $N^{\mathcal{T}} = 3, 5, 7$. In these experiments we have set the decay at $\beta = 2$. We have already seen in Figure 4.14 a decay greater than 1 benefits all the strategies in terms of number of hits. In a streaming environment setting up a decay is always desirable. With a varying $N^{\mathcal{T}}$ ($N^{\mathcal{T}}$ is the size for the object cache) all strategies report a drop in accuracy but the number of hits registered by each is increased.

In Figure 4.16 we plot the results for the experiments by varying the

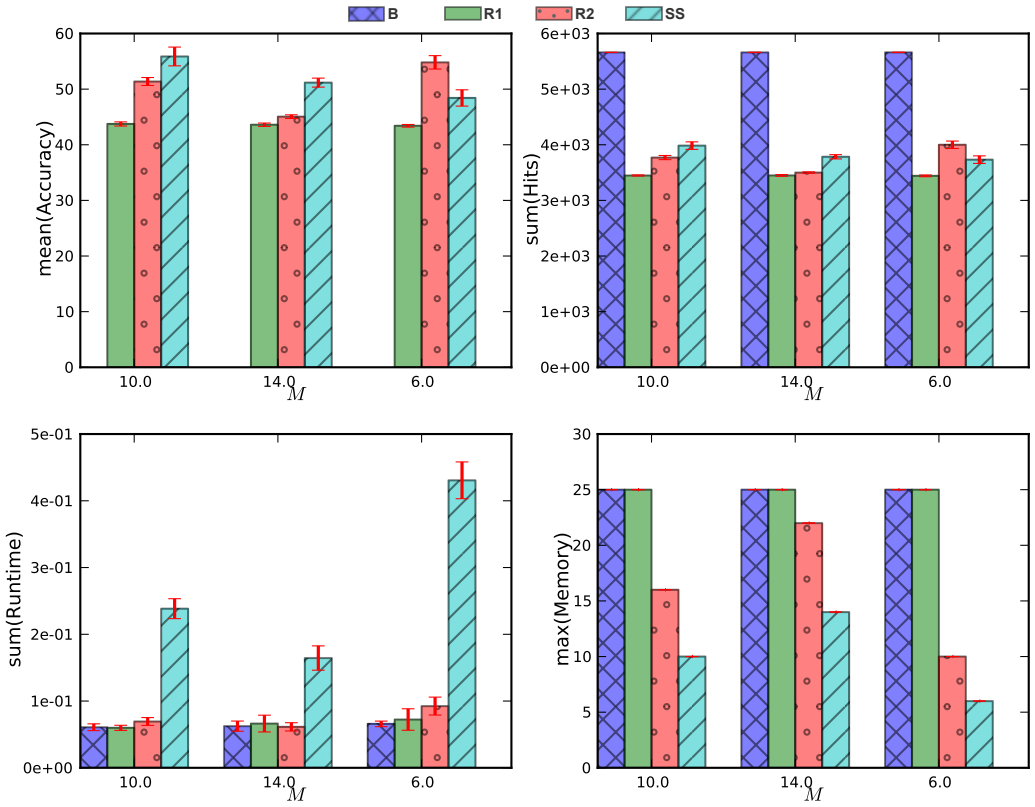

Figure 4.16: Agg. Plots for SmallDataset: $N^{\mathcal{T}}$=5, β=2 and varying M

parameter $M = 6, 10, 14$. Varying the parameter M doesn't have any effect on the space requirement strategies B and R1 as they have infinite space. The accuracy for strategies R2 and SS increases as more memory is made available to them and both also report a noticeable drop in their run times. Additional experiments with individual runs are attached in the Appendix A.3.

In Figure 4.17 we present results for the dataset Zipf Distribution with the following parameters: $N^{\mathcal{T}} = 100$, $m = 200$ and $\beta = 5$. As stated earlier we have imputed 4 concept drifts in the dataset (cf. Section 4.1.2), which occur at timepoints $t_0 - t_9$, $t_{10} - t_{19}$, $t_{20} - t_{29}$ and $t_{30} - t_{39}$. Number of new objects that appear at each drift moments are approximately 800, 1300, 1000 and 1700.

All the strategies report a drop in performance when the drift occurs. However, because of a stronger decay, i.e., $\beta = 5$, all react and recover quickly with almost similar speeds. The strategy R1 reports the best performance in comparison with the baseline both in terms of accuracy and

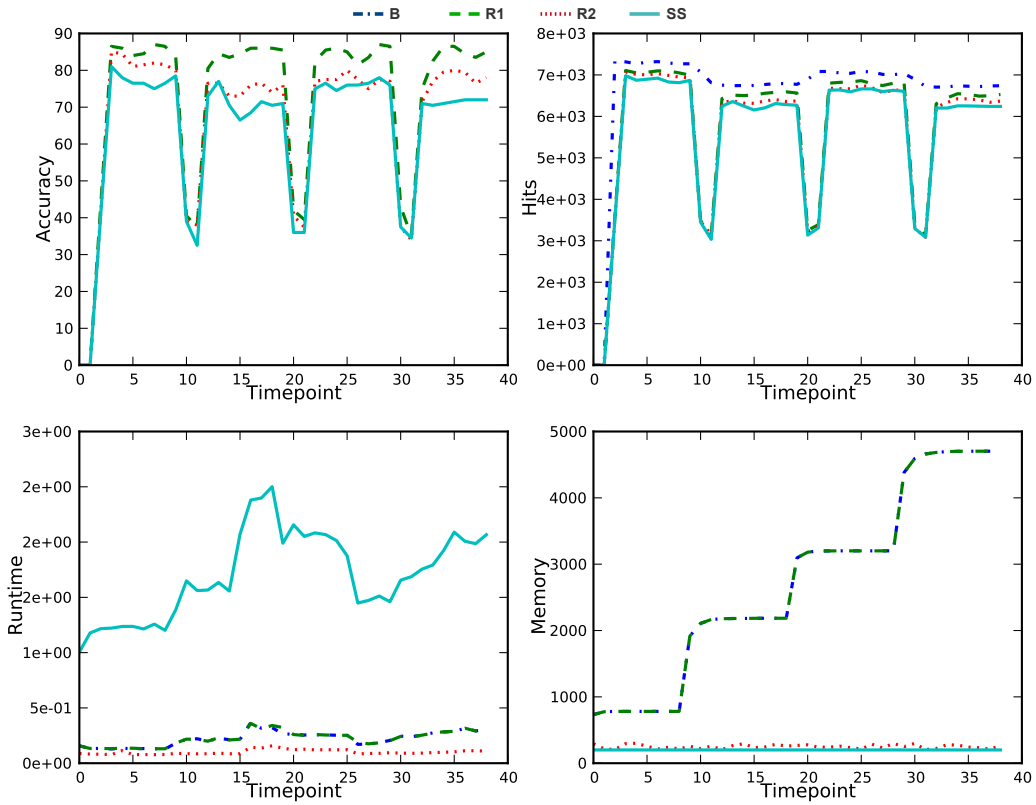

Figure 4.17: Plots for Zipf: $N^{\mathcal{T}}$=100, M=200, β=5

number of hits achieved. It is followed by the strategy R2 and the strategy SS ranks last.

One important observation one can draw (from Figure 4.17) is that SS performance grows competitive towards R2 when the overall number of objects in the drift moments is low, i.e., at the first drift moment when there are around 800 unique objects and at the third drift moment when there are around 1000 unique objects. For second and fourth drift moment when number of objects increases to 1300 and 1700, respectively, the performance of SS is visibly inferior. A similar effect can also be noted from the run times for SS, where for the second and fourth drift moment it requires more time for computation than at the first and third drift moments. R2 apparently does not show any such tendency, its performance is independent of the number of the objects.

R2 shows the best run times, even better than the strategies with unlimited space. We have investigated this more in Figure 4.18 to check if it can be explained or is it just an artefact.

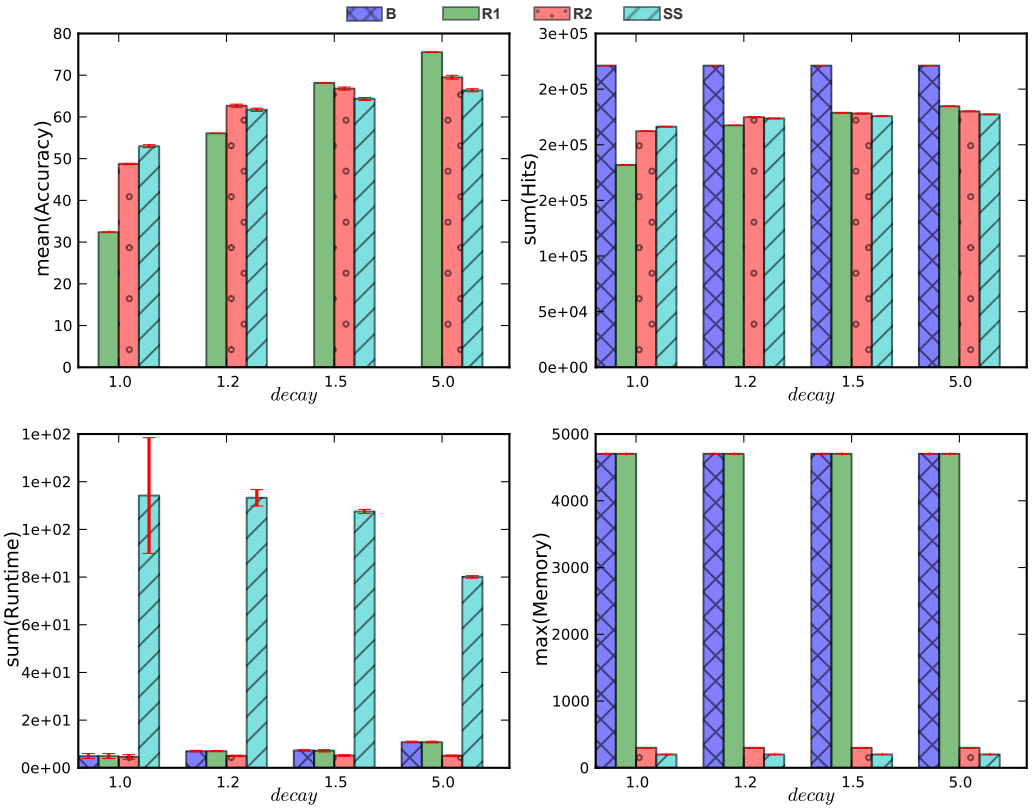

Figure 4.18: Agg. Plots for Zipf: $N^{\mathcal{T}}$=100, M=200 and varying β

In Figure 4.18 we plot the aggregated experiments for the parameters $N^{\mathcal{T}} = 100$, $M = 200$, while the decay was varied with the following values: $\beta = 1$, 1.2, 1.5, 5. Continuing from last observation, we find out that the better run times for the strategy R2 in the experiment in Figure 4.17 is not an artefact. We consistently get better run times for R2. Moreover, the run times for the strategies B and R1 gradually increase as the value for decay is increased. There is nothing in the data that might suggest this behaviour except that there are a large number of objects. At the end of each time, all the strategies make one pass over the objects stored in the memory to enforce decaying. For the strategies B and R1, with their infinite memory, it requires a lot of effort, which is one of the reason for their high run times. Moreover, with a high value for the parameter β, the objects towards the tail part, i.e., the ones with low counts become too fragile. Because of this fragility, the change in order becomes very frequent, which explains why they report higher run times.

The effect of varying the decay parameter β is similar to what we have

observed in the experiments with small dataset (see Figure 4.14). The run times for SS are inversely proportional to the decay. With a higher decay, the accuracy of the strategies for top-$N^{\mathcal{T}}$ cached objects towards the baseline B becomes consistent with the number of hits that are generated by them. With higher decay values, the strategies also generate far more hits than those generated for lower values. Additional experiments with individual runs for the decay for the Zipf dataset are attached in the Appendix A.3.

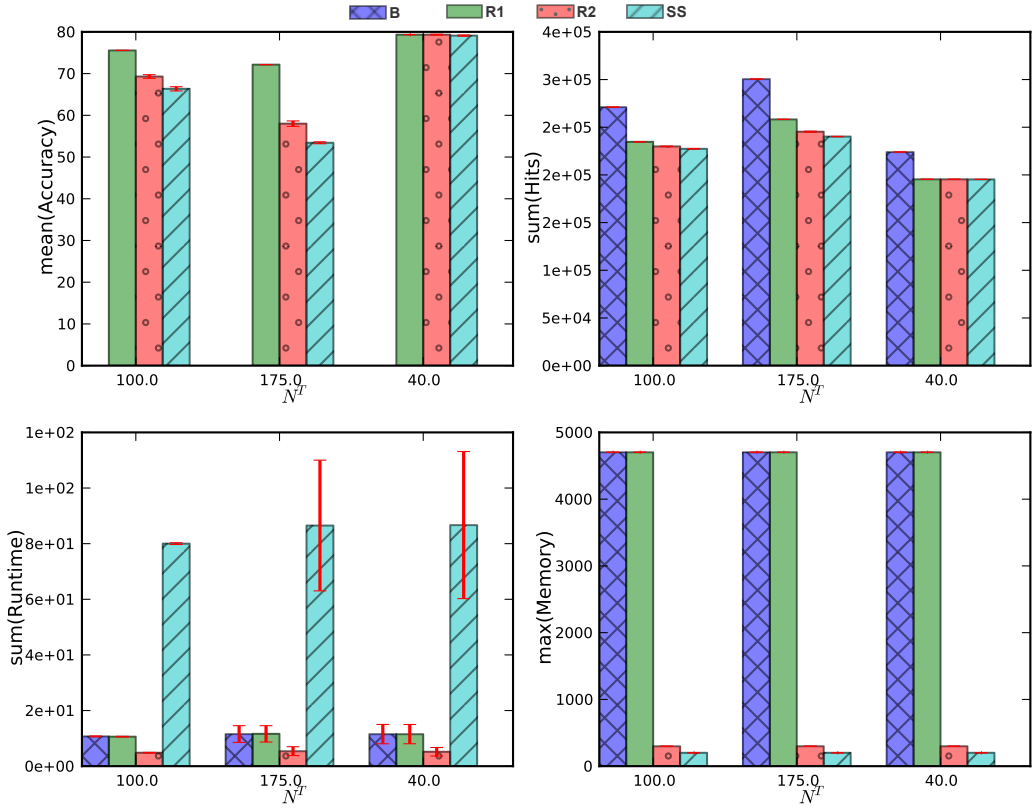

Figure 4.19: Agg. Plots for Zipf: M=200, β=5 and varying $N^{\mathcal{T}}$

In Figure 4.19, we present the aggregated experiments for the Zipf dataset with varying values for the parameter $N^{\mathcal{T}}$. For a smaller value of $N^{\mathcal{T}}$, the strategies R2 and SS report high accuracy. But their accuracy gradually decreases as is $N^{\mathcal{T}}$ increased while the number of hits generated by them increases as they can accommodate more objects.

The previous observation suggest that the strategies R2 and SS are tailored to approximate the "head" part of Zipf distribution, where the difference with respect to the tail is maximum. When the are forced to accom-

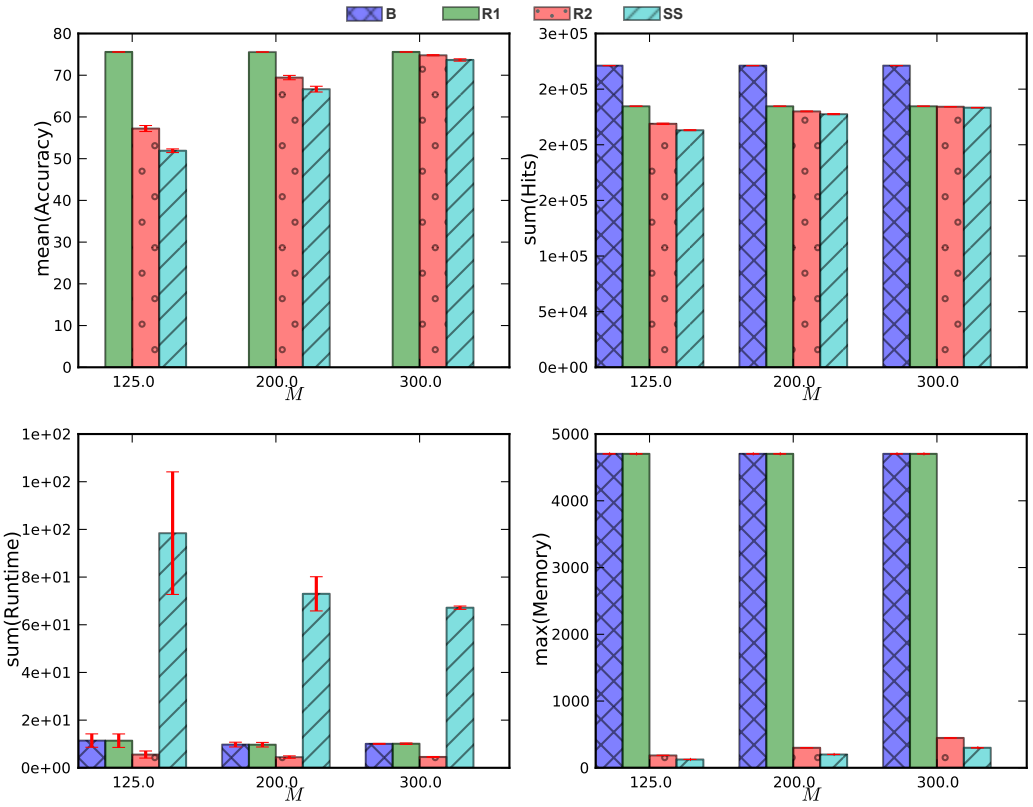

Figure 4.20: Agg. Plots for Zipf: $N^{\mathcal{T}}$=100, β=5 and varying M

modate objects that are further down the head, they run into problems because of their over optimistic behaviour and SS suffers more of the two.

In Figure 4.20, we plot the results of the aggregated experiments for the Zipf dataset with varying values for the parameter M. As more resources are made available for the strategies with limited space (i.e., R2 and SS), their approximation for top-$N^{\mathcal{T}}$ objects increase as indicated by their increasing accuracy and number of hits. They grow increasing competitive towards the strategy R1 both in terms of accuracy and the number of generated hits and demonstrate their ability to make an efficient use of the limited space available to them. Both R2 and SS also report decrease in their run times, which are noticeable for only SS in the Figure 4.20. R2 shows a marginally better performance in terms of accuracy and number of hits, while its are many fold faster than those for the strategy SS.

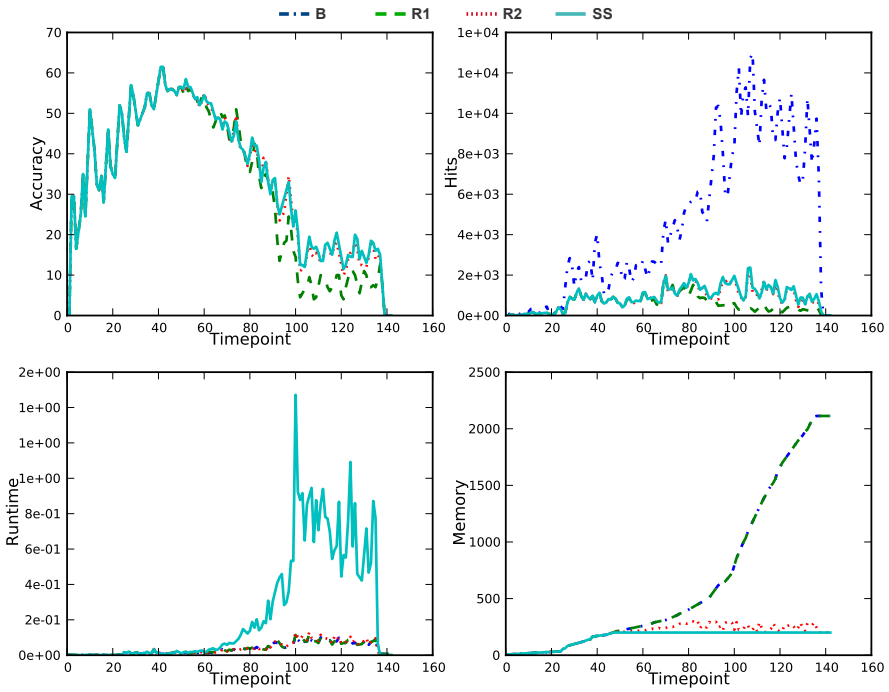

Figure 4.21: Plots for Movie Lens: $N^{\mathcal{T}}$=100, M=200, β=1

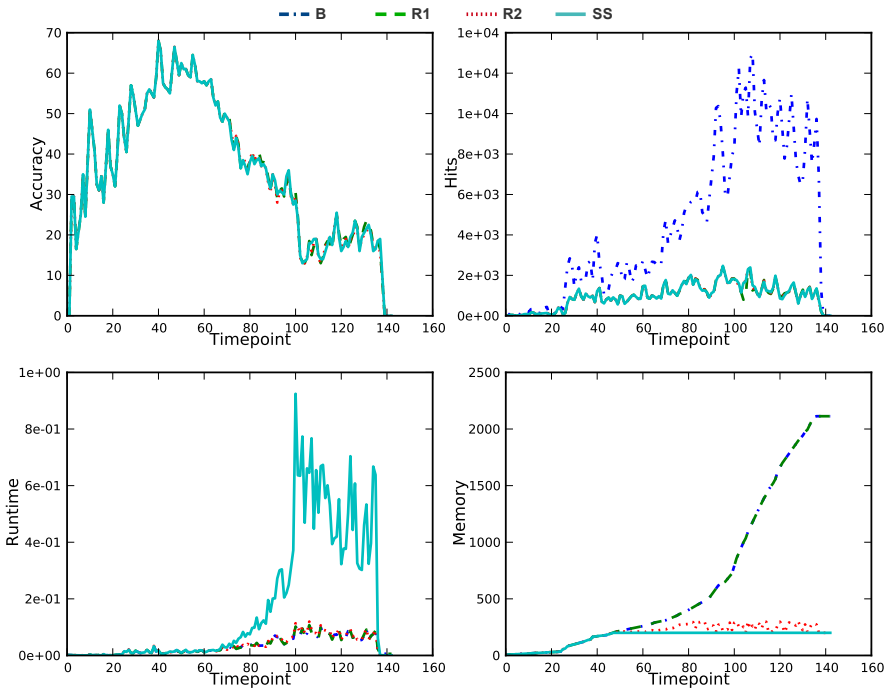

Figure 4.22: Plots for Movie Lens: $N^{\mathcal{T}}$=100, M=200, β=2

In Figure 4.21, we plot the results for the movie lens dataset with parameters $N^{\mathcal{T}} = 100$, $M = 200$ and $\beta = 1$. The accuracy for the limited space strategies R2 and SS completely breaks down after timepoint t_{80} in relation to the strategy R1. Whereas the number of hits generated by R2 and SS are higher during this break down period. That means they are able to recover because of their optimistic approximation behaviour. This is a clear indication of the presence of the drift in the data.

In Figure 4.22, we set the decay parameter $\beta = 2$. We immediately see visible improvements in terms of accuracy. The number of hits increase but not so much. Even after setting the $\beta = 5$, the improvements were minimal.

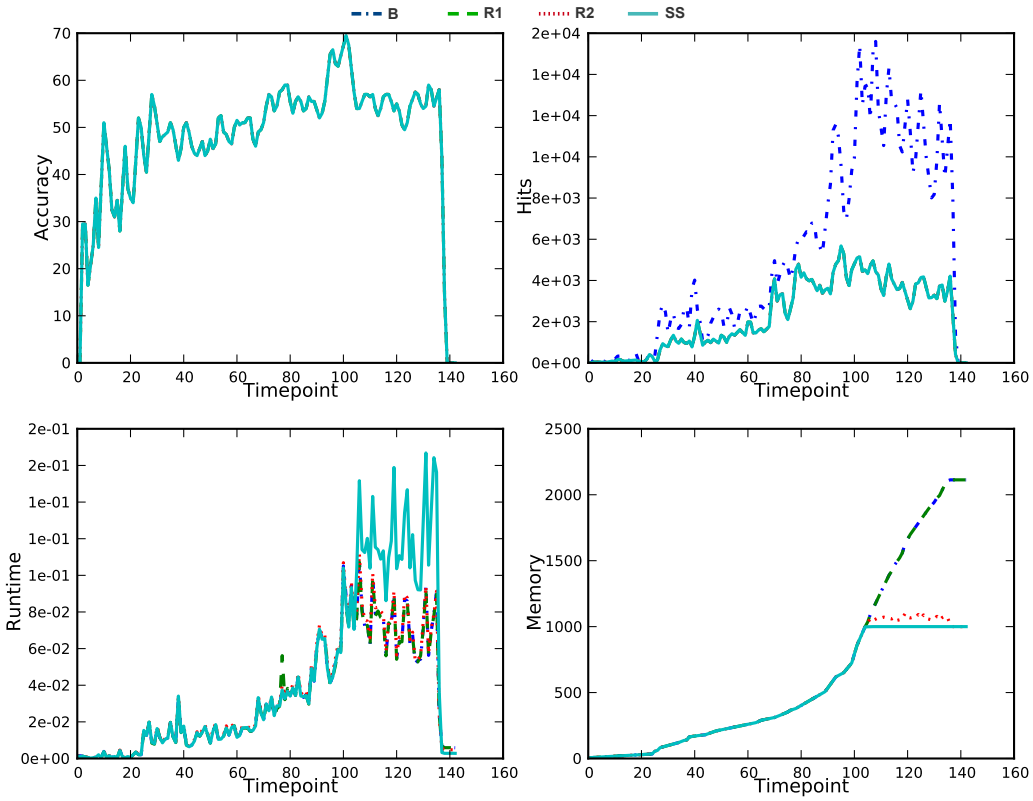

Figure 4.23: Plots for Movie Lens: $N^{\mathcal{T}}$=500, M=1000, β=2

One possible reason for such a low accuracy and so few number of hits is that the size of the cache $N^{\mathcal{T}}$ we have use is too small for the strategies to make a good approximation. In Figure 4.23, we increased the size of the cache to $N^{\mathcal{T}} = 500$ and keeping $\beta = 2$. The accuracy increases for all the strategies but the number of hits merely doubles in response to a *five-fold*

increase in the size of the cache. In comparison the strategy B registers more than the double amount of hits.

On exploring the dataset, we found the reason for such an erratic behaviour of the strategies R1, R2 and SS. In the dataset there are almost 600 users who appear only *once* and makes approximately 68K ratings. Moreover, there are 600 additional users who appear for two to five timepoints. The number of ratings generated by them are approximately 150K and these too are concentrated in one or two timepoints which may be far apart. For any strategy that has no knowledge of the future, it is extremely difficult to register hits for users with such skewed appearance and impossible for users with only a single appearance. This explains the low number of recorded hits.

However, in all the experiments on the movie lens dataset (cf. Figures 4.21-4.23), the strategy R2 registers similar performance in terms of accuracy, hits and run times in comparison to SS, with only marginally higher resource requirements.

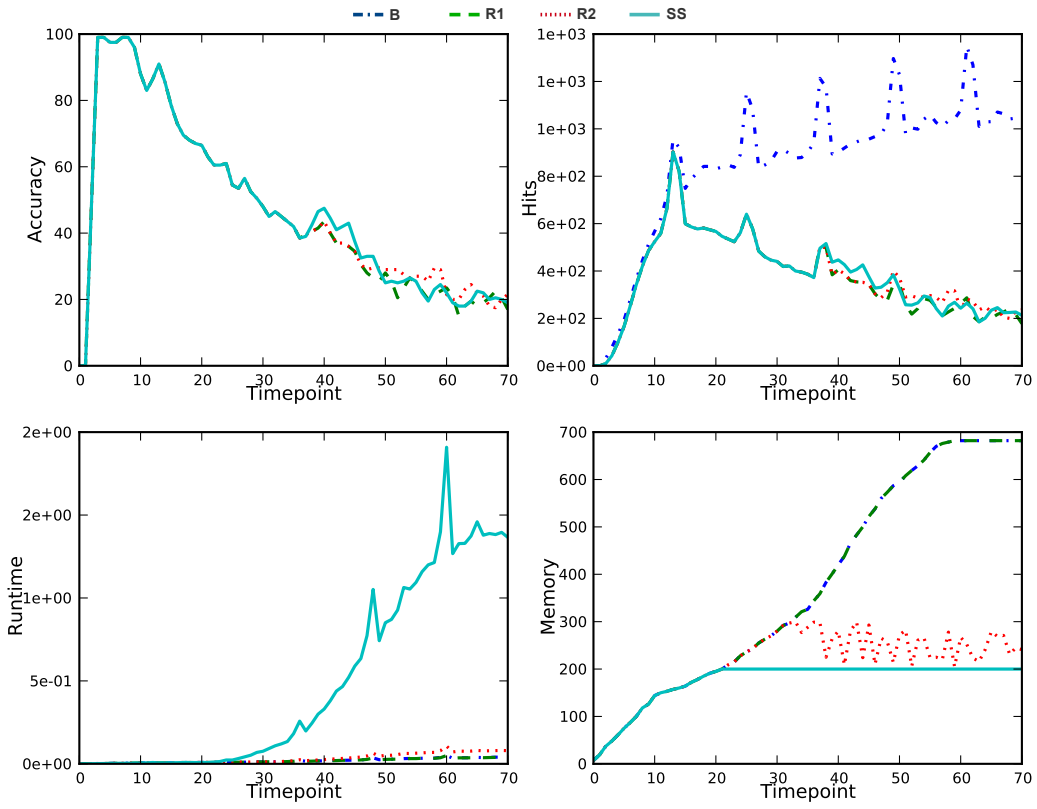

Figure 4.24: Plots for Financial: $N^{\mathcal{T}}$=100, M=200, β=1

In Figure 4.24, we plot the results for Financial dataset with the parameters, $N^{\mathcal{T}} = 100$, $M = 200$, $\beta = 1$. All the strategies start promisingly. But after t_{10} the performance gradually degrades. This indicates a very strong presence of drift and no strategy apart from B is able to capture it.

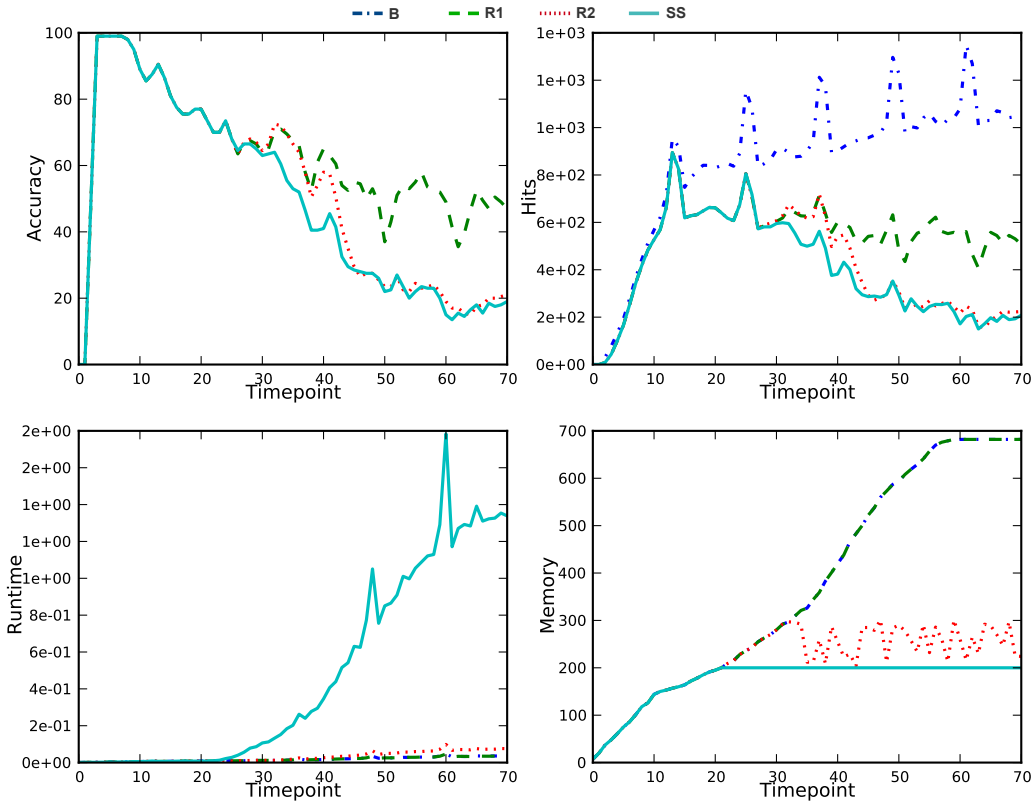

Figure 4.25: Plots for Financial: $N^{\mathcal{T}}$=100, M=200, β=2

In Figure 4.25, we increase the decay to $\beta = 2$. The only visible change is in the performance of R1. R2 registers slight improvement but after t_{45} it deteriorates as well. Even setting a decay value of $\beta = 5$ or increasing the size of $N^{\mathcal{T}}, M = 150, 300$ show no meaningful improvments in for R2 and SS (see Figures A.4 & A.5 in Appendix A.3).

On exploring the dataset further, we discovered that nature of the distribution for the number of references that an account receives in Financial per timepoint is Gaussian. In Figure 4.26 we plot the some of the statistics. On the left, we plot the cumulative references for an accounts objects receive from the timepoint it is first observed through to t_{72}. The plot on the right shows the average number of references an account objects registers at each timepoint after its arrival.

The main advantage that R2 and SS draw comes from their ability to separate highly referenced objects from those that receive only few references. In financial dataset, this phenomenon is non existent, thus they are unable post good performance.

In Appendix A.3, we report some more experiments on the Financial dataset. We modified the baseline slightly so that it still knows one timepoint into future, but rather than utilising that knowledge of future directly, it should utilise it via the counts it maintains for each object, i.e., it first updates its counters from the data at t_{i+1} and then decides on basis of those counts what to store in the cache. Under these conditions, even the baseline suffers performance degradation.

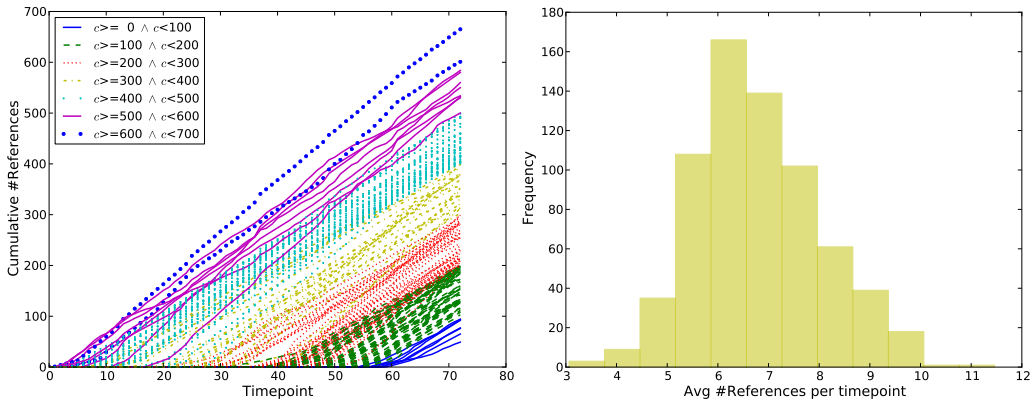

Figure 4.26: Statistics for # references for perennial accounts in Financial dataset. **(left)** Each line represents the accumulating references for an account over time, i.e., from the timepoint it first appeared through to the last timepoint (*plotted with a sample size of 170*). **(right)** Average number of references recorded for an account (we ignore the timepoints where an account did not appear).

4.4 Evaluating Incremental Propositionalisation

In the previous section we evaluated various components of Incremental-Propositionalisation, individually. In this section we present our experiments for the complete algorithm. For the experiments we are going to use three datasets: Ratings dataset, Gazelle dataset and Financial dataset, which we have already explained, earlier in Section 4.1.

IncrementalPropositionalisation first transforms the target stream of perennials into a propositionalised stream (cf. Algorithm 3, Section 3.2.3).

Acronym	REF	G1	G2	G3	G4	G5
`Products`	∞	5	25	75	150	200
`Contents`	∞	5	15	40	60	70
r	∞	7	7	7	7	14

Table 4.7: Experimental settings for Gazelle Dataset.

The clustering algorithm IncrementalKMeans (cf. Algorithm 5, Section 3.3) than operates over this transformed data to discover groups of similar objects. We describe our evaluation framework next.

4.4.1 Evaluation Framework

Our hypothesis is that the amount of information remembered from the multi-relational stream overtime, has an impact upon the quality of the clustering results. The remembered information is affected by the cache sizes $N^{\mathcal{T}}, N^{1...J}$, the size of the window ω over ephemeral streams and by the number of columns r_A reserved for each nominal attribute A. We have thus varied these values for the streams 'Accounts', 'Districts' of Financial dataset and the streams 'Products', 'Contents' for Gazelle dataset. The specification of cache sizes, i.e., $N^{\mathcal{T}}, N^{1...J}$, and reserved columns r_A is a "cache strategy". The strategies that we use for Gazelle dataset are shown in Table 4.7 and the ones for Financial dataset are shown in Table 4.8. For example, strategy *FIN2* uses a cache of 200 accounts and 40 districts objects and reserves 3 columns for storing nominal values of each attribute.

Acronym	REF	FIN1	FIN2	FIN3
`Accounts`	∞	100	200	300
`District`	∞	20	40	50
r	∞	3	3	3

Table 4.8: Experimental settings for Financial Dataset.

Evaluation Measures

At timepoint t_i, the incremental clustering algorithm adjusts the clusters of model ζ_{i-1} into model ζ_i. If the quality of ζ_i is lower than that of ζ_{i-1} below a threshold $\rho_{min} = 0.7$, then the data are re-clustered.

We use following quality measures: *Jaccard coefficient* (cf. Section 4.2 and/or [Tan et al., 2004] for details) to compare clusterings at different timepoints and for comparing different strategies (for the Gazelle dataset);

entropy to evaluate a clustering against explicit class labels (for the Financial dataset and the Ratings dataset).

The entropy measures the degree to which a cluster contains objects belonging to a single class. Let ζ be a clustering and ξ be the set of classes describing the data. For each $C_u \in \zeta$ and $L_v \in \xi$, the probability that an object in C belongs to L_v is $p_{uv} = \frac{|C_u \cap L_v|}{|C_u|}$. The entropy of C_u is $e(C_u, \xi) = \sum_{L_v \in \xi} p_{uv} log_2 p_{uv}$. The entropy of ζ is then

$$entropy(\zeta, \xi) = \frac{\sum_{C_u \in \zeta} |C_u| e(C_u)}{|\cup_{C_v \in \zeta} C_v|}$$

For entropy, *lower* values are better.

4.4.2 Experimental Results

Gazelle dataset

In the Gazelle dataset, we build user profiles on products inspected by each user (user "request") during each session, so sessions without product requests are dropped. There has been no information about re-visits of users, so a user corresponds to a session, and `Sessions` is the target. The contents of a session may be of several types, depicted in the table `Content`.

The Gazelle dataset has no ground truth, so we study the performance of our strategies towards the reference. We also mark the timepoints of re-clustering.

In the left side of Figure 4.27 we see the behaviour of each strategy for a sliding window of 14 units, where a unit is 75 propositionalised sessions. All strategies are initially comparable to the reference, but their performance deteriorates as records are forgotten. The large-cache strategies G4 and G5 have inferior performance than small-cache strategies. This indicates a tendency for new products and contents, in which small-cache strategies adapt fast.

A concept shift occurs soon after timepoint 40, whereupon the reference and some of the cache strategies experience re-clustering. The timepoint fits with a TV advertisement for Gazelle, which is known to have led to an increase in the site traffic. All strategies are slow in adapting to this concept shift, but the large-cache strategy G5 is the first to show an upward trend. This indicates that the range of products flowing into user profiles by that time is large.

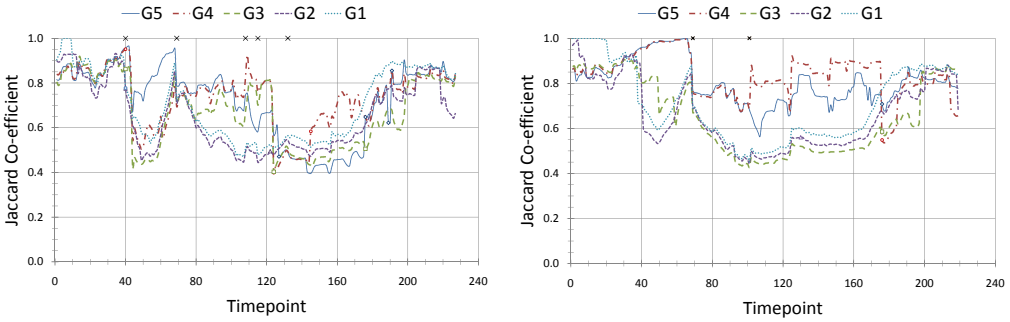

Figure 4.27: Gazelle Dataset: Jaccard for (a) $\omega = 14$ and (b) $\omega = 21$

The next concept shift occurs just before timepoint 70. After this shift, the large-cache strategies perform closer to the reference, while small-cache strategies do not perform well and re-cluster around timepoint 125. Two more shifts cause re-clustering of the reference a little earlier; they were both accompanied by a performance degradation for the large-cache strategies. It is remarkable that the strategy G4 has been close to the reference, although it did not re-cluster itself. This indicates that the clusters under this strategy were adapted adequately to the changes in the population. This also holds at the last shift around timepoint 135: the performance of the large-cache strategies deteriorates, but G4 experiences re-clustering only once. This indicates that a smaller cache leads to a better adaptation during that period. This claim is further supported by the small-cache strategies, whose performance improves without re-clustering. We interpret this as a tendency of the users to concentrate on few products.

On the right side of Figure 4.27, we see the behaviour of the strategies for a window size $\omega = 21$. We can observe the same phenomena, i.e. shifts and re-clustering, although the first shift is captured much later than for $\omega = 14$. Small-cache strategies show a steeper performance deterioration in early timepoints but improve in late timepoints and become more competitive. Until timepoint 175, G4 performs better than G5, which uses an even larger cache. At that timepoint, G4 shows very poor performance but improves a bit after re-clustering, although its quality is inferior to that of the small-cache strategies.

These results indicate that multi-table stream clustering allows for adaptation to shifts. Small-cache strategies adapt without need of quality monitoring and perform better when the data are very volatile. Large-cache strategies have higher performance when there is some stability in the data, but require quality monitoring to respond to concept shifts. As

for many phenomena on streams, the selection of a proper window size is of crucial importance. However, cache strategies allow for the incorporation of influence from data that do not belong to the target table, while softening the impact of the window size selection. As we have seen, even with a larger window size, caching allows for the adaptation to shifts, although these shifts are recognized with some delay.

Financial Dataset

In the Financial dataset, we use a cache for the stream `Accounts`; at each timepoint it accommodates a set of accounts and, after propositionalisation, of the transactions on them. Each account arrives with zero transactions and *evolves* into either "loan-trusted" or "loan-risk" class as more and more transactions are recorded for it. To avoid learning our models on data that arrive early but are not relevant (and would thus blur our results in an undisciplined way), we have trained a classifier (J4.8 Witten and Frank [2005]) on it, identified the subset of predictive attributes and then projected the remaining attributes away to reduce the noise. We varied the number of columns reserved for each nominal attribute: $r = 3$ and $r = 7$, but we found that the results were almost identical, except for some slight variations between timepoints t_{30} and t_{40}. Therefore, we report the results of $r = 3$ only.

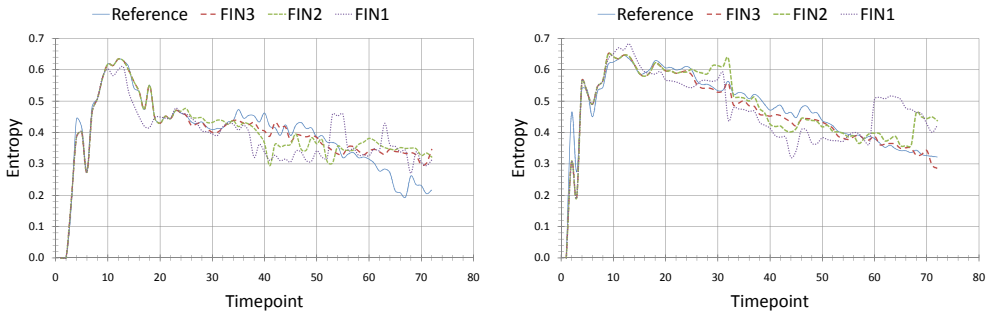

Figure 4.28: Financial Dataset: Entropy for $\omega = 30$ (left) $K = 9$ and (right) $K = 3$

With the strategies defined for Financial dataset in Section 4.4.1, we report experiments with two different settings. In the first set of experiments (the two plots shown in Figure 4.28), we consider the whole of transaction stream and use a window size of $\omega = 30$ for $K = 3, 9$. In the second set (shown in Figure 4.29 and Figure 4.30), instead of considering the whole of transaction streams, we consider the transactions from last 30 timepoints

only and repeat the selected transactions ≈ 4 times. We vary the window size $\omega = 24, 18$ and value of $\rho_{min} = 0.7, 0.85$ for these experiments.

In the left side of Figure 4.28, drawn for $K = 9$ clusters, we show the entropy values when comparing each cache strategy against the ground truth. At the beginning, all strategies have an entropy value of zero, because the first accounts belong all to the same class. As soon as accounts from the other class arrive, the entropy rises sharply.

It must be stressed here that accounts are perennial in nature, they mature and evolve over time. When accounts are introduced, the clusterer is only aware of their initial intrinsic properties e.g. the information about the owner(s) and types of card they hold, the district they were created in and etc. There is little or no transaction information associated with them, so they are initially grouped into clusters on the basis of these static properties. On the other hand, the class label reflects their *final* state, after many transactions have accumulated on them. This is the reason that all strategies perform poorly at the beginning.

Around timepoint t_{10}, the entropy of FIN1 (the strategy with the smallest cache) starts dropping. By this time, more than 100 accounts have arrived, and the $CacheUpdate$ algorithm (Section 3.2.1) prefers those that have performed more transactions. For FIN1 with its small cache, this means that accounts with few transactions are not in the cache. All other strategies, including the reference strategy, have larger caches and store these accounts, which cannot be easily classified as loan-risk vs loan-trusted, and thus result in bad performance. At later timepoints, i.e. after t_{23} for FIN2 and after t_{30} for FIN3, these strategies also drift away from the reference strategy: as they reach their cache size limit, they keep only mature accounts inside the cache, so their performance increases.

From timepoint t_{32} until t_{55}, the reference strategy with its infinite cache shows the worst performance. The lesson learned is that in stream mining it is not always desirable to remember all the data. For the Financial dataset, oblivion is best: FIN1 that has the smallest cache size outperforms all other strategies.

After timepoint t_{55}, only very few new accounts arrive, the last one at t_{60}. For the next 12 timepoints, all accounts keep evolving as new transactions arrive for them. The performance of the reference strategy also improves towards the end since there are no noisy accounts to perturb it. It outperforms all strategies as there is no information loss due to memory limitations.

In the right side of Figure 4.28, we show the cache strategies for $K = 3$ clusters. We chose a small K to test whether the objects of the majority class A/C are better accommodated in few large clusters. $K = 3$ turned

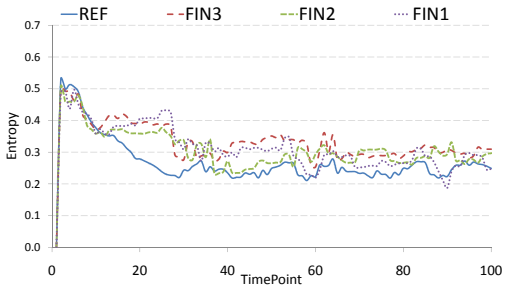

Figure 4.29: Financial Dataset: Entropy for $K = 9$, $\rho_{min} = 0.7$ ($\omega = 24$

out to be not a good choice for the data with a very skewed distribution of classes and very little separation between the data points. Before t_{32} when information about accounts is very little, all the strategies perform very poorly. Its performance starts getting better as information the accounts mature. However, it should be noted that all the strategies were able to separate a large number *loan-risk* accounts into one cluster but the cluster also contained almost the same amount (or even more at some timepoints) of *loan-trusted* accounts while the other two clusters were mostly clean. This explains the relatively low entropy for the strategies with $K = 3$. Whereas several small clusters with $K = 9$ achieved separation by discovering clusters that contained only the *loan-risk* accounts.

In Figure 4.29, drawn for $K = 9$, $\omega = 24$ and $\rho_{min} = 0.70$, we show the entropy for each cache strategy. As we have mentioned earlier that for this dataset we repeat the transaction from last 30 months/timepoints. Because of the large window size, i.e., $\omega = 24$, almost all of the available information gets enveloped. As the transactions accumulate, FIN1 and FIN2 are the first to show improvement. However, after timepoint t_{12} they are over taken by REF. From t_1 to t_{12} about 400 accounts are active. The main advantage that the smaller strategies draw comes from their ability to prefer objects that have grown substantially and are likely to carry more information. Due to the richness of information in the subset of transactions (i.e. last 30 timepoints), almost all of these accounts are growing simultaneously by the timepoint t_{15}. Therefore the reference strategy that can cache all objects shows best performance.

By timepoint t_{30} all accounts mature. As the transactions repeat, the reference strategy shows a strong periodic behaviour with lowest entropy. FIN2 is the second most competitive strategy during timepoints $t_{30} \ldots t_{60}$. FIN1 also has a periodic behaviour and shows improved performance as

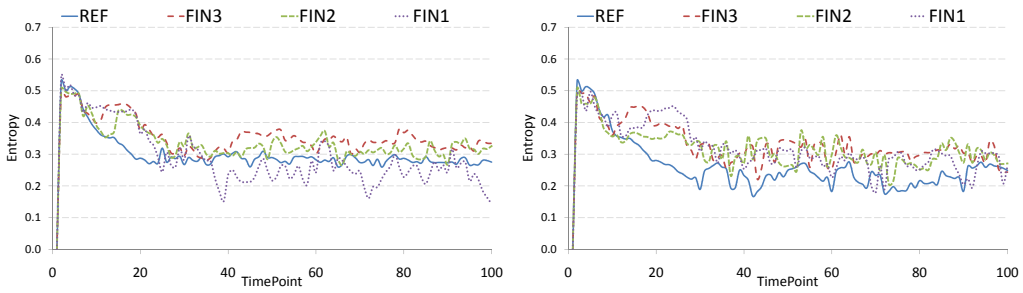

Figure 4.30: Financial Dataset: Entropy for $K = 9$, (left) $\omega = 18$, $\rho_{min} = 0.7$ and (right) $\omega = 24$, $\rho_{min} = 0.85$

the transaction data are repeated.

In the left side of Figure 4.30, we show the entropy of cache strategies drawn for $K = 9$, $\omega = 18$ and $\rho_{min} = 0.70$. Till timepoint t_{18} performance is similar to that for $\omega = 24$. The reference strategy shows best. As the window size is reached at t_{18}, the reference stabilises and does not show any significant improvement after that. By timepoint t_{30} all strategies have somewhat comparable performance. As the transactions are repeated, the performance of FIN1 starts improving. As we have pointed out earlier, the cache strategies draw advantage by focussing on objects that are more mature than others. Because of the smaller window size, accounts with less transactions and contain less information are dropped. FIN1 improves its performance by focusing on the accounts that have done more transactions and shows the best performance as the stream gets repeated over and over.

In the right side of Figure 4.30, we show the entropy of cache strategies drawn for $K = 9$, $\omega = 24$ and $\rho_{min} = 0.85$. The graph is comparable to Figure 4.30(right), however is a bit more fragile because it re-clusters more due to higher value of $\rho_{min} = 0.85$. The strategies, specially the reference, show slightly better performance at various timepoint due to stricter threshold.

Ratings Dataset

In the experiments with the Ratings dataset we use a cache for stream 'User' while use a window over the stream `Rating`. We have varied both the parameters and studied its effect on the learned clustering model.

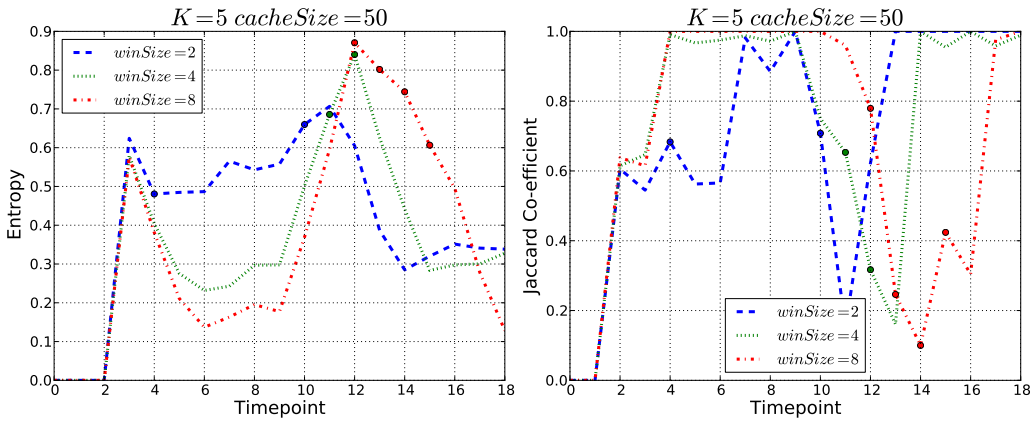

Figure 4.31: Result on Varying the sliding window on Ratings Dataset $K = 5$ and $N^{\mathcal{T}} = 50$ (left) Entropy and (right) Jaccard Co-efficient. The points on the plot lines represent the moments where the clustering algorithm performed re-clustering.

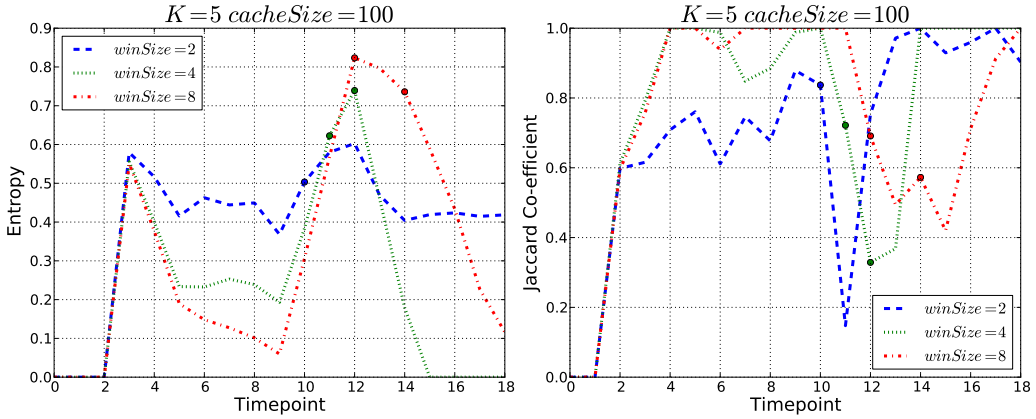

Figure 4.32: Result on Varying the sliding window on Ratings Dataset $K = 5$ and $N^{\mathcal{T}} = 100$ (left) Entropy and (right) Jaccard Co-efficient. The points on the plot lines represent the moments where the clustering algorithm performed re-clustering.

In Figure 4.31 we set the cache size over the stream `User` for all the strategies to be 50 and vary the amount of information they can retain from the ratings stream. The effect of varying the parameter ω are predictable. All strategies undergo performance degradation at timepoint t_{12}. The ones with the larger window sizes react late towards the drift and their recovery is slow as well. Their late reaction is identified by the 'dots' on the plot lines. These dots represent the timepoints when re-clustering was done

by the algorithm. However, once they are stable (i.e., timepoints $t_4 - t_{t10}$ and t_{14} onwards), their performance are better than the ones that relatively smaller to them. Both in terms of entropy and stability (depicted using the Jaccard values on the left of the Figure 4.31).

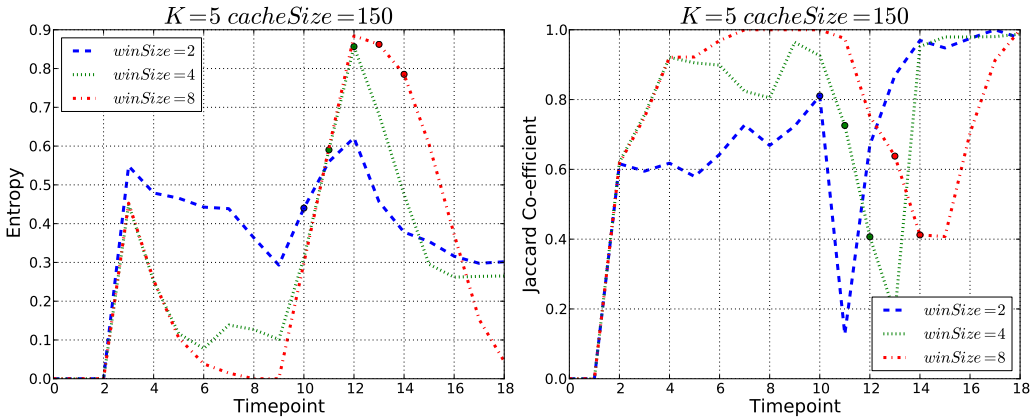

Figure 4.33: Result on Varying the sliding window on Ratings Dataset $K = 5$ and $N^{\mathcal{T}} = 150$ (left) Entropy and (right) Jaccard Co-efficient. The points on the plot lines represent the moments where the clustering algorithm performed re-clustering.

The same affect can be seen in the Figures 4.32-4.34 when we vary the cache size. However, when we juxtapose the figures with varying cache size over each other we observe that performance of the strategies improves and not deteriorates with the increasing cache size. This effect of the caches is exactly opposite to the one that we have see earlier in the experiments with Gazelle and Financial datasets, where performance actually improves.

Upon investigating the dataset, we discovered that there is a slight difference in the nature of these datasets. In Financial dataset, there are perennial objects with very few transactions, and also those with large number of transactions. This holds for the Ratings dataset as well. However, in Financial dataset new objects arrive gradually, all the time, while for Ratings dataset all the objects arrive at once. Hence, in the Ratings dataset, when we increase the cache size, the strategies accommodates objects that have done a large number of transactions but do not include completely immature objects For this reason the entropy for the learned model decreases. Where as, in Financial dataset strategies have to wait until the cache gets filled by mature objects and only after that moment, their performance starts to increase.

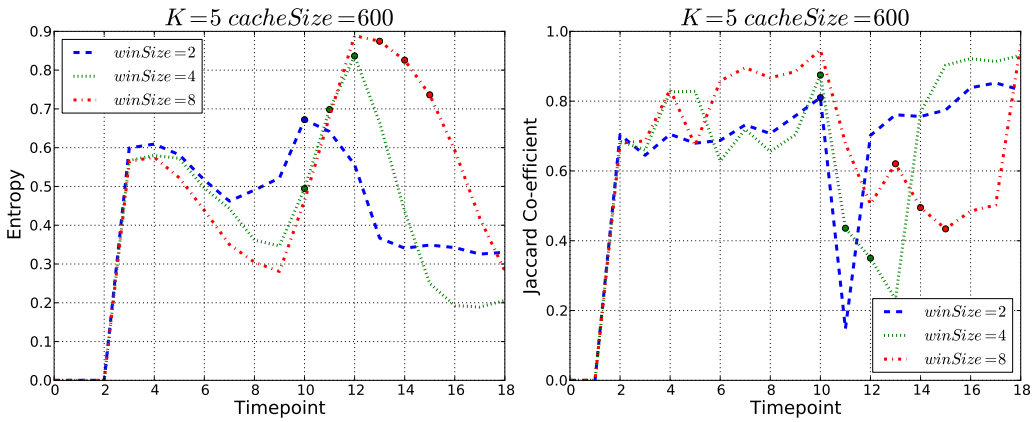

Figure 4.34: Result on Varying the sliding window on Ratings Dataset $K = 5$ and $N^{\mathcal{T}} = 600$ (left) Entropy and (right) Jaccard Co-efficient. The points on the plot lines represent the moments where the clustering algorithm performed re-clustering.

However, when we increase the cache to encompass all the objects seen thus far in the Ratings dataset (see Figure 4.34), i.e., ones with low number of transactions and the ones with large number of transactions, we see a drop in the performance of the all the strategies in relation to the ones with smaller cache sizes. Not only their entropy increases, but their Jaccard values are lower (which means that they change much with regards to the model from the previous timepoint) and they re-cluster more as well.

4.5 Conclusion

In this chapter we have presented a lengthy evaluation of Incremental-Propositionalisation, which include the evaluation of the algorithm for nominal space adjustment (*NominalSpaceAdjustment*, cf. Section 3.2.4), the different caching mechanisms (*CacheUpdate* and *SmallCache*, cf. Section 3.2.1) and the incremental propositionalisation of multiple interrelated streams of perennial objects.

Different cache strategies proposed in this thesis, cater well if the underlying distributions for the number of perennial objects is Zipf distribution. Our caching strategies generally performs well, especially R2 which not only shows competitive performance towards in comparison with R1 (with infinite resources) under stable conditions[5] but its recovery time is

[5]When there is no concept drift.

also competitive with respect to SS. The main reason for its competitive performance is its tighter and more realistic approximation on the number of missed objects.

The evaluation performed regarding caching strategies, also laid bare some important characteristics about perennial objects. These characteristics also depend on the context in which perennial objects are found. For example, in the financial dataset from banking domain, perennial objects do not arrive all at once, but gradually. Once they are seen for first time, all of them reappear regularly with the number of references for each object at each timepoint following a normal distribution (as depicted in Figure 4.26). In contrast, the appearance and re-appearance of perennial objects in the MovieLens dataset from the recommendation is irregular or consider the online auction sites, such as eBay, where an auction receives the most bids when it is about to be closed. This calls for more elaborate caching strategies, that can also take the domain of the considered problem into account.

In the last section of this chapter we evaluated IncrementalPropositionalisationas a whole (with all its sub procedures). Another important aspect that we discovered about the perennial is that the maturity of a perennial objects has a direct effect on the quality of the learned clustering model. From stream mining we know that a model that remembers everything is not always the best one. In IncrementalPropositionalisationdue the size of the cache, perennial objects that are less mature get filtered out implicitly. The algorithm prefers those objects that have more references (and more information as well), which contribute towards the quality of the learned model.

CHAPTER 5
Classifying Perennial Objects

Traditional classification algorithms designed for the analysis of stream data assume that objects (along with their class label) enter the horizon of observation, are processed to the effect of learning and adapting the model (e.g., a decision tree), and are then forgotten when they get old. This has been inspired by the stream mining paradigm that has been motivated by the obvious fact that it is practically impossible and pragmatically unnecessary to maintain each data point (e.g., registered by a sensor or recorded in a server log) for longer time than needed to update the underlying model. However, many applications require learning over *perennial* objects like patients or customers. As we discussed earlier (cf. Chapter 1) perennial objects are complex objects that (a) may not be forgotten, (b) may evolve over time, (c) may change their class labels and (d) constitute themselves a stream. Stream classification over perennial objects is a new problem for which conventional techniques for model learning and adaptation must be reconsidered. In this chapter we build upon our solution on building perennial objects through propositionalisation (cf. Chapter 3) and propose a solution for the task of decision tree induction over a stream of perennial objects.

Among the four challenges mentioned above, particularly important for stream classification is the issue of label change (challenge c), because it corresponds to a new kind of concept drift. Contrast that in conventional classification in conventional classification, an object has a given label, e.g., a customer is trustworthy or not, an Alzheimer patient either exhibits pacing or does not; the label of a perennial object may change though, i.e., the customer may stop being trustworthy, while the patient may start exhibiting pacing (by the nature of the Alzheimer illness). Such drifts must be captured by the model, while retaining the circumstances (data) before and after the drift for use upon further similar objects. This calls for a new kind of model adaptation. Finally, as mentioned before, even if some perennial objects must be moved out of storage, their contribution in the model should be retained, at least for some (application-specific) time period.

In this chapter, we propose a new approach for classification over a stream of perennial objects, taking account of the above challenges. We

Table 5.1: List of used terms and symbols.

Notation	Description
ψ	attributes from the propositionalised stream $\mathcal{P}$.
$\mathcal{R}$	root node of the tree induced by TrIP
Ω	a node of the induction tree
$Q_{t_i}^{\Omega}$	accuracy of the sub-tree with root Ω at t_i
$var_{t_i}^{\Omega}$	variance in the accuracy of the children Ω at t_i
$age_{t_i}^{\Omega}$	age of node Ω at t_i
$s_{t_i}^{\Omega}$	support of node Ω at t_i: number of objects in Ω at t_i
coefficients	$supportCoeff(\Omega, t_i)$: support coefficient of a node Ω at t_i
	$qualityCoeff(\Omega, t_i)$: quality coefficient of a node Ω at t_i
δ	confidence threshold for the Hoeffding Bound
τ	used for resolving ties in split decisions
n	min #objects that must be in a leaf node before it is checked whether the node can be split; used to compute the Hoeffding Bound
f	min #objects to check for concept drift
l^{min}	min #objects that must be in a leaf node after a split
s^{min}	support $[0, 1]$ below which a leaf starts to age

build upon our earlier work on multi-relational stream mining to combine the stream of perennial objects and the streams of simpler objects (transactions, activities and similar) associated to them into a "multi-table stream" upon which a classifier can be applied. We then extend the incremental tree induction algorithm CVFDT proposed by Hulten et al. [Hulten et al., 2001] for a conventional data stream into an adaptive learner for a stream of perennial objects. The extensions are twofold: the new algorithm can deal with the fact that the objects are perennial, hence their contribution to the model may not be forgotten, and with the concept drift incurring as some perennial objects change their label.

5.1 Tree Induction over a Stream of Perennial Objects

Our ***Tr**ee **I**nduction algorithm for **P**erennial objects* (TrIP) has several components. First, the IncrementalPropositionalisation algorithm (cf. Algorithm 3 in Chapter 3) that combines the stream of perennial objects with further streams (of perennial or ephemeral objects) transforming them into

Algorithm 6: TrIP

Input : $\mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon$, $\omega, n, f, l^{min}, s^{min}, \delta, \tau$
Output: Ω

Create root node $\mathcal{R}_0$.
Initialise *sufficient statistics*.
for $t_i = t_1$ **to** *STREAM_END* **do**
    $\mathcal{P}_i \leftarrow$ `IncrementalPropositionalisation(` $\mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon$)
    $\mathcal{R}_i \leftarrow$ `TrIPCore(`t_i, $\mathcal{R}_{i-1}$, $\mathcal{P}_i$, $\omega, n, f, l^{min}, s^{min}, \delta, \tau$`)`

Algorithm 7: TrIPCore

Input : t_i, $\mathcal{R}$, $\mathcal{P}_i$, $\omega, n, f, l^{min}, s^{min}, \delta, \tau$
Output: Ω

foreach $x \in \mathcal{P}_i$ **do**
    if $x.id \in (\mathcal{P}_{i-\omega}, \mathcal{P}_{i-1}]$ **then**
        `RecursivelyForget(`$\mathcal{R}, x$`)`
    `PresentExample(`$\mathcal{R}$, x`)`
    foreach n *objects* **do**
        $\mathcal{R} \leftarrow$ `GrowNAdaptSubtree(`$\mathcal{R}$, $n, f, l^{min}, s^{min}, \delta, \tau$`)`
foreach $x \in \mathcal{P}_j \mid j = i - \omega$ **do**
    `RecursivelyForget(`$\mathcal{R}, x$`)`
Return $\mathcal{R}$

a single stream for mining. This component is coupled with a sliding window mechanism that replaces outdated perennial objects with their up to date version and forgets those that are not needed for the current version of the model. The tree induction and adaptation with help of alternate trees is based on CVFDT [Hulten et al., 2001], but this component has been extended to cover the particularities of perennial objects, including a new form of concept drift. We present each component in turn in the following subsections. The basic notations and symbols are same as that from the last chapter (c.f. Table 3.1). Additional notation and parameters are presented in Table 5.1.

For a target stream $\mathcal{T}$ that receives ephemeral objects from streams $S^1, S^2 \ldots S^J$, TrIP grows a decision tree incrementally. For simplicity and the ease of invocation, we present TrIP in two parts: The first part, de-

picted in Algorithm 6, provides a high level overview of the algorithm. It first initialises the root node $\mathcal{R}$ (Line 1) and the *sufficient statistics* (cf. Chapter 2, also discussed in the next subsection). Then, as new data arrive in the multiple interrelated streams at each timepoint, it summarizes them into a single propositionalised stream $\mathcal{P}_i$ (Line 4) and the invokes the core functionality of TrIP (depicted in Algorithm 7: TrIPCore) to update the tree model R_{i-1} from the previous timepoint by incorporating new data from $\mathcal{P}_i$ (Line 5).

The core functionality of TrIP is wrapped inside Algorithm 7: TrIPCore. At each timepoint, it recursively presents each perennial object x from $\mathcal{P}_i$ to all the nodes in the tree, starting from $\mathcal{R}$ (Line 4). When at least n objects have been presented, TrIPCore expands the tree by splitting the leaf nodes (Line 6). Periodically, TrIPCore updates and forgets outdated perennial objects (Line 3 & 8).

In the remainder of this thesis, we would refer to the Algorithms 6 & 7 as TrIP.

5.1.1 Maintaining Sufficient Statistics

We have briefly discussed the notion of sufficient statistics in the context of tree induction algorithms in Chapter 2. We describe it again in detail. Stream based tree induction algorithms [Domingos and Hulten, 2000; Hulten et al., 2001; Gama et al., 2003] stores the information about the distribution of class labels with respect to each attribute-value pairs. This information is maintained for each node in the tree. For example,

$$(\text{att=}\textbf{forecast}, \text{val=}\textit{rain}) \rightarrow [\texttt{+ve}\text{=}90, \texttt{-ve}\text{=}10]$$

It reads, that of the 100 examples that satisfy (**forecast**=*rain*), 90 belong to the positive class while 10 belong to the negative class.

Maintenance of sufficient statistics for nominal attributes is straightforward. For maintaining sufficient statistics for numerical attributes, we use an approach which is similar to the binning method employed in VFML [Holmes et al., 2005]. For every new value that we observe, we create a new bin. New bins are added until a pre-defined number of bins is reached, after which the observed range is sub-divided into z bins of equal length. For example,

$$(\text{att=}\textbf{temperature}, \text{val=}\textit{bin}_{[22.5,25.0)}) \rightarrow [\texttt{+ve}\text{=}40, \texttt{-ve}\text{=}10]$$

Algorithm 8: RecursivelyForget

Input : Ω, x

```
1 if x.tp ≥ Ω.tp ∧ x.ID ≥ Ω.firstID then
2     Decrease sufficient statistics according to x.
3     Ω_c ← FindChildConsistentWithExample(x)
4     RecursivelyForget(Ω_c, x)
5 foreach Ω_alt ∈ GetAltTrees(Ω) do
6     RecursivelyForget(Ω_alt, x)
```

It reads, that of the 100 examples that where the value for **temperature** is between *22.5 to 25.0* or fall inside $bin_{[22.5,25.0)}$, 40 belong to the positive class while 10 belong to the negative class. The split decisions over numerical attributes are binary, e.g., att=**temperature**>22.5. For checking/employing binary splits, the boundaries of the numerical bins serve as potential split points.

The bin-based approach to store the numerical values is an expensive approach as also observed in [Holmes et al., 2005]. There exist other more sophisticated approaches for maintaining sufficient statistics over numerical attributes [Gama et al., 2003, 2004; Jin and Agrawal, 2003].

5.1.2 Dealing with Outdated Perennial Objects

The stream of perennial objects cannot be propositionalised once and forever: as new perennial or ephemeral objects arrive, the target schema needs to be updated – adding, modifying and deleting columns, as explained in Chapter 3, as well as the perennial objects seen thus far. Hence, TrIP employs a sliding window of length ω timepoints: at timepoint t_i the newest block of objects from the caches and windows of $\mathcal{T}$, $S^1 \dots S^J$ is read and propositionalised into $\mathcal{P}_i$. Then, outdated perennial objects must be forgotten. We distinguish two cases, as described below.

Perennial objects are kept up-to-date by the arriving ephemeral objects that reference them. If for a perennial object x, no ephemeral object has been observed for last ω timepoints, then x carries no new information and is unlikely to influence the current model. Hence, we remove such objects from the model. Obviously, the objects themselves are not eliminated from secondary storage.

The second case of outdated perennial objects concerns object replacement. In particular, let $x \in \mathcal{T}$ be a perennial object from the target stream

Algorithm 9: PresentExample

Input : Ω, x
Output: **None** (Updates the node Ω only)

```
1 Update sufficient statistics for Ω according to x.
2 if Ω is not leaf then
3 |   Ω_c ← FindChildConsistentWithExample(x)
4 |   PresentExample(Ω_c, x)
5 foreach Ω_alt ∈ GetAltTrees(Ω) do
6 |   PresentExample(Ω_alt, x)
```

and x_i its content at timepoint t_i, composed of the data in $\mathcal{T}$ and the propositionalised data from $matches(x)$ from the other streams. If a new instance $x_{i'}$ arrives at $t_{i'} > t_i$, then $x_{i'}$ replaces the old instance x_i.

The process of forgetting the impact of outdated perennial objects is undertaken by Algorithm 8: RecursivelyForget. To do so, it maintains for each node Ω the timepoint tp at which it was created, as well as the identifier $firstID$ of the first object placed in this node. Then, at a timepoint $t > tp$, the function ignores all objects that have appeared before tp for the last time and those that have identifiers less than $firstID$ and have not been seen within the last ω timepoints. The sufficient statistics on the contents of node Ω and its sub-trees are updated accordingly.

5.1.3 Growing a Decision Tree on a Propositionalised Stream

TrIP starts building the decision tree from the root $\mathcal{R}$, which is initially the only node in the tree and is also a leaf. For each node, the sufficient statistics are maintained separately. When an object x arrives, it is recursively presented to the tree, starting from the root node $\mathcal{R}$. The process is undertaken by Algorithm 9: PresentExample. When x is presented to a node Ω, first, the sufficient statistics for Ω gets updated (Line 1) and then x is passed down to the relevant child node Ω_c (Lines 3 & 4).

Once n objects have been presented to a *leaf* node Ω, TrIP considers Ω as a candidate for split and expands the tree. This process is undertaken by Algorithm 10: GrowNAdaptSubtree and is invoked recursively from TrIP.

Algorithm 10: GrowNAdaptSubtree first checks if the node Ω is leaf or not (Line 1). If Ω is a leaf node, it attempts to split the node (Line 2).

Algorithm 10: GrowNAdaptSubtree

Input : $n, f, l^{min}, s^{min}, \delta, \tau$
Output: Ω

```
1  if Ω is leaf then
2      child_list ← Split Ω using the best attribute ψ_a.
3      if child_list is not empty then
4          Add newly created child nodes from list_child to Ω .
5  else
6      AgeTree(Ω)
7      Ω ← ReplaceWithAltTree(Ω)
8      for every f objects seen do
9          ValidateSplit(Ω)
10     forall the Ω_c ∈ GetChildren(Ω) do
11         Ω_c ← GrowNAdaptSubtree(Ω_c, n, f, l^min, s^min, δ, τ)
12     forall the Ω_alt ∈ GetAltTrees(Ω) do
13         Ω_alt ← GrowNAdaptSubtree(Ω_alt, n, f, l^min, s^min, δ, τ)
14 Return Ω
```

Before a node can be split, the information gain $G(\psi)$ is computed for every attribute $\psi \in \mathcal{P}_i$. Based on the value of $G()$, it selects the best and second best split attributes ψ_a and ψ_b, respectively. It the calculates $\Delta G = G(\psi_a) - G(\psi_b)$ and computes the Hoeffding bound ϵ on the true mean of ΔG with confidence δ (cf. Equation 2.1). As explained in Chapter 2, if $\Delta G > \epsilon$, then Ω is split on ψ_a, otherwise the split decision is postponed and further objects are read and processed.

As a node sees more and more objects, n increases and ϵ decreases (cf. Equation 2.1). As long as ΔG remains less than ϵ, no split can be performed. To avoid spending time on attributes with very close gain, TrIPuses the tie breaking mechanism from CVFDT [Hulten et al., 2001]: once the value of ϵ drops below a threshold τ, the algorithm is forced to make a split on the current best attribute.

Next to the leaf node splitting (Lines 2-4), Algorithm 10: GrowNAdaptSubtree is responsible for the adaptation of a node to concept drift (Lines 6-9). We elaborate on these steps hereafter.

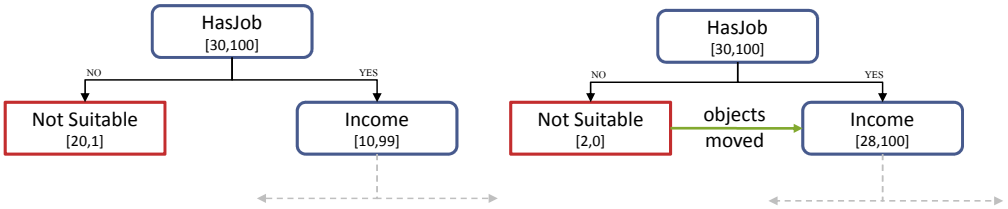

Figure 5.1: Example: Tree losing support as objects change their label

5.1.4 Tree Ageing and Generation of Alternate Trees

Similarly to CVFDT, TrIP maintains alternate trees as means for fast adaptation to concept drift. However, concept drift upon a stream of perennial objects occurs in two ways: (1) some perennial objects change their label as time passes; (2) the number of objects associated with each label changes as time passes. The second type is the conventional concept drift. The first type occurs only upon perennial objects because of their very nature, and is independent of the second type. For both types of drift, we must take account of the fact that changes in the number of objects having a given label may be of temporary nature only. We explain this by means of an example.

Assume a partner matching site, in which a set of persons are registered. The objective is to learn the concept "get a spouse". Assume that the tree Ω shown in Figure 5.1 to be the one learned till timepoint t_i. The current best split attribute is "HasJob" with both left and right sub-tree accommodating a sufficient number of objects. At a later timepoint t_j, some objects currently in the left sub-tree get a job and then a partner. This means that the number of objects in the left sub-tree decreases and the gain for the attribute "HasJob" may drop as well. This renders the current split suboptimal and forces the algorithm to choose a new split attribute and start growing an alternate sub-tree under it.

However, the original concept may well re-appear and the number of objects in the left subtree may start growing again. When new objects that are consistent with the replaced sub-tree start arriving again, it would take some time before the algorithm re-learns the discarded concept. Hulten et al. also identified re-learning of a discarded periodical concept as a direction for future work [Hulten et al., 2001].

TrIP checks for concept drift every f objects seen during maintenance (Algorithm 10, Line 9). TrIP inspects each node Ω if the current split attribute $\psi_{a'}$ is still the best split decision or not. In case the current split

attribute is not the best split attribute any more, i.e., $\psi_{a'} \neq \psi_a$, TrIP checks if an improved split can be installed at Ω using ψ_a. If a new split is possible, it initializes an *alternate* sub-tree Ω_{alt} (rooted at $parent(\Omega)$) with ψ_a as the new split attribute, separately. To avoid excessive alternate tree generation, tie breaking condition is made tighter, i.e., $\Delta G \leq \epsilon < \frac{\tau}{2}$. Hulten et al. points out that a simple approach would be to replace Ω as soon as an alternate tree Ω_{alt} with a split anchored at ψ_a is generated [Hulten et al., 2001]. This would ensure that the induced tree, rooted at $\mathcal{R}$, is always as accurate as possible. However, it would force Ω_{alt} (with 2 leaves only) to do the job that was earlier done by a whole sub-tree. Similarly to the CVFDT, TrIP waits for the accuracy of alternate tree Ω_{alt} to be greater than Ω and then replaces Ω with Ω_{alt} (Algorithm 10, Line 7). However, due to the dynamic nature of the perennial objects, TrIP employs a different method for the adaptation.

To deal with the pitfalls of accepting a short-lived concept, TrIP checks (a) whether the accuracy of the tree deteriorates, (b) whether each node of the tree still represents, i.e., "is supported by", an adequate number of perennial objects, and (c) for how long the tree has (or has not) received support. This corresponds to monitoring for each subtree (a) the quality of the concept in it, (b) the support, i.e., the number of objects accommodated in it and (c) its age as the time elapsed since quality deterioration has started in it. By this, a tree of good quality is not aging, even if its support drops.

More formally, TrIP calculates for each node Ω two coefficients, *supportCoeff* and *qualityCoeff* that take a tree node Ω and a timepoint t_i as parameters. The support coefficient calculates the penalty for the children of Ω with support less that the threshold s^{min} (cf. Table 5.1). The formula for support penalty for child c is shown in Equation 5.1, where, $s^{\Omega_c,max}$ is maximum observed support for child node Ω_c prior t_i:

$$s^{\Omega_c,max} = \max(s^{\Omega_c}_{t_1}, \ldots, s^{\Omega_c}_{t_{i-1}})$$

$s^{\Omega_c}_{t_i}$ is the current support at t_i.

$$supportCoeff(\Omega, t_i) = \sum_{\Omega_c \in child(\Omega)} \begin{cases} \frac{s^{\Omega_c,max} - s^{\Omega_c}_{t_i}}{s^{\Omega_c,max}} & if\ s^{\Omega_c}_{t_i} < s^{min} \times s^{\Omega_c,max} \\ 0 & if\ s^{\Omega_c}_{t_i} \geq s^{min} \times s^{\Omega_c,max} \end{cases} \tag{5.1}$$

The quality coefficient calculates the change in the classification accuracy of a node at timepoint t_i from that in t_{i-1}.The formula for calculating the quality coefficient is shown in Equation 5.2

$$qualityCoeff(\Omega, t_i) = \begin{cases} (1 - var^{\Omega}_{t_i})\Delta Q^{\Omega} & if\ \Delta Q^{\Omega} > (1 - \delta) \\ Q^{\Omega}_{t_i} - 0.5 & otherwise \end{cases} \quad (5.2)$$

where $\Delta Q^{\Omega} = Q^{\Omega}_{t_i} - Q^{\Omega}_{t_{i-1}}$ and $Q^{\Omega}_{t_i}$ and $Q^{\Omega}_{t_{i-1}}$ are the classification accuracy of tree Ω at t_i and t_{i-1} respectively and $var^{\Omega}_{t_i}$ is the variance in the classification accuracy of the children of Ω (cf. Table 5.1). The variance among the children of Ω is used to capture the case where one child/sub-tree c has much lower quality than the other: then, the quality coefficient of Ω itself is not affected, thus preventing a premature ageing of the whole sub-tree under Ω. The quality coefficient of c itself will be low, hence c will age faster.

Then, the age of an internal node at timepoint t_i is computed upon the age of the node at the previous timepoint, and the node's support and quality coefficients:

$$age^{\Omega}_{t_i} = age^{\Omega}_{t_{i-1}} + supportCoeff(\Omega, t_i) - qualityCoeff(\Omega, t_i) \quad (5.3)$$

The maintenance procedure is recursively invoked at each tree node Ω. At node Ω, TrIP checks if the attribute $\psi_{a'}$ originally used to split the node is still the best split decision (cf. Algorithm 10, Line 9). If not, i.e., another attribute ψ_a provides a split of higher gain, then TrIP starts growing an alternate (sub-)tree Ω_{alt} and split using ψ_a.

Similarly to CVFDT, TrIP replaces Ω with Ω_{alt} only if (and after) the accuracy of Ω has dropped below that of Ω_{alt} (cf. Algorithm 10, Line 7). Dissimilarly to CVFDT, TrIP may retain Ω even if its accuracy dropped lower than that of Ω_{alt}. Ω is discarded only if its age exceeds a certain threshold, which is expected to capture background knowledge about periodicity in the application. At each timepoint t_i the age age_{Ω,t_i} of node Ω is computed as a function of its support and quality and is updated incrementally (Line 6). If the threshold is not exceeded, Ω is available and its quality is checked at each timepoint: if its quality improves, it *rejuvenates*, i.e., its age decreases, as can be seen in Equation 5.3.

5.2 Experiments

In this section we evaluate TrIPwith synthetic and real datasets. Our objective is to study the performance of our methods over a stream of perennial objects. To this purpose, we designed a variety of experiments that deal

with the effect of window size and tree ageing on quality. First we describe the datasets and the evaluation measures and then present our findings on the dataset.

5.2.1 Datasets

Synthetic: Marriage Dataset

It is a small dataset with 3 features and 200 data instances that was created to test specific aspects of TrIP. The objects in the dataset are dynamic. They represent people, labelled on whether they are likely to find a match or not on the basis of their age, income and marital status. They grow old, their income changes from year to year and their marital status changes as well. The underlying data generating process remains unchanged until t_{30}, where we have imputed a concept drift for the last 5 timepoints. However, the objects can change their properties and their class labels any time.

Synthetic: Ratings Dataset

The dataset that we have used here is identical to the one that we have used in the increment propositionalisation experiments in Chapter 3. The Ratings dataset has around 600 users. The number of transactions made by them vary with a zipf distribution. There were five user profiles and 20 item profiles. The data is distributed across 20 timepoints, i.e., t_1 to t_{20}. The user profiles undergo drift at timepoint t_{11}.[1]

Financial Dataset

We have already described the financial dataset in Section 4.1 of Chapter 4. We briefly describe it again. Financial dataset is a multi-relational dataset. The tables represent the activities (transaction and loans request etc.) of bank customers.

This dataset puts forwards a difficult learning problem. The class distributions are not only very skewed to begin with; they also reflect the state of accounts only when they have matured, i.e., class labels become applicable at a much later timepoint than when the objects were introduced.

[1]Full specification of the dataset, i.e., the parametric settings for the generator are given in Appendix A.2.

Acronym	ω	age_{MAX}	δ	τ	n	f
QR	5,7	0	0.9	0.1	4	8
DR		4				
NR		∞				

Table 5.2: Strategies on Marriage Dataset

5.2.2 Evaluation Framework

In this section we describe our framework for evaluation. First. we describe the experimental settings and the strategies that we use and describe the evaluation measure to for those strategies.

Experimental Settings

Trip Variants for the Synthetic Datasets For the marriage dataset, as there are no multiple streams, we use $\omega = 5, 7$ timepoints and vary $age_{MAX} = 0, 4, \infty$. The parameter ω controls how past data is to be incorporated within the model, while age_{MAX} is that threshold for the replacing a tree with its alternate tree. When $age_{MAX} = 0$, a sub-tree rooted at Ω that no longer has best split decision at its root can be immediately replaced once the accuracy of alternate tree Ω_{alt} becomes better. When $age_{MAX} = 4$, Ω can be replaced by Ω_{alt} only if $a_{T,i} >= age_{MAX}$. When $age_{MAX} = \infty$, Ω never gets replaced by Ω_{alt} as it never really reaches age_{MAX}. We name the strategies as Quick (QR), Deferred (DR) and No Replacement (NR), respectively. The other tree parameters are: $n = 4$, $f = 8$, $\delta = 0.9$, $\tau = 0.1$, $s^{min} = 0.1$. The experimental settings and the strategies are depicted in Table 5.2.

Users	Ratings	r	age_{MAX}	δ	τ	n	f
50	ω =2,4,8	3	0	0.95	0.05	10	50
100						20	100
150						30	150
∞		∞				300	600

Table 5.3: Strategies on Ratings Dataset

The experimental settings for the Ratings dataset are given in Table 5.3. The different strategies for Ratings dataset are the different sizes of the sliding window, i.e., $\omega = 2, 4, 8$.

Acronym	Accounts	District	Transaction	r	age_{MAX}	δ	τ	n	f
FIN1-DR	100	20	$\omega = 30$	3	5	0.95	0.2	200	400
FIN2-DR	200	40						200	400
FIN3-DR	300	50						300	600
REF-DR	∞	∞	$\omega = 30$	∞	5	0.95	0.2	400	800

Table 5.4: Strategies on Financial dataset

Trip Variants for Financial Dataset For the Financial dataset, the amount of information that becomes available as the multi-table stream progresses has an impact upon the quality of the classification results. This remembered information is affected by the size of the cache and the sliding window over individual streams. We have thus varied these values for the streams Accounts, District and Transaction.

The strategies we use are depicted in Table 5.4. Strategy FIN2-DR uses a cache of size 200 for the stream Account and of size 40 for the districts, a sliding window of size $\omega = 30$ months for Transaction stream, 3 columns to store nominal values (c.f. Section 3.2.4), $age_{MAX} = 5$ to perform deferred replacement, update tree every $n = 200$ objects, check for concept drift every $f = 400$ objects etc. We test these strategies against our reference strategy that has unlimited storage and knows the future.

Due to large number of moving Account objects, the window size $\omega = 30$ used by TrIP to forget outdated objects (c.f. Section 5.1.2) has little impact. Most objects are frequently updated and never become obsolete. We conducted experiments with $age_{MAX} = 0, 5, 10$. However, we report results for $age_{MAX} = 5$ as only two strategies (i.e., FIN1-DR and FIN3-DR) experience change in performance after timepoint t_{60}.

Evaluation Measure

Our evaluation measure is the "Area under the ROC curve" (AUC), a measure derived from the *Receiver Operating Characteristic*, commonly known as the ROC curve. As the name implies, the ROC curve is a plot: it combines two curves that measure the performance of a binary classifier. In particular, let C_p, C_n be the number of positive, respectively negative examples, let T_p be the number of positive examples recognized as such by the classifier, and let T_n be the number of negative examples recognized as such by the classifier.

Following Bradley [Bradley, 1997], a classifier's *sensitivity* is defined as $P(T_p) = \frac{T_p}{C_p}$, while its *specificity* is $P(T_n) = \frac{T_n}{C_n}$, i.e., the complement of the so-called "α-error" $1 - P(T_n)$. The ROC curve consists of the sensitivity curve and the α-error curve, when the decision threshold is varied.

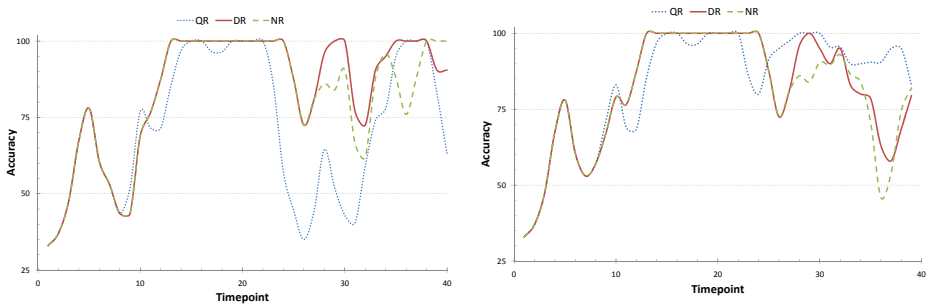

Figure 5.2: The performance of strategies (left) $\omega = 5$, (right) $\omega = 7$ timepoints

The AUC is the area under those two meeting curves; Bradley provides a formula for the computation of this are (Equation 7, page 2 of [Bradley, 1997]). The AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

5.2.3 Experimental Results

In the following, first we report on the experiments on the two synthetic dataset and then the financial dataset, under separate subsections.

Results on Marriage Dataset

In the left part of Figure 5.2 we show the accuracy of strategies QR, DR and NR for $\omega = 5$. From timepoint t_1 to t_{10} all the strategies are in a learning phase: they perform similarly and have identical trees. Until t_6 the main split attribute at the root was *MaritalStatus* and anyone already married was deemed as unsuitable to find a match. By timepoint t_6 the arriving objects are singles, while most of the old ones have become single again. This renders the previous split decision invalid. Strategies QR and DR start growing alternate trees. It needs to be stressed here that the conventional CVFDT would try to forget the concept; however, it is important to keep the concept for the objects that would come in the future and turn to be consistent with this old concept.

At timepoint t_{10} strategy QR replaces the subtree with invalid split decision by the alternate tree with better accuracy. The alternate tree for strategy DR, although it has better accuracy, does not replace the original subtree as it has not yet reached $age_{MAX} = 4$. To adapt to the new

data, strategy DR as well as NR are forced to expand their trees by further split decisions.

From t_{10} to t_{20} all trees stabilize because the objects that arrive are mostly consistent with them. Married individuals cause a fluctuation in the accuracy for strategy NR.

Around t_{22} the new objects cause a drop in accuracy for all strategies. The strategies QR and DR replace their subtrees: it is quite obvious that QR would do a tree replacement; DR is forced to replace the tree because of aging for one of its decision nodes. The drop in accuracy has greater effect than the loss in support (c.f. Equation 5.2). As a result the tree with low accuracy ages faster. And by the time the original tree in DR is overtaken by its alternate tree in terms of accuracy, it reaches age_{MAX} and thus, is replaced.

Around t_{25} objects with *MaritalStatus=true* start arriving again. Strategies DR and NR that have maintained the trees from earlier timepoints show no drop in performance. However, these objects result in drop in accuracy for strategy QR as it had replaced its tree at t_{10}. It starts growing the alternate tree again but it takes time before the outdated information can be replaced.

As new objects with a different concept are introduced around t_{30}, strategy NR's performance deteriorates. While strategies QR and NR recover by replacing their tree with the alternate ones. We expected strategy NR to show the worst performance during this period but even this one recovers. On closely inspecting the results, it became apparent, that the due to an already expanded tree, it is able to manage the incoming data in its leaves. Although the split decision w.r.t. to gain criteria are not valid any more, it still has competent accuracy but is more sensitive due to over fitting.

In the left of Figure 5.2, we show the performance for $\omega = 7$. The strategies behave similarly to $\omega = 5$, except strategy QR whose performance does not deteriorate much during t_{20} to t_{30}. This is probably due to the larger ω as it is able to make better informed split decision based on *MaritalStatus* than in $\omega = 5$.

Results on Ratings Dataset

In the Figure 5.3, we plot the AUC performance for the strategies on Ratings dataset for cache size of 600 and 100 objects. We have varied the window size and kept the size of the cache constant across the experiments. The experimental results for the Ratings dataset are slightly different from the results for incremental clustering (cf. subsubsection *Ratings Dataset*

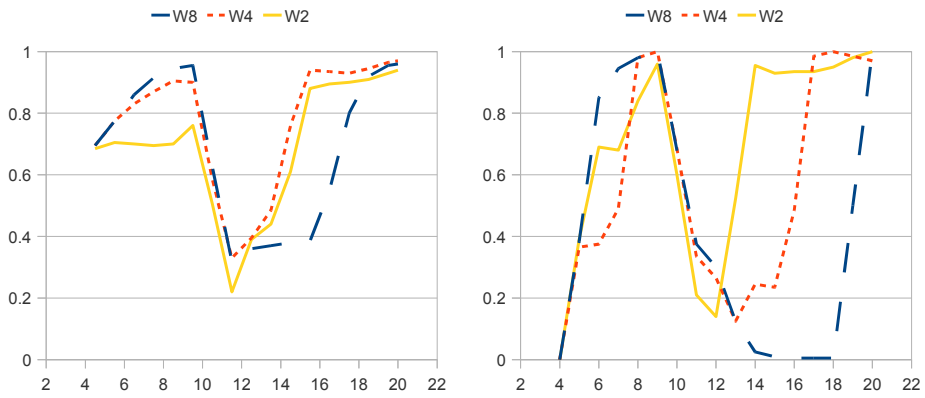

Figure 5.3: Ratings Datset: AUC for the strategies with (left) $N^{\mathcal{T}} = 600$, (right) $N^{\mathcal{T}} = 100$

in Section 4.4.2). During incremental clustering the clustering algorithm utilised all the attributes for learning the model while TrIP only selects a subset of the predictive attributes with the help of information gain and Hoeffding bound. We would get back to this point at the end of this section.

In the left of the Figure 5.3 for cache of 600 objects, the strategies start learn quickly as there are more data available to them then the ones with cache of size 100 (see right of Figure 5.3). The strategies with smaller window sizes have lower performances before the drift occurs at t_{11}.

When the drift happens at timepoint t_{11}, all the strategies report a drop in performance. The strategy with window $\omega = 4$ recovers the fastest around t_{14} (at $\omega = 4$, the oldest data would be from t_{11}, which means that the strategy recovers only after it has forgotten everything that arrived before the drift). One would expect for $\omega = 2$ to recover fastest however, for $\omega = 2$ perennial objects are more fragile and change their values often and quickly. Strategies for $\omega = 8$ and $\omega = 4$ with their large window sizes are more stable as they have more data available to them and the aggregates from propositionalisation change gradually. The strategy $\omega = 8$ starts to recover around t_{16} and by t_{18} recovers completely.

For cache size of 100, the strategies show better overall performance once they have learned the concept. However, because of lesser amount to data available to them, their recovery gets slightly slower, not only after the drift but also at the beginning, where they start building after t_4, whereas for cache size of 600, the strategies already have a small tree at t_4 with a good AUC value. This observation stands in counter to our earlier findings with IncrementalKMeans, where strategies having smaller cache

sizes were better and faster (cf. Section 4.4.2 and Section 4.5). The main reason for this is how TrIP induces the decision tree. It relies on Hoeffding bounds (and the tie-breaker τ) to ensure good splits. The value of Hoeffding bound is a decreasing function in terms of number of objects seen. This means that for larger caches with more objects, the split decisions (either through Hoeffding splits or Tie-breaker splits) are easier and quicker to achieve, while the smaller caches with lesser objects suffer[2]. In contrast, IncrementalKMeans doesn't need to observe a large number of objects to partition them into K clusters.

Financial Dataset

In the Financial dataset, the stream Account contains perennial objects; a cache (c.f. Section 3.2.1) is used to accommodate the most active accounts and the ephemeral transactions on them. Initially, all accounts are empty. As transactions are recorded for an account, it becomes either "loan-risk" or "loan-trusted" class.

Kroegel pointed out that identifiers of the objects from other streams may be useful in some learning tasks [Kroegel, 2003]. In a stream scenario, this may be more likely than in a static scenario, especially if identifiers are generated sequentially. For example, the dataset of KDD Cup 2008 [Perlich et al., 2008] contained identifiers that indicated the class of the patient. Although such correlations must be suppressed in most real-world scenarios, there are scenarios where the proximity of identifier numbers indicate exploitable correlations, e.g. similar age of people or addresses in the same region. Therefore, we report on two experiments for the financial dataset. In one we exploit the identifiers of the ephemeral objects like transactions and bank cards that reference the accounts, while ignore the identifiers from these objects in the other.

In Figure 5.4 we depict the performance of each strategy on the financial dataset with $\omega = 30$. In the left side of the figure, we show the AUC values for strategies with varying cache sizes, when the identifiers from the ephemeral objects not exploited. Initially, the AUC values are zero as TrIP only gathers sufficient statistics about the incoming objects and does not grow the tree.

Around t_{10}, first splits are performed almost simultaneously for all strategies. It must be stressed here that accounts are perennial objects defined by their transactions. Initially there is only static information avail-

[2]For a more insightful discussion about the nature of splits and their quality via Hoeffding bound and Tie-breaking, we refer the work of Holmes et al. [Holmes et al., 2005].

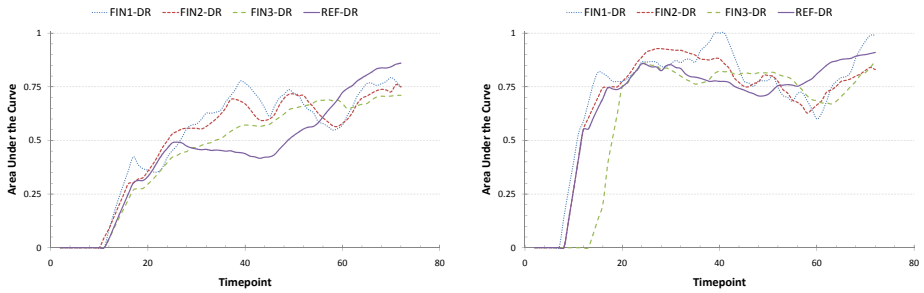

Figure 5.4: Performance of different strategies over financial dataset (left) without IDs (right) with IDs

able about the accounts, e.g., the information about the owner(s) and types of card they hold etc. There is little or no information on transaction. Moreover, the class labels attributed to the accounts reflect their state after many transactions have accumulated on them, i.e., they indicate their final states, which is unknown at early timepoints. For this reason almost all the strategies perform poorly at the beginning.

At the beginning, the arrival of accounts is rather slow. As the stream progress and information accumulates, subsequent splits are performed faster. The first strategy to experience a rise in AUC is FIN1-DR around timepoint t_{25}. FIN1-DR is also the strategy with the smallest cache of 100 accounts. By this time more than 100 accounts have arrived. The propositionalisation algorithm (c.f Section 3.2) keeps those objects inside the cache that are referenced most often, i.e., they have most ephemeral objects associated with them. For FIN1-DR with its small cache size, this means that accounts with fewer transactions and less information are not considered for propositionalisation. It focuses on the informative accounts and shows large AUC. All other strategies have larger caches and store less informative accounts as well. These account cannot be easily classified and result in bad performance. At later timepoints, (i.e., around t_{30}) the strategies FIN2-DR and FIN3-DR also register improvements in their AUCs. They reach their cache size limits and start focusing on mature accounts.

Unlike other strategies, the performance of the Reference (with its infinite cache size), deteriorates between timepoints t_{25} and t_{50}. During this time many accounts with very little information arrive, the Reference strategy remembers all of them.

Although, the strategies with smaller cache sizes (i.e., FIN1-DR and FIN2-DR) show good AUCs during the middle time period, they are also the most unstable ones. Strategies with larger caches (i.e., FIN3-DR and

Reference) have a more stable performance. A possible reason is that due to their smaller sizes, even a single misclassification for FIN1-DR and FIN2-DR gets severely punished by the AUC measure as class distributions are fairly skewed.

After timepoint t_{55}, only very few new accounts arrive, the last one at t_{60}. For the next 12 timepoints, all accounts keep evolving as new transactions arrive for them. The AUC for the Reference strategy also registers a rise towards the end since there are no immature accounts to perturb it. It outperforms all strategies as there is very little information loss.

In the right-side of the Figure 5.4 we depict the performance of the strategies with the same parameter settings but with exploiting the identifiers. By contrasting the results to those at the left side of the Fig 5.4 we see that identifiers from the referring ephemeral objects carry valuable implicit information and all the strategies get a boost in their AUC values. The identifier attribute that conveys most information is the *Maximum Transaction ID* from among the set of transactions performed by an account while other identifiers are also utilized at some timepoint or the other. The relative performance of the strategies with and without identifier are quite similar. The strategies that benefit most are the ones with small caches, i.e., FIN1-DR and FIN2-DR, during t_{35} to t_{45} and t_{20} to t_{35}, respectively. At late timepoints, the Reference strategy again has best performance.

5.3 Conclusion

In this chapter we presented a tree-based classification algorithm, TrIP, which induces a tree over perennial objects. The tree is learned incrementally, where decision to split a node is taken on the basis of Hoeffding bound. TrIP can handle the dynamic nature of the perennial objects. TrIP keeps track perennial objects as they evolve over times. It incorporates method that age a node based on its support and quality and assists it in updating the tree when concept drift occurs.

TrIP was evaluated on synthetic and real datasets. In one set of experiments we experimented with different strategies. The strategies were based on their ability to replace an outdated tree as soon as possible, with a delay controlled through the ageing of the tree and not replace at all. The results showed that in situations when concept recurs and learning horizon (size of sliding) is small, strategy that replaces a tree with showed promising results. The results on the strategies with different sizes of caches for maintaining perennial objects, generally agreed with our earlier

findings (cf. Section 4.4.2 and Section 4.5), where cache size had a direct impact on the quality of the learned model, i.e., strategies with smaller caches had a quick response time and better quality when the data was volatile[3]. The set of experiments, where this observation did not hold, it was discovered to be the peculiarity of the Hoeffding bound induced splits.

[3]when concept drift occurs

CHAPTER 6

Using Classification Rules for Mining Perennial Objects

IncrementalPropositionalisation turns attributes from the streams feeding the perennial stream, into summarised attributes. In doing so, it ignores the correlations that may exist between the original attributes. If these correlations can be identified during the data preparation phase, we can use them for mining. To this purpose we propose classification rule mining over ephemeral streams in the preparation phase. It is shown in the experiments that the new method produces more compact trees.

When a target stream containing perennial objects is transformed into a single stream, the perennial objects are expanded with propositionalised information from the streams that feed the target stream. IncrementalPropositionalisation (cf. Algorithm 3, Chapter 3) summarises each attribute from the other streams, $S^1 \ldots S^J$, separately. It assumes that each attribute contribute individually towards the class label. Such a transformation would be able to preserve a part of overall information, however, the information or the patterns that span more than one attribute, irrevocably get lost.

Name	Course Discipline	Course ID	Grade	Short-listed
John	Economics	EC-101	3.3	-
	Maths	MT-109	1.0	
	History	HT-209	1.3	
Doe	Economics	EC-101	1.0	+
	Maths	MT-211	1.3	
	History	HT-209	3.3	
Mary	Economics	EC-101	1.7	+
	Economics	EC-309	1.7	
	History	MT-209	4.0	

Figure 6.1: Example: Applicants and their undergraduate examination records. Grading scale is from 1.0 to 4.0, with 1.0 being the best grade.

More concretely, consider an automated graduate admission process

at a university. The process short lists the promising applicants for an economics graduate course. The short listing criteria requires an applicant to have achieved good grades (at most 2.0 or lower) in their undergraduate economics courses. The applicants and their respective grades are shown in Figure 6.1.

The applicants have a 1-to-m relationship with the courses that the have attended. The propositionalised representation of the applicants is shown in Figure 6.2. Such transformation is able to preserve the general information about the applicants, i.e., what type of courses did they attend and what was their overall performance during their study period but details on how an individual fared in each course gets lost. The propositionalised vectors of Doe and John are identical, even though former is a short list candidate while the latter is not. There is not enough information available for the automated process to separate the applicants.

Name	**Category**			**Grade**				**Short-listed**
	c_{eco}	c_{math}	c_{hist}	Min	Max	Avg	Sum	
John	1	1	1	1.0	3.3	1.9	5.6	-
Doe	1	1	1	1.0	3.3	1.9	5.6	+
Mary	2	1	0	1.7	4.0	2.5	7.4	+

Figure 6.2: Propositionalised representation of the applicants' data from Figure 6.1 (where, c_v is the frequency of a value v observed for a certain).

In the next sections, we propose the generation of new attributes through classification rule mining for the ephemeral or fast stream. These attributes encompass patterns that span across more than one attribute and have potentially high predictive power. We enhance TrIP to exploit these attributes during learning.

6.1 Challenges of Rule Mining over Streams

As discussed in Chapter 2 (Section 2.2.4), there are many algorithms on classification rule mining over streams. Since the number of rules to be discovered is exponential to the number of attributes in the stream, all these algorithms encompass solutions to the problem of space demand. Several techniques have been used in the literature to reduce number of itemsets discovered from the data without sacrificing quality.

Setting an optimal value for support threshold is important for limiting the number of the discovered itemsets, however, it is not trivial. Several techniques have been proposed in the literature that uses a concise

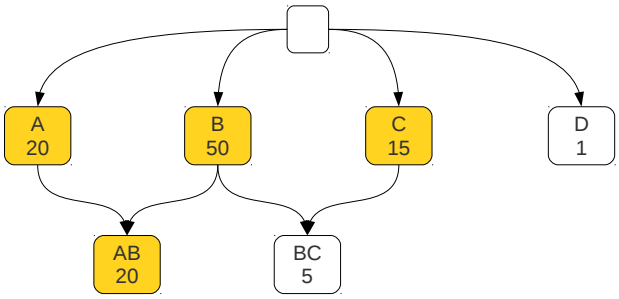

Figure 6.3: A set of discovered itemsets. The numbers depict the support count for each itemset and support threshold is 10. The frequent itemset are shown as shaded boxes.

representation of the itemsets, instead of relying on the optimal value for support. Veloso et al. [Veloso et al., 2002] stores only the *maximally frequent itemsets*, while Pasquier et al., Zaki and Hsiao and Chi et al. [Pasquier et al., 1999; Zaki and Hsiao, 2005; Chi et al., 2006] uses the notion of *closed itemsets* in order to achieve a concise representation. In Figure 6.3, a set of frequent itemset is depicted and in Figure 6.4 we show their concise representation. In the context of approximate stream mining, the method of Boettcher et al. [Boettcher et al., 2009] extends the notion of closedness to *approximately closed* and *temporally closed*.

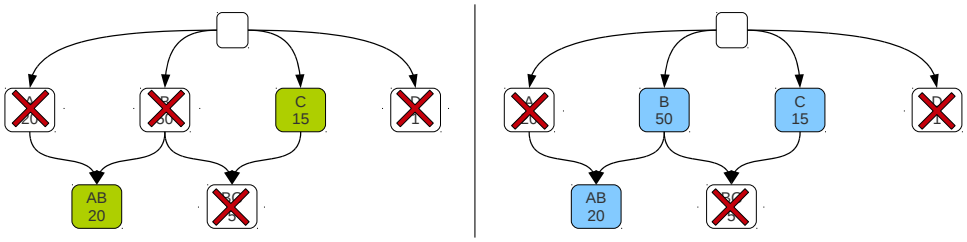

Figure 6.4: Concise representation of the frequent itemset from Figure 6.3, (left) maximally frequent itemsets and (right) closed itemsets.

6.2 Overview

Our objective is to learn a classifier over a stream $\mathcal{T}$ of perennial objects. We propose *the Classification Rule Miner for a stream of Perennial Objects*, CRMPES that learns a lattice $\mathcal{L}$ of classification rules over the streams $S^1, \ldots, S^J$ incrementally, derives predictive attributes for the class labels

Table 6.1: List of used terms and symbols.

Notation	Description
$\mathcal{C}$	set of class labels $\{l_1, l_2, \ldots, l_o\}$
$\mathcal{I}$	classification rule $X \rightarrow [p_1, p_2, \ldots, p_o]$: X is the antecedent (a set of (attribute,value)-pairs), and p_u is the number of tuples with antecedent X and label l_u, i.e. the "contribution" of $\mathcal{I}$ to l_u ($u = 1, \ldots, o$)
$\mathcal{L}$	lattice of classification rules
$\mathcal{F}$	set of features extracted from the lattice $\mathcal{L}$ for the classification of the stream $\mathcal{T}$, where a feature $F \in \mathcal{F}$ is the antecedent of a classification rule
g	maximum number of features to be extracted from $\mathcal{L}$.
σ^{min}	minimum support a rule must have, before adding a tentative rule below it to $\mathcal{L}$
$e(\mathcal{I})$	entropy of $\mathcal{I}$,: $\sum_{u=1}^{o} \rho_u log \rho_u$, where $\rho_u = \frac{p_u}{\sum_{l=1}^{o} p_l}$
$d(\mathcal{I})$	d-score of $\mathcal{I}$ as its *un*-interestingness
d^{min}	minimum permitted d-score of a rule
δ	confidence threshold for significance computations

$\mathcal{C}$ in $\mathcal{T}$ and delivers them to the tree classifier TrIP (cf. Algorithm 6 in Chapter 5).

CRMPES operate as follows. At each timepoint t_i, the labels $\mathcal{C}$ from $\mathcal{T}$ are propagated to the arriving objects of the other streams $S^1, \ldots, S^J$, and the rules in the lattice are updated, as described in Subsection 6.3. The lattice is grown incrementally, thereby removing uninteresting rules, as explained in Subsection 6.4. Next, attributes are generated from the learned lattice, as discussed in Subsection 6.5. IncrementalPropositionalisation is invoked next to generate the propositionalised attributes for the stream classification algorithm. This input, together with the classification rules discovered at t_i are considered for stream classification, as explained in Subsection 6.6. The basic notations and symbols are same as that from Chapter 3 and Chapter 5 (c.f. Table 3.1 and Table 5.1). Additional symbols and parameters are presented in Table 6.1.

6.3 Aligning the Ephemeral Streams

Let $\mathcal{T}$ be the stream of perennial objects, and let $S^1, \ldots, S^J$ be the streams feeding it. The label of each object $x \in \mathcal{T}$ is propagated to all the objects

that reference x from the streams $S^1, \ldots, S^J$, i.e., $matches(x)$. Similarly to the propagation of the target identifiers (cf. Section 3.2.2), propagation of the target labels has often been used in the literature to discover patterns from the other streams [Srivastava and Gore, 2008; Yin and Han, 2005; Yin et al., 2006].

For example, assume that we classify customers by lifetime value into classes A, B, C, D; let customer x at timepoint t_i belong to class B: B becomes the label of all transactions performed by x that are inside the sliding window.

A classification rule $\mathcal{I}$ has the form $X \rightarrow [p_1, p_2, \ldots, p_o]$, where X is the antecedent of the rule and contains is a set of literals based on (attribute, value)-pairs in accordance to the schema of an ephemeral stream, and p_u is the number of objects supporting X and having the label $l_u \in \mathcal{C} = \{l_1, \ldots, l_o\}$. The generation of literals or (attribute, value) -pairs for nominal attributes is straight forward. For learning the literals of numerical attributes we use the same approach as presented for maintaining the sufficient statistics of numerical attributes (cf. Section 5.1.1). We build a lattice $\mathcal{L}$ of classification rules incrementally as described in the next subsection.

The operation of stream alignment, i.e., target label propagation and lattice updating, is depicted in Algorithm 11: AlignStreams takes as input the currently observed object y, the cache of perennial objects $\mathcal{T}_i$ and the current state of the lattice $\mathcal{L}$. Intuitively, the algorithm reads the label of the perennial object x referenced by y (Line 1), finds all classification rules supported by object y and, for each such rule $\mathcal{I}$, it update the support p_u for the label l_u by factor cnt (Line 6). Variable cnt can take either of the two values from $[1, -1]$. In Section 6.4, we explain how the value for cnt is determined and what *tentative* rules are.

6.4 Building a Lattice of Classification Rules

The core of our Classification Rule Miner, CRMPES_Core (cf. Algorithm 12), grows the lattice $\mathcal{L}$ of classification rules incrementally. At timepoint t_i, CRMPES_Core considers the objects in $\cup_{j=1} S_i^j$. Some of these objects will be forgotten later, i.e. they *exit* the sliding window, while new ones, yet unseen in t_i, will *enter* the sliding window. Let OLD_i be the objects to be forgotten at t_i, and NEW_i be the new objects at t_i.

CRMPES_Core takes as input (1) the objects in OLD_i and NEW_i, (2) the objects $\mathcal{T}_i$ referenced by these objects, (3) the lattice $\mathcal{L}$ built thus far, and (4) thresholds σ^{min} on the support of classification rules, d^{min} on their

Algorithm 11: AlignStreams

Input : $y, cnt, \mathcal{T}_i, \mathcal{L}$
Output: $\mathcal{L}$

```
1 x ← perennial object in T_i that is referenced by y
2 l_u ← label of x
3 foreach I ∈ L do
4     if y supports I then
5         if I is not "tentative" then
6             I.p_u ← I.p_u + cnt
7 return L
```

interestingness, and δ on the confidence in estimating rules' redundancy[1]. CRMPES_Core traverses $\mathcal{L}$ from the root downwards, *grows* it by adding rules and *shrinks* it by removing rare, uninteresting and redundant ones, as we explain below.

6.4.1 Update Lattice with new objects

(Line 1) CRMPES_Core starts processing $\mathcal{L}$ by first updating the supports of the rules in it. It invokes AlignStreams for each new object y, whereupon the label of y is identified (cf. Algorithm 11, Line 1), and the contribution of each rule supported by y to this label is increased by $+1$ (cf. invocation of AlignStreams in Algorithm 12, Line 1).

6.4.2 Grow Lattice with new rules

(Lines 2-7) Arriving objects in NEW_i may give raise to new rules that are not yet in the lattice. CRMPES_Core grows the lattice *pro-actively*. First, it identifies all classification rules that have no children and are supported by at least σ^{min} objects thus far (Lines 6) and then *expands* them (Line 7) by *tentative* rules (Line 8).

Lattice *expansion* is only possible for rules that have no common child yet, and whose antecedents differ by only one attribute. For each such a pair of rules, with antecedents XB and XD respectively, a common child with antecedent XBD is created and marked as "tentative". From this moment on, the contribution of XBD to the label of each arriving object y

[1]check for rule redundancy is synonymous with check *closedness* of an itemset (see right of Figure 6.4).

Algorithm 12: CRMPES_Core

Input : $OLD_i, NEW_i, \mathcal{T}_i, \mathcal{L}, \sigma^{min}, d^{min}, \delta$
Output: $\mathcal{L}$

1 **foreach** $y \in NEW_i$ **do** AlignStreams($y, +1, \mathcal{T}_i, \mathcal{L}$) ; /* UPDATE */
2 $E \leftarrow \emptyset$
3 **foreach** $\mathcal{I} \in \mathcal{L}$ **do** /* GROW */
4 **if** $\mathcal{I}$ *has no children* $\wedge$ ***not tentative*** $\wedge$ ***not locked*** **then**
5 **if** $\sum_{u=1}^{o} \mathcal{I}.p_u \geq \sigma^{min}$ **then**
6 add $\mathcal{I}$ to E
7 Expand $\mathcal{L}$ by creating new rules as children of the rules in E.
8 Mark the new rules as **tentative**.
9 **foreach** $y \in OLD_i$ **do** AlignStreams($y, -1, \mathcal{T}_i, \mathcal{L}$) ; /* UPDATE */
10 **foreach** $\mathcal{I} \in \mathcal{L}$ **do** /* SHRINK */
11 let $\mathcal{I}'$ be the child of $\mathcal{I}$
12 **if** $\mathcal{I}$ *is "redundant" towards* $\mathcal{I}'$ **then**
13 mark $\mathcal{I}$ as **locked**
14 redirect accesses to $\mathcal{I}$ towards $\mathcal{I}'$
15 **else if** $\sum_{u=1}^{o} \mathcal{I}.p_u < \sigma^{min}$ *OR* $e(\mathcal{I}) - e(\mathcal{I}') < \epsilon$ **then**
16 mark $\mathcal{I}$ as **locked**
17 remove all children of $\mathcal{I}$ from $\mathcal{L}$
18 unmark all **locked** rules that are redirecting towards $\mathcal{I}$
19 **else** umark $\mathcal{I}$
20 return $\mathcal{L}$

is increased whenever AlignStreams is invoked (cf. Algorithm 11, Line 6). However, the lattice cannot grow below a tentative rule (cf. Algorithm 12, Line 4), i.e. a tentative rule cannot acquire a child until it stops being "tentative" (cf. Line 19). We explain the treatment of tentative rules in sequel.

6.4.3 Update Lattice with old objects

(Line 8) When an object exits the sliding window, the rules it supports must be modified: CRMPES_Core invokes AlignStreams for each object $y \in OLD_i$ with a count value of -1, so that the support of the rules is decreased (cf. Algorithm 11, Line 6). An exception is made for tentative

rules whose support is not decreased.

The reason for this exception lays in the behaviour of the sliding window. Let t_i be the timepoint of creating a new tentative rule $\mathcal{I}_{XBD}$. To compute its support, all $\cup_{j=1} S_i^j$ objects should be considered. However, CRMPES_Core processes only the objects that arrive new at t_i (cf. Line 1), while those that have already been seen are not reconsidered – although they are still inside the sliding window. Since the objects that are getting removed have no contribution towards the support of the tentative rule, its support is not decreased. Once all such objects have exited the window, tentative rules are unmarked, provided of course that they are worth retaining in the lattice: CRMPES_Core uses several criteria to shrink the lattice by eliminating useless rules.

6.4.4 Shrink Lattice by removing useless rules

(Lines 9-17) CRMPES_Core uses three criteria to assess the usefulness of a rule: the rule's support, the rule's predictive power, and the rule's redundancy with respect to its children. Because of the fact that number of rules that can be discovered from stream of ephemeral objects is very large, we use these criteria to limit the size of lattice $\mathcal{L}$.

A rule $\mathcal{I}'$ is *redundant* if it is parent of a rule $\mathcal{I}$ and its support (or class distribution) is not significantly higher than of $\mathcal{I}$. Then, the parent rule $\mathcal{I}'$ is marked as "locked" (Line 13), and all accesses to it are redirected to $\mathcal{I}$ (Line 14). For the significance test, we use $\chi^2 = \sum_{u=1}^{o} (p'_u - p_u)^2 / p_u$ (with critical value $\chi^2_{1-\delta}$), where p_u, p'_u are the contributions of $\mathcal{I}$, $\mathcal{I}'$, respectively, to label $l_u, u = 1 \ldots o$.

If the support of a rule $\mathcal{I}$ remains/drops below the threshold σ^{min}, then the rule is also marked as "locked" (Line 16). This means that the lattice cannot grow below it (Line 7). Further, all existing rules below $\mathcal{I}$ (i.e. being more specific and thus having no higher support than $\mathcal{I}$) are removed from $\mathcal{L}$ (Line 17). In contrast, rules that were earlier "locked" and redirecting to the now locked rule $\mathcal{I}$ are themselves unlocked (Line 18).

A rule $\mathcal{I}$ may have adequate support but its predictive power might not be significantly better than its parent $\mathcal{I}'$. Such rules are also marked as locked. The computation is based on the entropy of a rule $\mathcal{I}$: $e(\mathcal{I}) = \sum_u \rho_u log \rho_u$, where $\rho_u = p_u / \sum_l p_l$. The significance test is done using the Hoeffding bound [Catlett, 1991; Domingos and Hulten, 2000]. It states that, if $e(\mathcal{I}) - e(\mathcal{I}') < \epsilon$ for some threshold ϵ, then the rule $\mathcal{I}$ is not significantly better than $\mathcal{I}'$.

Classification rules that have high support and predictive power are re-

tained in the lattice. If they were marked earlier (as "tentative" or "locked"), they are unmarked (Line 19). Retaining them in the lattice may turn other rules *redundant* though: it is possible that a small number of rules suffices for distinguishing among the classes, while the rest are redundant.

We also model the *interestingness* of a rule $\mathcal{I}$ by combining the rule's support and entropy $e(\mathcal{I})$ (cf. Table 6.1).

We define:

$$d(\mathcal{I}) = \frac{\sum_{u=1}^{o} p_u}{w} \times (1 - e(\mathcal{I})) \tag{6.1}$$

where a rule is more interesting, the higher its $d()$-score is.

CRMPES_Core uses the score $d()$ to rank and to eliminate the rules with scores lower than d^{min} from $\mathcal{L}$. Uninteresting rules are treated similarly to those with very low support. The weight of a rule's support ratio inside the sliding window with the 1-complement of the rule's entropy is intended to prevent rare rules with possibly low entropy from acquiring high $d()$ scores.

6.5 Generating Features from the Rules' Lattice

The lattice of classification rules built and maintained by CRMPES_Core is used to generate additional features for the learning task over the stream of perennial objects $\mathcal{T}$. A *feature* f is not a single attribute but rather the antecedent X of some classification rule $\mathcal{I}$ in the lattice $\mathcal{L}$, i.e. a set of (attribute,value)-pairs, as specified in Table 6.1.

Feature generation takes place at each timepoint t_i. The Feature Generation algorithm CRMPES_FGen (cf. Algorithm 13) takes as input the lattice $\mathcal{L}$ built by CRMPES_Core thus far, and it incrementally builds a set of generated features $\mathcal{F}$ of cardinality g.

First, CRMPES_FGen ranks the classification rules in $\mathcal{L}$ on their interestingness $d()$-score, as defined in Eq. 6.1, thereby skipping closed and tentative rules (Line 1). Let R be an ordered list of ranked rules (Line 2), so that $R[k]$ is the rule at position k with $k = 1, \ldots, |R|$, and $R[k].antecedent$ is this rule's antecedent. At the first iteration, CRMPES_FGen selects the antecedent of the top-ranked classification rule (Line 3) and removes it from R (Line 5). At the m^{th} iteration, the rule of highest rank is chosen among those that have an empty intersection of (attribute, value)-pairs to the set SF, i.e., to the set of (attribute,value)-pairs already in $\mathcal{F}$ (Line 7). Then, the sets and the list R are updated accordingly (Lines 8-10).

In fact, the selection of the m^{th} rule is slightly more general than depicted in Line 7. If CRMPES_FGen finds no rule with empty intersection

Algorithm 13: CRMPES_FGen

Input : $\mathcal{L}, g$
Output: $\mathcal{F}$

1 $\mathcal{L}' \leftarrow \{\mathcal{I} \in \mathcal{L} | \mathcal{I}$ is not "closed" and not "tentative"$\}$
2 $R \leftarrow$ rank $\mathcal{L}'$ on $d()$-score descending
3 $\mathcal{F} \leftarrow \{R[1].antecedent\}$
4 $SF \leftarrow R[1].antecedent$
5 remove$(R, R[1])$
6 **for** $m = 2, \ldots, g$ **do**
7 $\quad \mathcal{I} \leftarrow argmin_m\{R[k] antecedent \cap SF = \emptyset\}$
8 $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{\mathcal{I}.antecedent\}$
9 $\quad SF \leftarrow SF \cup \mathcal{I}.antecedent$
10 $\quad$ remove$(R, \mathcal{I})$
11 return $\mathcal{F}$

to SF, it considers rules in R that have minimal intersection. This implies additional scans over the list R. However, the additional cost is low, since R becomes smaller at each iteration (Lines 5, 10).

6.6 Enhancing TrIP with Rule-based Features

The algorithm TrIP receives as input a propositionalised stream $\mathcal{P}$, the schema of which is a large set of derived attributes (Chapter 3). With CRMPES, we deliver an additional set of features, derived from classification rules of high predictive power. We thus enhance TrIP by allowing it to choose from a larger list attributes for learning.

The new algorithm, as invoked at each timepoint t_i is outlined in Algorithm 14. At each timepoint, it determines the set of new objects NEW_i that arrive at t_i and old objects OLD_i that leave the sliding window, i.e., the ones that arrived at timepoint $t_{i-\omega}$ (Line 4). It invokes CRMPES_Core to get the lattice of the discovered rules $\mathcal{L}$ (Line 5). The lattice $\mathcal{L}$ is then passed to CRMPES_FGen, which derives the set of predictive features $\mathcal{F}$ (Line 6). To get the basic propositionalised features, IncrementalPropositionalisation is invoked (Line 7). The propositionalised data and the data from the derived set of features is concatenated and together and TrIP is invoked over this concatenated data (Line 8). $\mathcal{R}$ is the model learned and adapted by TrIPfrom one timepoint to the next.

Algorithm 14: CRMPES_TrIP

Input : $\langle\, \mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon \,\rangle$, $\langle\, \omega, n, f, l^{min}, s^{min}, \delta, \tau \,\rangle$, σ^{min}, d^{min}, g
Output: ζ

$\mathcal{L}_0 \leftarrow \emptyset$
$\mathcal{T} \leftarrow S^0$
foreach *timepoint* t_i **do**
  Compute OLD_i, NEW_i using ω from $S^1 \ldots S^J$
  $\mathcal{L}_i \leftarrow$CRMPES_Core($OLD_i, NEW_i, \mathcal{T}_i, \mathcal{L}_{i-1}, \sigma^{min}, d^{min}, \delta$)
  $\mathcal{F}_i \leftarrow$CRMPES_FGen($\mathcal{L}_i, g$)
  $\mathcal{P}_i \leftarrow$IncrementalPropositionalisation($t_i, \mathcal{X}^B, \mathcal{S}, \mathcal{N}, \omega, r_A, \beta, \epsilon$)
  $\mathcal{R}_i \leftarrow$ TrIPCore($t_i, \mathcal{R}_{i-1}, \mathcal{P}_i \oplus \mathcal{F}_i,\ \omega, n, f, l^{min}, s^{min}, \delta, \tau$)

6.7 Experiments

In the experiments we study the quality of the rules discovered by CRMPES and the effect of the derived attributes on the quality of the final stream classifier.

6.7.1 Datasets

For the evaluation we have uses synthetic datasets and a real dataset on learning user profiles. The synthetic datasets have been generated from the Multi-Gen generator (Chapter 7). Multi-Gen creates a stream of users and adjoint streams of items and user ratings. Each user in the streams of users adheres to a user profile. A user profile determines the affinity of a user for an item descriptor: a generated rating for an item depends on the user's affinity for an item, as set in her profile. A user may change from a profile to another (drift). The learning task in these datasets is to predict each user's profile at each moment, given the ratings thus far.

We specified 6 user profiles and 8 item descriptors described by 5 nu-

Table 6.2: Description of Synthetic Datasets

Name	**Description**	Drift
Ratings 1500r	1000 users, 800 items, 1500 $\frac{transaction}{timepoint}$	No Drift
Ratings 6500r	1000 users, 800 items, 6500 $\frac{transaction}{timepoint}$	No Drift
Drift 150u	150 users, 800 items	@ t_{30}
Drif 1000u	1000 users, 800 items	@ t_{50}

merical attributes for all the datasets. We grouped the profiles into two classes (á three profiles). The short description of the dataset is given in Table 6.2. The complete parameters' specifications for the synthetic datasets are provided in Appendix A.2.

The real dataset is Financial dataset from PKDD 1999 Challenge on predicting defaults in bank loans. We have described the dataset in Section 4.1 of Chapter 4.

6.7.2 Evaluation Framework

For the experiments we have used three strategies: **P** uses only attributes generated by the incremental propositionalisation, while **R** uses only rule-based attributes delivered by CRMPES, and **PR** uses both. Essentially, strategy **P** corresponds to TrIP, as proposed in Chapter 5 and is our baseline. The strategies are listed in Table 6.3.

Table 6.3: Evaluation Strategies

Strategy	**Description**
R	CRMPES+ TrIP
P	IncrementalPropositionalisation + TrIP
PR	IncrementalPropositionalisation + CRMPES + TrIP

The complete list of parametric values for all datasets is provided in Table 6.4.

We evaluate the strategies using the Area under the Curve (AUC) measure (cf. Section 5.2.2). We use prequential evaluation to evaluate the strategies in all experiments except for the Figure 6.5, where we use a hold out part of data stream for the evaluation.

Table 6.4: Experimental settings for synthetic and real datasets.

Datasets	ω	σ^{min}	d^{min}	δ	τ	n	f	g	r
Ratings 1500r	30	0.05	0.4	0.95, 0.99	0.01	500	1000	8	$\emptyset$
Ratings 6500r									
Drif 1000u									
Drift 150u						75	150		
Financial		0.01	0.1	0.95	0.2	200	400		3

6.7.3 Experimental Results

In this section we first present the results on synthetic datasets and then on real dataset.

Learning User Profiles on Synthetic Data

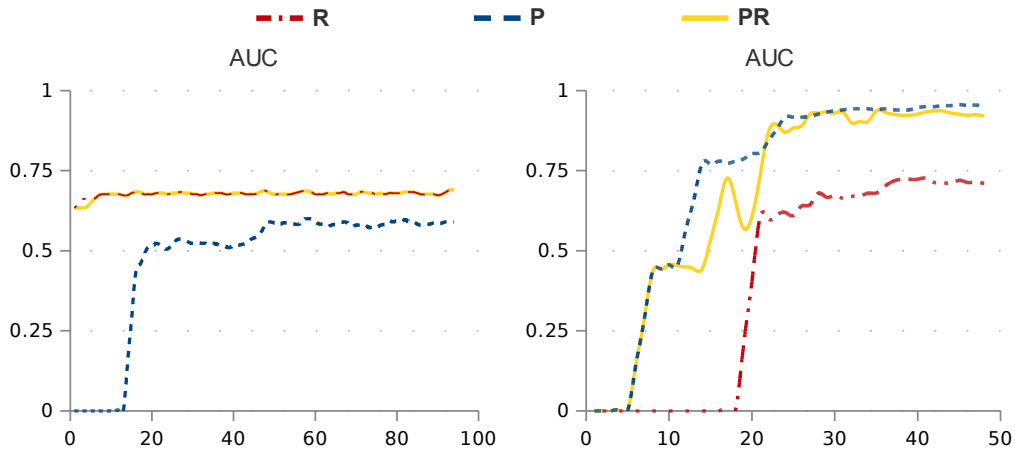

Figure 6.5: AUC plots for (left) Ratings 6500r and (right) Ratings 1500r datasets.

In Figure 6.5 we plot the AUC for the strategies P, R and PR for the datasets *Ratings 1500r* and *Ratings 6500r*. Both the datasets do not under go any form of concept drift. The nature of both the datasets is different (see Appendix A.2 for details). In the dataset *Ratings 6500r* the attributes are correlated towards the profiles or class labels. PR shows a good a performance for both the datasets. On the other hand, R and P experiences a drop in performance for the datasets *Ratings 1500r* and *Ratings 6500r*, respectively. The better performance of PR for both the datasets is mainly due to the advantage that it draws from utilising the rich feature space, i.e., attributes from IncrementalPropositionalisation and CRMPES feature generation. Strategies P and R utilise only one set of attributes (either from IncrementalPropositionalisationor from CRMPES, respectively) and show a relatively lower performance when the concept becomes complex. Strategy PR although has competitive performance for both the datasets, but its performance during t_{10} and t_{20} is inferior to strategy P. The main reason is the availability of a large number of attributes for PR (i.e., from CRMPES and IncrementalPropositionalisation). Large number of attributes makes it harder for PR to reach split decisions quickly because of the presence of

ties in the split. However, the final resulting tree of PR was shorter than that of P. We explore this further in the next experiments.

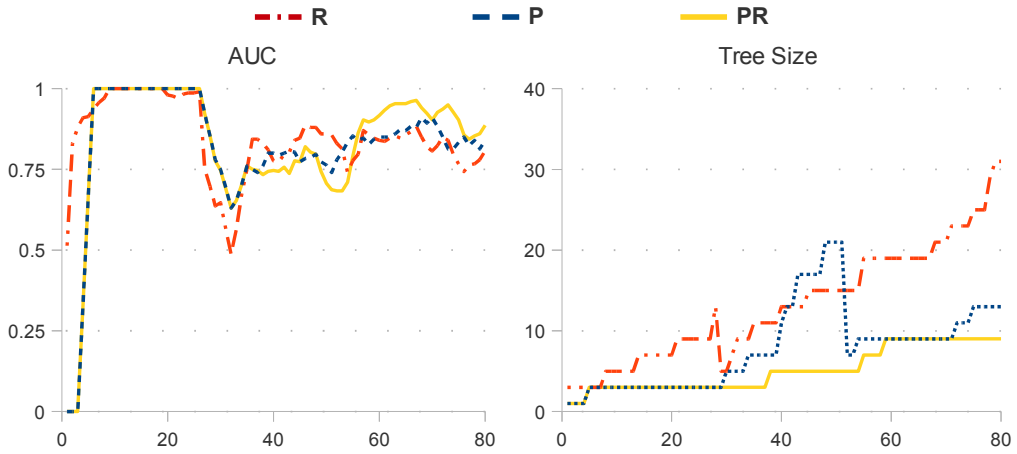

Figure 6.6: AUC and Tree size plots for Drift 150u dataset.

In Figure 6.6, we plot the AUC values and the tree size of results on *Drift 150u* dataset. At the beginning, all the strategies show good performance, since the classes are rather easy to learn (cf. AUC plot in Figure 6.6). At timepoint 30 we have imputed concept drift, which causes a drop in performance for all the strategies. After the drift, strategy R shows the fastest recovery. Strategy PR takes sometime to start growing its tree after the concept drift, when it does around timepoint t_{56}, it shows the best performance. During this time, the other strategies show a comparatively unstable behaviour (cf. Tree size plot in Figure 6.6). Usage of attribute from CRMPES alone, results in very large trees for R, more so after the concept changes at timepoint 30. The tree learned by P is consistently much larger than the tree learned by TrIPupon **PR**. Hence, **PR** achieves comparable or better predictive performance than the baseline, while learning smaller and less volatile trees, where as, P and R constantly split their leaf nodes in order to improve their quality but are unable to do so.

In Figure 6.7, we plot the AUC values and the tree size of results on *Drift 1000u* dataset. with 1000 users. Here, the drift occurs around t_{50} and is the only place where **PR**'s tree grows to adjust to the drift. Strategy **R** is unable to learn the concept before the drift and shows a volatile behaviour after. Strategy **P** overtakes **PR** during $t_{50} - t_{70}$, but it is more of over-fitting than better performance as its tree is big and is forced to adjust in the subsequent time points.

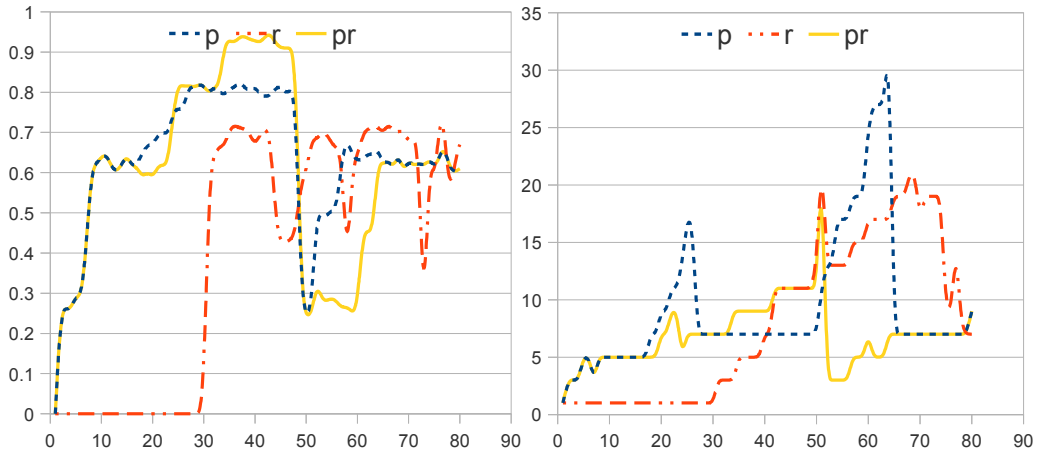

Figure 6.7: AUC and Tree size plots for Drift 1000u dataset

Predicting Defaults in Bank Loans

The dataset "Financial" of the PKDD 1999 Challenge contains data on bank customers, the transactions of their bank accounts and static data on the districts they come from. The data are timestamped, so we can transform them into the dataset `District`, the stream `Transaction` and the slower stream `Account`, which contain the customer data is the target stream. The learning task is to predict whether a customer (represented by her Account) will default in paying back her loan. The originally 4 classes were already merged into two (*loan-trusted* and *loan-risk*) during the 1999 Challenge.

In Section 4.1 Chapter 4, we have stressed that the "Financial" dataset exhibits a difficult learning problem: the class distributions reflect the state of the accounts only when they have matured. Hence, labels become applicable at a much later timepoint than when the objects are introduced. So, we expect low performance at the first timepoints. We also point out that the amount of data cached per stream affects performance. We experimented with similar settings.

We used only the baseline **P** and strategy **PR**. We see in Figure 6.8 that they perform similarly. Inspection of the CRMPES output showed that there are some useful rules, e.g. `balance within [-600, 600] AND penalty imposed` $\rightarrow$ *`loan-risk`*, but they were not very frequent and were not utilised by TrIP. An explanation is also the nature of the dataset: the labels are predictable only when the accounts mature, i.e. rather late. Further, the `Financial` dataset contains many nominal attributes that can be exploited directly by TrIP; numerical attributes, as in the

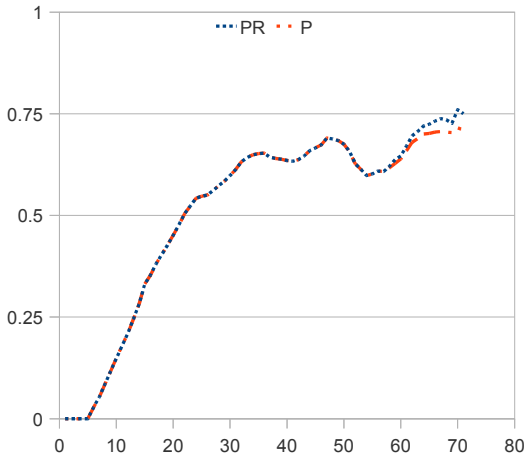

Figure 6.8: AUC and Tree size plots for *Financial* dataset with $\omega = 30$, size of *Account* $N^{account}$=200 and size of *District* $N^{district}$=40

synthetic dataset, are more difficult to exploit in a decision tree, so CRMPES brings higher advantage.

6.8 Conclusion

In this chapter we have presented an improvement to the basic IncrementalPropositionalisation. IncrementalPropositionalisation in its basic variant, uses simple SQL[2] aggregates to generate the propositionalised attributes, e.g., sum, avg, min, max. We enrich it with attributes that are based on classification rules that can capture complex information from the data. We have presented a classification rule algorithm CRMPES that operates on the ephemeral streams that feeds the target stream $\mathcal{T}$. The labels for the ephemeral objects come from the perennial objects that they reference. CRMPES learns rule incrementally and maintain them in a lattice. The lattice is maintained in a compressed form. CRMPES_FGen then selects g rules to be used as feature during propositionalisation.

In the results, we experimented with different strategies that utilised different sets of attributes: propositionalised attributes, rule based attributes and both. The strategy that utilised attributes both propositionalised and rule based attributes showed competitive performance over all the experiments. It was more stable and learned the shortest trees. This was expected as splits based on complex patterns (rule-based attributes) carry

[2]Structure Query Language

more information and provide better information gain.

For numerical attributes, the literals they add to the antecedent part of the rules are based on numerical bins that we compute using first 100 values seen for numerical attributes. We don't use binary splitting for the literals of nominal attributes but use the generated bins. An improved handling of the numerical attributes as proposed in [Gama et al., 2003, 2004], coupled with binary splits, would lead to the discovery of more informative rules and would likely result in even better results.

CHAPTER 7

Multi-Relational Data Generator for Streams

Overview

In this chapter we present the generator of 'user profiles', which we used in Chapter 4, Chapter 5 and Chapter 6. We call it Multi-Gen: it is a generator of *multiple interrelated streams*, appropriate for testing classification and clustering algorithms on concept discovery and on adaptation to concept drift, both for mobile and web applications. The number of concepts in the data can be specified as parameter to the generator; the same holds for the membership of an instance to a class. Hence, the generator can also create data sets with overlapping concepts (classes or clusters). Although Multi-Gen is mainly designed to learn concepts of the temporal objects, as they drift with the streams of transactions, it can also be used to learn a model of the conventional transaction stream itself.

7.1 Introduction

Most of the data stored in databases, archives and resource repositories are not static collections: they accumulate over time, and sometimes they cannot (or should not) even be stored permanently - they are observed and then forgotten. Many incremental learners and stream mining algorithms have been proposed in the last years, accompanied by methods for evaluating them [Gama et al., 2009; Bifet et al., 2009]. However, modern applications ask for more sophisticated stream learners than can currently be evaluated on synthetically generated data. In this work, we propose Multi-Gen, a generator for complex stream data that adhere to multiple concepts and exhibit drift. Multi-Gen can be used for the evaluation of (multi-class) stream classifiers, stream clustering algorithms over high-dimensional data and relational learners on streams.

Multi-Gen is inspired by recommendation engines [Symeonidis et al., 2008; Adomavicius and Tuzhilin, 2005], where data are essentially a combination of perennial objects and adjoint streams: people rank items - the

rankings constitute a *fast* stream of ephemeral objects; new items show up, while old items are removed from the user's portfolio - the items constitute a *slow* stream of perennial objects; new users show up, while old users re-appear and rank items again, possibly exhibiting different preferences as before - users also constitute a *slow* stream of perennial objects. Recommender systems propose items to users based on different types of information (i.e. users' friendships, their comments on items, their ratings on items, their tags on items etc.), which evolve over time. There are several web sites such as Amazon or Netflix, which allow users to rate, tag and comment on items. Recognizing the patterns that govern these multi-streams data as they evolve, enables us to better understand users' behaviour.

The idea of data generation and drift imputation in Multi-Gen is as follows. Multi-Gen generates three streams: the stream of Users, which is also the *Target Stream*, the stream of Rankings and the stream of Items (perennial) feed the target stream. The preference of a user towards some item(s) defines its behaviour. Multiple users' exhibiting similar behaviour can be grouped/categorised together as a single user profile. Conversely to learning task where these profiles are learned from among a group of users, Multi-Gen first creates these user profiles which serve as prototypes. These profiles are then used to generate individual user data according to the item preferences stored in them. Noise can be imputed to the data by forcing a user to rank in discordance to her profile with some probability. Drift is imputed to the data by allowing a profile to exist only for some timepoints and then forcing it to mutate to one or more profiles with some probability.

7.2 Related Work

Our generator is inspired by the demands of a recommendation engine, a prominent one is learning under drift. We build on a generator for recommenders proposed by Symeonidis et al. [Symeonidis et al., 2010] for a static concept. Concept drift is intensively studied in the context of stream mining. We discuss generators that simulate a drifting stream, after describing the static-concept generator of [Symeonidis et al., 2010].

7.2.1 Generating a Static Concept for Recommendations

The generator of [Symeonidis et al., 2010] produces a unipartite user-user (friendship) network and a bipartite user-item rating network. In contrast

to purely random (i.e., Erdos-Renyi) graphs, where the connections among nodes are completely independent random events, the synthetic model ensures dependency among the connections of nodes, by characterizing each node with a m-dimensional vector with each element a randomly selected real number in the interval [-1,1]. This vector represents the initial user profile used for the construction of the friendships and ratings profiles, which are generated as follows:

- For the construction of the friendship network, two nodes are considered to be similar and thus of high probability to connect to each other if they share many close attributes in their initial user profile. Given a network size N and a mean degree k of all nodes, the generator starts with an empty network with N nodes. At each time step, a node with the smallest degree is randomly selected (there is more than one node having the smallest degree). Among all other nodes whose degrees are smaller than k, this selected node will connect to the most similar node with probability $1 - p$, while a randomly chosen one with probability p. The parameter $p \in [0, 1]$ represents the strength of randomness in generating links, which can be understood as noise or irrationality that exists in almost every real system.

- For the construction of the user-item rating network, the generator follows a similar procedure. It uses the following additional parameters as well: (i) the ratings range, (ii) the mean number of rated items by all users. Notice that each user can rate different items from others and has in his profile a different number of rated items, following the power law distribution.

7.2.2 Generating Social Data with Concept Drift

Faloutsos et al. [Du et al., 2010] classify graph generators models into emergent and generative. In the emergent graph models, such as the small-world model [Du et al., 2010], the macro network properties emerge from the micro interactions of nodes over time. On the other hand, generative graph models facilitate a utility function that performs recursive iterations, until the generated networks meet real networks properties. Previous research works have studied the patterns that follow particular features of the network such as the link weights, in order to map the evolution of the network and build network generators. For example, Nan Du et al. [Du et al., 2009] have proposed a utility-driven generator that models the way in which humans decide when and whom to communicate with.

xSocial is a multi-modal graph generator that mimics real social networking sites in their way of producing simultaneously a network of friends and a network of their co-participation [Du et al., 2010]. In particular, xSocial consists of a network with N nodes, each of which has a preference value $calf_i$. At each time, every node performs three independent actions (write a message, add a friend and comment on a message). A node chooses his friends either by their popularity of by the number of messages on which they have commented together, which is determined by his preference $calf_i$. A node can also follow the updated status of his friends by putting comments on the corresponding new messages.

Fei Yan et al. [Yan et al., 2011] designed a graph generator that simulates how social links among actors of affiliation networks (e.g. research paper co-authorship) emerge, based on the events they co-participate in and the clique superposition evolution process. Their model generates undirected weighted time evolving graphs, using an actor-projection of the actor-event affiliation network. Specifically, the generator includes two basic graph evolution steps, node and edge evolution. At each time, a probability parameter p_{node} determines whether a node or an edge evolution step occurs. In a node evolution step, a newcomer node joins the graph. In an edge evolution step, a node is randomly selected as a newcomer and connects with all members of an event. If there are members of the event[1] who are already connected, then the weight increases.

The generator of Symeonidis et al., [Symeonidis et al., 2010] uses both structural (friendship network) & content-based information (item-rating network) for making recommendations to the users. However, it is only suitable for static learning. xSocial, on the other hand, simulates the stream-based problem but it only uses the structural information (i.e., networks of friends and their interactions) for making recommendations to the users. Different from [Symeonidis et al., 2010], our generator aims to alleviate the problem of static recommendations by making use of the dynamic ratings profile (i.e., a user's rating preferences may change overtime), and unlike xSocial [Du et al., 2010] its primary focus is on content-based recommendations.

7.2.3 Generating Arbitrary Streams with Concept Drift

The data generators of [Symeonidis et al., 2010; Du et al., 2010] are intended for specific applications, and simulate the properties of the data in these applications. Research on stream mining demands artificial datasets

[1] affiliated item

that exhibit particular properties of the data, such as overlapping concepts or correlations among the features, or particular forms of drift. In their seminal work on the evaluation of stream learning algorithms [Gama et al., 2009], Gama et al. bring forward several issues on how to design evaluation experiments, and propose evaluation metrics for stream learners. They make apparent that it is essential to control several properties of the stream, not least the moments and nature of drift. The objective of stream data generators is essentially to provide control over the stream properties in a transparent way that allows proper experiment design.

In [Bifet et al., 2010], Bifet et al. report on the MOA stream mining framework and discuss a number of data generators that have been contributed to MOA until the moment of writing (2010). These generators are for a single, conventional stream, and are not appropriate for learning a model on temporal objects that are referenced again and again by the stream.

The core idea of MOA in modeling concept drift is that "a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift" [Bifet et al., 2010] (section 3.1), whereupon a sigmoid function is used to define the probability of an instance belonging to the new concept after the drift. The idea of combining two distributions is very appealing, but rather restrictive for the simulation of evolving temporal objects, each of which is associated with its own substream of instances/observations.

Ikonomovska et al. study relational learning over multiple streams [Ikonomovska et al., 2011]. Their learning scenario is in accordance to our perennial objects paradigm, hence the method they use to generate synthetic data for evaluation purposes are also relevant here. In particular, one of their data generators simulates the problem of discovering "individuals that would be suitable targets for viral marketing" [Ikonomovska et al., 2011] (Section V, page 702) by generating three streams on posts, comments and friends of a user. The generator takes a predefined relational regression tree as input to create the artificial data. This is an appealing idea, but it is rather limited to the evaluation of algorithms that learn trees gradually, and does not give any controls on the forms of drift that may occur.

Closest to Multi-Gen is the generator of evolving high-dimensional trajectories used to evaluate the online clustering algorithm TRACER in [Krempl et al., 2011]. A "trajectory" reflects the movement of an object in a multi-dimensional feature space, and the objective of the algorithm TRACER is to find clusters of objects that move in a similar way. Despite the geospatial connotation of the term "trajectory" (e.g. migratory

birds that travel in flocks), the clustes found by TRACER consist of objects that *evolve* similarly: people whose preferences change in the same way as they grow older, patients with chronic diseases respond similarly to certain medications etc. One of the most appealing characteristics of TRACER is its parsimonious nature: it requires as little as two observations per perennial object to learn and adapt to concept drift. Its data generator, which we term *Parsimonious Generator* hereafter, takes as parameter the size of the feature space, the number of clusters K (which is fixed over time), the individuals to be traced, the total number of observations to be generated and several parameters for tuning the suddenness of concept shift, the state-to-signal noise etc [Krempl et al., 2011]. Quoting [Krempl et al., 2011] (section 4.1), the data generator "uses a mixture model with K components. For each component a multivariate Gaussian density function is used to generate observations. The component centres are themselves functions of time. For the initial state, the position, speed and acceleration as well as possible higher derivatives are generated at random. Subsequent states are calculates using a state-transition-matrix as described above, and adding random state-transition noise. Furthermore, sudden shift occurs at a given point in time, offsetting the cluster centres by a random vector."

A major advantage of the Parsimonious Generator [Krempl et al., 2011] is its ability to capture complex forms of drift with a relatively small number of parameters in a transparent way. A disadvantage is that the mixture model assigns objects to clusters probabilistically. This may be restrictive for the evaluation of crisp classifiers and clustering algorithms.

7.3 Multi-Gen for the Generation of Profiles and Transactions

Our generator takes as input the parameters depicted in Table 7.1, and described in sequel. It generates: item profiles and from them items; user profiles and from them users; and ratings of users for items at each timepoint. A user profile may live at most $\mathcal{L}_{\Updownarrow\dashv\S}$ timepoints before it mutates.

7.3.1 Generation of Item Profiles and of Items

Item profiles are described by v^i synthetic variables. Multi-Gen creates a set P^i with N^i item profiles and stores for each one the mean and variance of each of the v^i variables. Next, each of these item profiles is used as

Table 7.1: Parameters of Multi-Gen.

Param	Description
u	user
i	item
P^i	set of item profiles; $N^i = \|P^i\|$
P^u	set of user profiles; $N^u = \|P^u\|$
n^i	number of items per item profile
n^u	number of users per user profile
v^i	number of synthetic variables that describe an item profile
v^u	number of synthetic variables that describe a user profile
τ_d	number of drift moments across the time axis
A_d	set of active user profiles at drift moment d
$\mathcal{L}_d$	number of timepoints at drift moment d; upper boundary is $\mathcal{L}_{max}$
β_d^u	number of items a user u can rank at drift moment d
α_t^u	number of items a user u can rank at timepoint t
R	max. number of items rated by a user at any timepoint
s, v	parameters to control the shape of ZipF distribution for generating user ratings data in conjunction with R, which is the main parameter of the distribution
$\phi_{\mathcal{U}\to\mathcal{I}}$	the probability of a user profile $\mathcal{U}$ selecting an item from item profile $\mathcal{I}$ for rating.
$UP2IP$	This variable is used to control the assignment of degree of affinity between user profiles and item profiles
PKS	This variable is used to control the position of the item profiles.

prototype for the generation of n_i items, producing $n^i \times N^i$ items in total. Items also adhere to the v^i variables; the value of each variable in an item adhering to profile $\mathcal{I} \in P^i$ is determined by the mean and variance of this variable in the profile $\mathcal{I}$. An example of item profiles with $v^i = 4$ is depicted in Figure 7.1. To control the position of the item, variable PKS is used. When the value of PKS is close to 0, item profiles are generated randomly and when the value is close to 1, items profiles evenly get spread out to the extreme edges of the feature space. Two consecutive profiles are spread out to the opposite corners, e.g., assume a feature space with two dimension Dim_x and Dim_y. Both dimension can take values from 0 to 100. If 4 item profiles are to be generated, then profile 1 would be spread

towards the corner (0,0) and profile 2 towards the corner (100,100). Profile 3 would be spread towards (0,100) and profile 4 towards (100,0).

The number of items considered (rated) at some timepoint may vary from one timepoint to the next, but there is no bias towards items of some specific profile(s). Hence, item profiles are not exhibiting concept drift.

Item Profiles	Var 1	Var 2	Var 3	Var 4
IP1	23 ±4	52 ±9	97 ±2	8 ±9
IP2	2 ±9	1 ±7	46 ±3	91 ±6
IP3	72 ±7	71 ±3	26 ±2	52 ±1
IP4	3 ±4	2 ±4	27 ±5	25 ±3

Figure 7.1: Sample item profiles with $v^i = 4$ synthetic variables. The mean and variance associated with each variable are used to generate items according to the normal distribution.

7.3.2 Generation and Transition of User profiles

User profiles are described by a set of parameters v^u. These are synthetic variables. Multi-Gen creates a set P^u with N^u user profiles and stores for each one the mean and variance of each variable in v^u. User profiles serve as templates for the generation of users, in much the same way as item profiles are used to generate items. However, there are two main differences. First, user profiles are subject to transition, and not all of them are active at each drift level $d = 1, \ldots, \tau_d$. Second, a user profile exhibits affinity towards some item profiles, expressed through the probabilities between the item profile and the user profile. An example of user profiles with $v^u = 3$ is depicted in Figure 7.2.

The affinity of user profiles towards item profiles manifests itself in the user ratings: a generated user adheres to some user profile and rates items belonging to the item profile(s) preferred by her user profile. The affinity $\phi_{\mathcal{U}\to\mathcal{I}}$ is defined as the probability of a user profile $\mathcal{U}$ selecting an item from item profile $\mathcal{I}$ for rating. This probability is controlled by a user-defined global parameter $UP2IP \in [0, 1]$. If $UP2IP$ is close to zero, user profiles show strong affinity towards a certain item profile, while if the value is closer to 1, the probabilities are initialised randomly.

At each drift level $d = 1, \ldots, \tau_d$, only a subset of user profiles $A_d \subseteq P^u$ are active[2]. The active profiles at each drift level is determined at the

[2]the number of active profiles at each drift moment d is an integer and calculated using $\frac{N^u}{d}$ and is similar for each drift level

User Profiles	Var 1	Var 2	Var 3	Item Profile Probabilities			
				IP1	IP2	IP3	IP4
UP1	13 $\pm$0	22 $\pm$5	51 $\pm$1	0.1	0.6	0.25	0.05
				20 $\pm$5	50 $\pm$8	80 $\pm$9	29 $\pm$1
				IP1	IP2	IP3	IP4
UP2	34 $\pm$4	55 $\pm$0	68 $\pm$9	0.7	0.1	0.1	0.1
				90 $\pm$2	25 $\pm$5	93 $\pm$4	9 $\pm$1
				IP1	IP2	IP3	IP4
UP3	21 $\pm$5	98 $\pm$4	1 $\pm$5	0.2	0.5	0.1	0.2
				42 $\pm$2	95 $\pm$2	10 $\pm$0	12 $\pm$5

Figure 7.2: Sample user profiles with mean and variance for synthetic variables with probabilities of selecting an item from a certain item profile (row 1) and mean and variance of the rating that item (row 2), where $v^u = 3$.

beginning. For drift moment $d > 1$, Multi-Gen maps the profiles A_{d-1} of moment $d-1$, to the new profiles A_d of moment d on similarity, i.e. the transition probability from an old to a new profile is a function of the similarity between the two profiles. The result is a *profile transition graph*, an example of which is depicted in Figure 7.3.

The coupling of profile transition to the profile similarity function ensures that profile mutation corresponds to a gradual drift rather than an abrupt shift. The extent of profile mutation is further controlled by a user-defined global variable $\in [0, 1]$ that determines the *true preference* of an old user profile for a new user profile. A value close to zero means that the most similar new profile will always be preferred. Larger values allow for a weaker preferential attachment, while a value close to 1 means that the new profile is chosen randomly, and the transition is essentially a concept shift rather than a drift (a "drift" is a change of gradual nature, e.g. when a profile mutates into a similar one; while a "shift" is a more drastic and abrupt change, e.g. a profile being *replaced* by another one). The similarity between two user profiles $\mathcal{U}$ and $\mathcal{U}'$ is defined in Equation 7.1.

$$sim(\mathcal{U},\mathcal{U}') = \sqrt{\sum_{\mathcal{I} \in P^i} (\phi_{\mathcal{U}\to\mathcal{I}} - \phi_{\mathcal{U}'\to\mathcal{I}})} \tag{7.1}$$

where $\phi_{\mathcal{U}\to\mathcal{I}}$ is the probability of rating an item from profile $\mathcal{I}$ for $\mathcal{U}, \mathcal{U} \in A_{d-1}$ and $\mathcal{U}' \in A_d$.

Affinity of user profiles towards each other is also affected by profile transitions. Once a user profile $\mathcal{U}$ mutates to $\mathcal{U}'$, all its users adhere to the $\mathcal{U}'$ profile: they prefer the item profiles to which $\mathcal{U}'$ shows affinity, and rate items adhering to these item profiles.

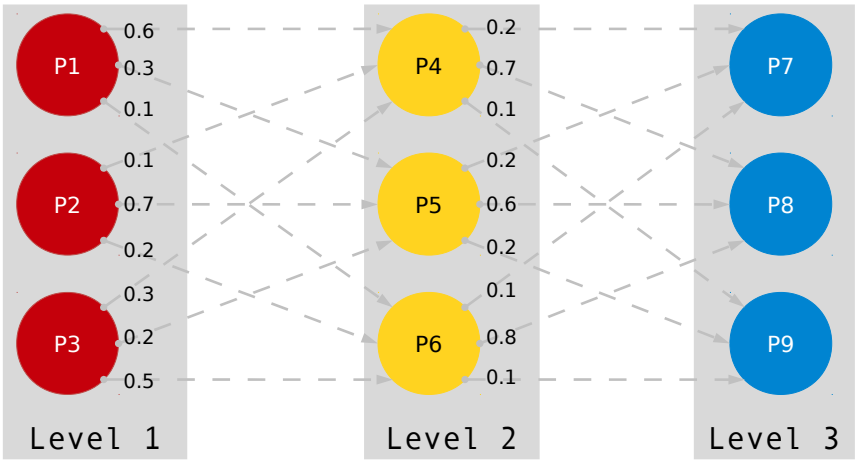

Figure 7.3: Graph for the transition of user profiles; each column corresponds to a timepoint, indicating that the number of profiles/classes may change from one timepoint to the next (transition probabilities between level 2 and level 3 have been omitted)

7.3.3 Generation of Users

For each user profile $\mathcal{U} \in P^u$, Multi-Gen creates n^u users. As for items, users adhere to the set of parameters v^u as user profiles; the value of each parameter in a user adhering to profile $\mathcal{U}$ is determined by the mean and variance of this variable in the profile $\mathcal{U}$. The profiles of each drift level d exists for at most $\mathcal{L}_{max}$ timepoints, before profile transition occurs; the lifetime of a profile is equal to the number of timepoints $\mathcal{L}_d \leq \mathcal{L}_{max}$ at drift moment d.

Generation of Ratings

For each timepoint, Multi-Gen creates ratings for all users in each active profile. At each timepoint a user can rank β_d^u items per drift moment. Then, for each user profile $\mathcal{U}$ and user u adhering to $\mathcal{U}$, an item i adhering to $\mathcal{I}$ is randomly chosen based on the probability $\phi_{\mathcal{U}\to\mathcal{I}}$. A rating value for item i is generated based on the normal distribution with mean and variance parameter from the user profile $\mathcal{U}$ for the item profile $\mathcal{I}$ (see Figure 7.2).

The number of rated items by a user follows a power-law [3] distribution,

[3] discrete Zipf

generated using the rejection-inversion method described in [Hörmann and Derflinger, 1996]. Using this approach to the Zipf distribution, we implemented an algorithm that is at least twice as fast as the standard methods, following the technical details of [Hörmann and Derflinger, 1996].

The Zipf distribution (also called Riemann zeta or Pareto distribution) has been selected, as it frequently appears in real data (ratings, linguistics, social events, etc.) [Johnson et al., 2005; Dagpunar, 1988], and also is used in simulators.

We implemented the two parameter generalization of the discrete Zipf distribution as defined in [Dagpunar, 1988] with unnormalized pdf[4]:

$$p_k = \frac{1}{(v+k)^s} \qquad (k = 0, 1, ...)$$

where $s > 1$ and $v > 0$ are the two parameters of the distribution. k parameter is synonymous with parameter R. The mono-parametric case is derived with a default value of $v = 1$:

$$p_k = \frac{1}{(1+k)^s} \qquad (k = 0, 1, ...)$$

7.3.4 Algorithm: Multi-Gen

The complete algorithm for generating a perennial stream of ratings is given in Algorithm 15. The algorithm starts by creating a set of user profiles P^u and items profiles P^i (Lines 1-2). It then computes the set active user profiles A_d for each drift moment d (Line 3). The computation involves distributing the user profiles equally across all drift moments. Using A_d, the algorithms computes the profiles transition graph using the active profiles from consecutive drift moments, i.e., between A_{d-1} and A_d (Line 4). The creation of n^u users and n^i items, concludes the initialisation of Multi-Gen (Lines 5-6).

The algorithm, then iterates over the defined drift moments. At the beginning, a user u is assigned a random user profile $\mathcal{U}$. For the first drift moment, it uses $\mathcal{U}$ to generate the data. For the subsequent drift moments, when the drift moment changes, the $\mathcal{U}$ gets mutated into $\mathcal{U}'$ by the profile transition graph (Lines 10-11) and the mutated profile $\mathcal{U}'$ is used for generating further data. Then for each user, the power law distribution is used

[4]probability function

to generate ratings at a drift moment (Line 13).

Algorithm 15: Multi-Gen

Input : $N^i, N^u, n^i, n^u, v^i, v^u, \tau_d, \mathcal{L}, R,$
Output: $P^i, P^u,$ profile transition graph and perennial stream of ratings

```
                                                    /* INITIALISE GENERATOR */
1  Generate a set of item profiles P^i with v^i number of attributes.
2  Generate a set of user profiles P^u with v^u number of attributes.
3  Select the active profiles A_d for each drift moment d.
4  Compute profile transition graph using A_d.
5  Generate n^i items per item profile in P^i.
6  Generate n^u users per user profile in P^u.
                                                    /* DATA GENERATION */
7  for d = 1 → τ_d do
8      foreach user u do
9          U ← profile of user u
10         if d > 1 then
11             Mutate user profile U using profile transition graph.
12         Generate data for u using user profile U
13         β_d^u ← PowerLaw(R, s_R)

14     L_d ← Uniform(0,L)   /* #timepoints for drift moment d */
15     for t = 1 → L_d do
16         foreach user u do
17             U ← profile of user u
18             α_t^u ← Uniform(0,β_d^u)
19             β_d^u ← β_d^u − α_t^u
20             for j = 1 → α_t^u do
21                 Select an item based on φ_{U→I} & rate it wrt. to U.
```

A drift moment d contains multiple timepoints (Line 14) and the ratings are generated at individual time points. The number of ratings to be generated for a user u at timepoint t, α_d^u, is calculated using the uniform distribution, whose lower limit is 0 while upper limit is β_d^u (Line 18). Once α_d^u is calculated, the same amount is also subtracted from β_d^u to ensure that u's rating don't exceed more than β_d^u items (Line 19)[5]. The individual rat-

[5]For the last timepoint t' of drift moment d, $\alpha_{t'}^u \leftarrow \beta_d^u$. This ensures that there are outstanding items to be rated.

ings are generated using the specifications based on the affinities of user profile $\mathcal{U}$ towards the different item profiles.

The list of Multi-Gen components are as follows:

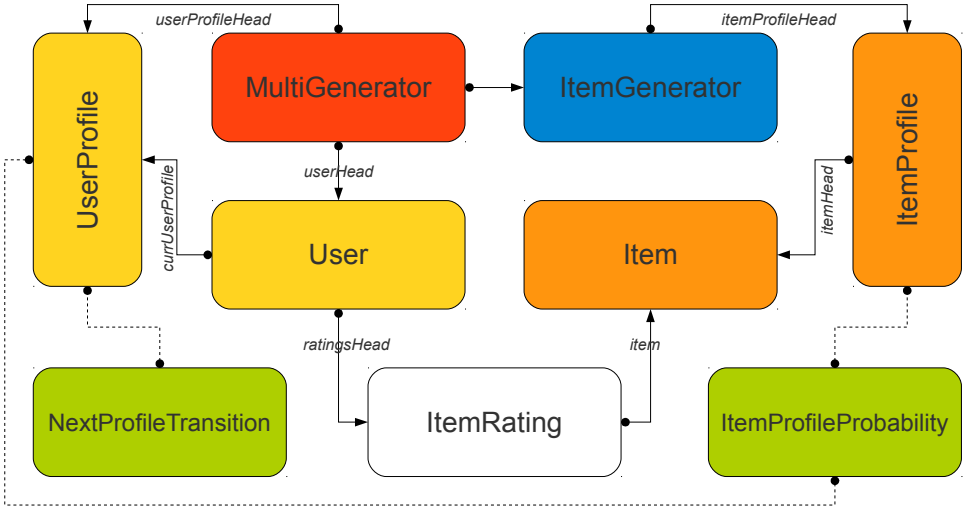

Figure 7.4: Components of Multi-Gen.

- **MultiGenerator** This is the main component of the generator. Manages users, userprofiles and also manages items and their profiles using the ItemsGenerator.
- **ItemsGenerator** Generates items profiles, which are then used to create the items.
- **ItemProfile** & **Item** Item profile stores the prototypes for creating items, i.e., mean and variance. While items only contain the generated vals.
- **UserProfile** & **Users** The main aspect of these are similar to ItemProfile and Item.
- **ItemRating** Is part of the user object and contains information about how user rated a particular item.
- **NextProfileTransition** Additionally, each profile also holds the information about possible transition to the profile in the next level. As

already discussed in Section 7.3.2, each transition has a probability being made. This transition information managed through NextProfileTransition. Each user profile maintains a list of next level profile usint NextProfileTransition and selects one of them whem prompted.

- **ItemProfileProbability** Each user profile also maintains the information about the groups of item the user is associated with and how he rates them. As items are generated using profiles, we also use these profiles to divide items into groups. Each group/profile is assigned a certain probability of being rated. During user rating process, a random item is picked from the group, its rating is acquired and is then associated with the user.

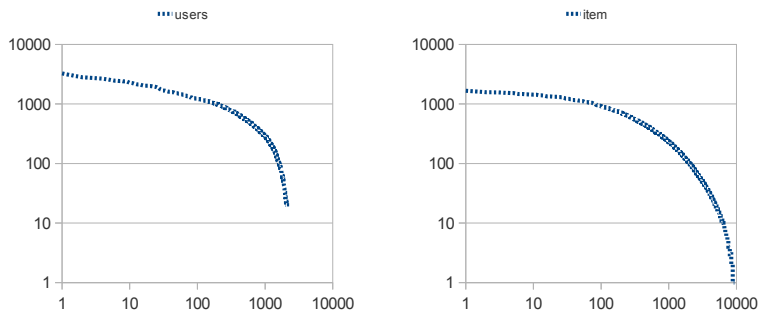

Figure 7.5: Statistics of the MovieLens dataset. On the left, we see the number ratings given by each user (x-axis) and on the right the number ratings given for each item (x-axis), both sorted in descending values of the number of ratings and plotted on a logarithmic scale.

7.4 Experiments with Multi-Gen

In this section, we show that Multi-Gen is capable of simulating the properties of a real world dataset, namely the dataset **hetrec2011-movielens-2k** described below. In this section we provide the empirical analysis of our generator[6]. In the experiments, we first vary the main parameter s of the distribution (skewness parameter) by keeping $v = 1$ (cf. Table 7.1) and we vary the parameter v and the maximum number items rated by a user, by keeping the value of s equal to 3.

[6]The data generator along with further results can be downloaded from *http://omen.cs.uni-magdeburg.de/itikmd/mitarbeiter/zaigham-faraz-siddiqui.html*

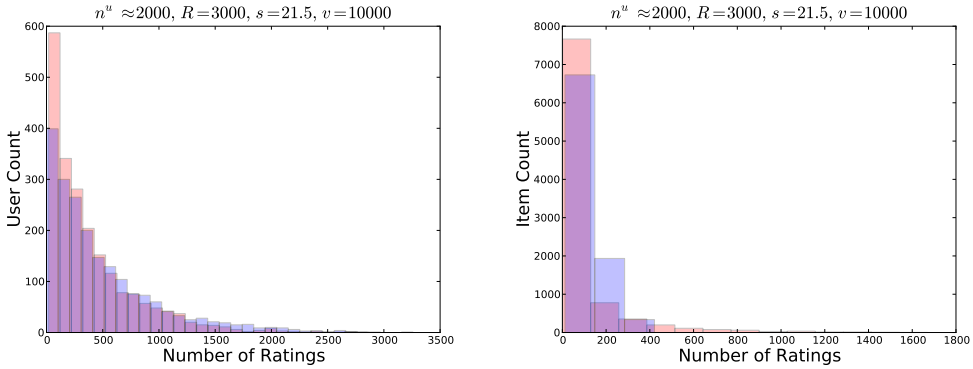

Figure 7.6: Comparison of the dataset from Multi-Gen and **hetrec2011-movielens-2k** dataset. (Left) distribution of the number of ratings provided the user (right) number of ratings an item has

7.4.1 Quantitative Results

Datasets from the *MovieLens* are widely used for testing of the algorithms. From the various datasets available at the GroupLens web site[7], we chose **hetrec2011-movielens-2k**. It has 2113 users, approximately 10,000 movies and approximately 0.85 million ratings provided by the users. Additionally with the dataset the information about each movie, i.e., genre, cast, location is also provided. We treat this as our baseline and compare the results of Multi-Gen against it.

In Figure 7.5 we plot the distribution of user (left) and item ratings (right) from the *MovieLens* dataset. We see that the distributions are sim-

[7]http://www.grouplens.org/node/73

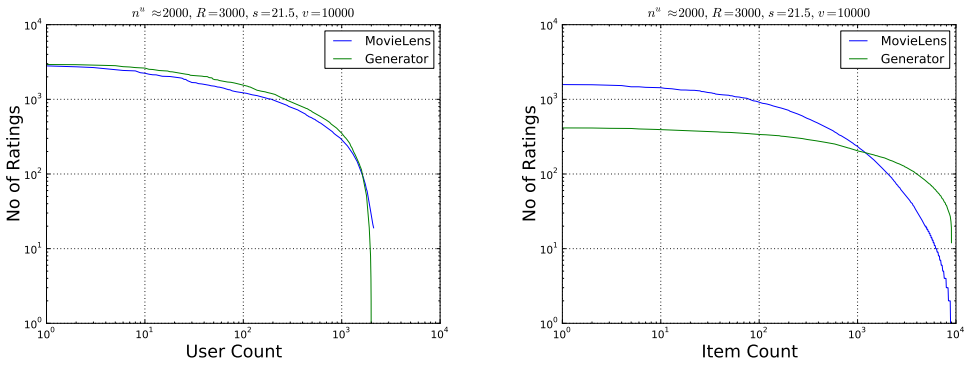

Figure 7.7: Log-Log plot from the Figure 7.6

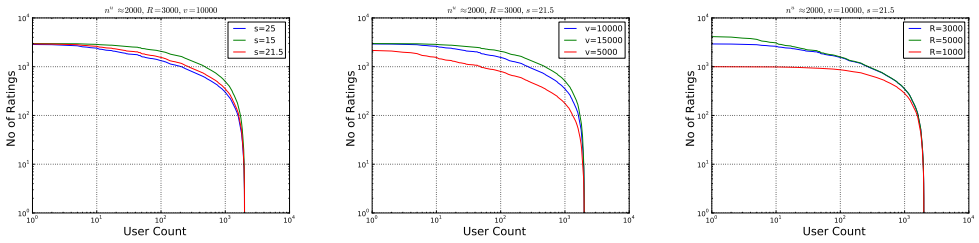

Figure 7.8: Effect of changing various parameters of the Zipf distribution on the user's rating behaviour (left) variable s, (middle) variable v and (right) variable R

ilar, but the left one ends abruptly. This is because the data provider has removed all the users who have less than 20 ratings. This made it slightly difficult to estimate the correct parameter for the Zipf distribution.

7.4.2 Validation Measures

Common measures used for parameter calibrations and validation are the *Mean Absolute Error* (MAE) and the *Root Mean Square Deviation* (RMSD). Especially $RMSD$ is a good measure of the accuracy [Wikipedia, 2012a,b].

For two separate samples, both of cardinality N: $A = \{a_1, ..., a_N\}, B = \{b_1, ..., b_N\}$, MAE and RMSD are defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|a_i - b_i|$$

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - b_i)^2}$$

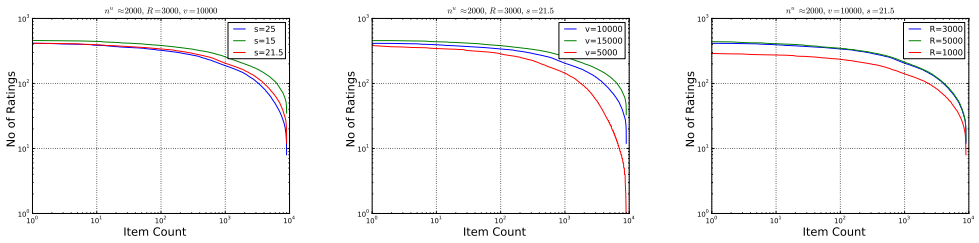

Figure 7.9: Effect of changing various parameters of the Zipf distribution on the how items get rated (left) variable s, (middle) variable v and (right) variable R

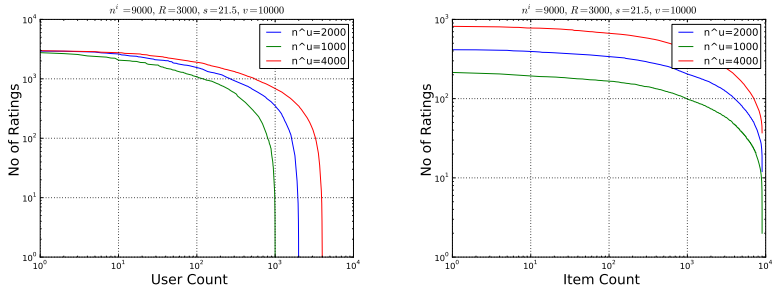

Figure 7.10: Effect of changing the parameter n^u on the no of ratings distribution for user and item

We have $N = 2113$ users, a_i is the number of ratings in **hetrec2011-movielens-2k** for the user i, ($i = 1, ..., n$), and b_i are the corresponding number of ratings for the user i in the generated dataset.

7.4.3 Results

After a lot of experimentation and calculations we found that the best fit holds when $s = 21.5$, $v = 10000$, $R = 3000$. In the Figure 7.6 we compare the distributions from the MovieLens dataset and the dataset generated by Multi-GenṪhe discovered parameter settings minimize both errors: $MAE = 105.13$, $RMSD = 142.57$. However, these parameter settings can still be further improved as seen in the figure, especially in terms of which items get rated more and how much (see left of Figure 7.6). The skewed effect on the items rating is more clearly visible in Figure 7.7. Currently, we use random variable for generating the number of ratings by a customer only and the same random number is used implicitly for selecting an item. For this reason, it is easier to predict the effect of parametric changes on the number of ratings by the customers but not on the items.

In Figures 7.8-7.11 we show the effects of parametric change on how the ratings for the items get affected.

7.5 Conclusion

In this chapter we presented a multi-stream generator that has been inspired from the domain of recommendation system. It generates ratings data for users according to user profiles. With time the profiles mutates into newer ones. The mutation can be adjusted to simulate drastic shifts

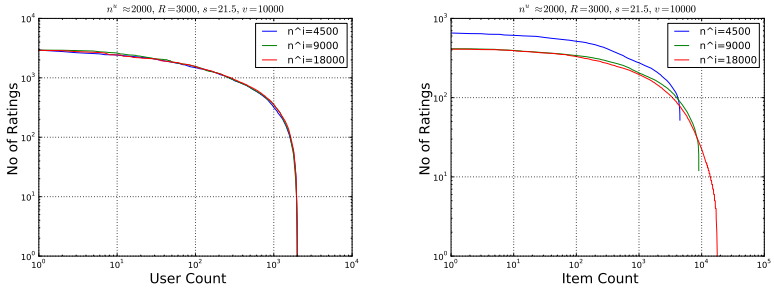

Figure 7.11: Effect of changing the parameter n^i on the no of ratings distribution for user and item

as well more gradual drifts. The generator can be used for evaluating supervised and unsupervised learning task for discovering and adaptation to concept drift.

We allow the user profiles to mutate over time, however, the items remain static. In real world the item profiles under go change as well, e.g., comedy movies have gradually gotten more and more anarchic or with the advances in technology the sci-fi genre have gradually changed its outlook. While the mutation of user profiles represents sudden shift in the distribution of the data, by incorporating the dynamics of change for item profiles, more gradual shift can be incorporated into the dataset. Such a generator would also be beneficial trajectory-based stream methods. Another change that we plan to work on is to use an independent random variable for the items rating to make the generator more closer to the real world phenomenon.

CHAPTER 8

Conclusion and Future Work

In this chapter, we first present concise summary of the work that was undertaken in this thesis. Here we also mention some of the important findings and then discuss possible directions for future research.

8.1 Summary

In this thesis, we have introduced the problem of mining over a stream of perennial objects. Perennial objects come from a multiple interrelated streams. The stream that is the focus of the learning task, is designated as the target stream. This stream consists of perennial objects and is fed from other streams, which may contain perennial or ephemeral objects. The properties of perennial objects pose several challenges for the mining algorithms which are:

1. perennial objects constitute a stream,
2. are multi-relational objects that are linked to objects from other streams,
3. they may not be forgotten,
4. may evolve over time, and
5. may change their class labels.

In order to perform mining over a stream of perennial objects, an algorithm is required to have the following properties: 1) the capability to handle streaming multi-relational data, 2) a mechanism that can recall old objects instead of discarding them forever, 3) the capability to handle dynamic nature of perennial objects, and 4) the capability to update the classifier when perennial objects change their labels.

Our solution for mining over a stream of perennial objects is made up of several components. In order to process the multi-relational structure of

perennial objects and learn models, we developed an incremental propositionalisation algorithm[1] that transforms multi-relational perennial objects into propositional objects. The transformation process operates over the multi-relational stream. For each stream that is in 1-to-1 or 1-to-m relationship with the target stream of perennials, it is concatenated to the target stream. For a stream that is in m-to-1 relationship, the objects in the stream are first summarised using simple aggregates and then concatenated to the respective perennial objects.

To maintain short-lived ephemeral objects inside limited memory, we use a sliding window to store only the most recent ones. For maintaining perennial objects that may not be forgotten, we use a cache to store most important ones, while remaining ones are kept inside a secondary storage and can be recalled when needed. In the experiments, we observed that the storage of only the most important objects in the caches, facilitates the creation of better and robust models.

We also developed an incremental K-Means algorithm to learn groups of homogeneous perennial objects. The algorithm incorporates procedures to handle the evolving nature of perennial objects[2] and can adapt the clustering model.

Another implication of the evolving nature of perennial objects is that they can change their class label. For example, a customer who were earlier trustworthy, may evolve to become an untrustworthy customer. For learning a classification model over a labelled stream of perennial we developed a tree induction algorithm[3]. The tree-based model is learned incrementally. As perennial objects evolve, the algorithm forgets their earlier contribution and updates the model with their new state. For model adaptation, the algorithm maintains alternate trees and chooses among them on the grounds of their *accuracy*, *support* and *age*.

The propositionalisation algorithm uses simple aggregates that capture only simple patterns from the neighbouring streams. This leads to creation of tress that are large and unstable. We enrich the process of propositionalisation, for labelled perennial objects, by using classification rules to capture complex patterns. For this purpose, we developed a classification rule mining algorithm for ephemeral streams. The rules are stored in compressed form and the predictive ones are chosen to be used as attributes during propositionalisation. The experiments showed that the usage of rule-based attribute leads to smaller trees and robust performance.

[1] based on RelAggs by Krögel [Kroegel, 2003]

[2] We haven't formalised the notion of age and size in the context of evolution for a perennial object. We will elaborate on it in the outlook section (cf. Section 8.2)

[3] based on CVFDT by Hulten et al. [Hulten et al., 2001]

We also presented a multi-stream generator. We have used the generator to create synthetic datasets that we have used in this thesis. It has been inspired from the domain of recommendation system and generates ratings data for users according to user profiles. With time the user profiles mutates or undergo concept drift, which changes the behaviour of the users that adhere to them. This drift imputation is adjustable and can be used to simulate from sudden shifts to more gradual drifts.

8.2 Directions for Future Research

Mining over a stream of perennial objects is a new problem. We have presented the first solutions in this regard, which expands the existing stream mining paradigm. However, this new problem raises many unanswered questions.

Propositionalisation alleviates some of the limitations of relational methods, but introduces several of its own. One of them is its assumption about the attributes' independence [4]. Several solutions for the static case have been proposed in this regard but they either generate far too many attributes [Alfred, 2008a] or are limited by high computational cost [Anderson and Pfahringer, 2009]. Our solution for handling this particular problem (cf. Chapter 6) maintains a concise representation of the discovered rules but still suffers from the generation of enormous amount of *redundant*. With a wealth of stream clustering methods in the literature, one possible direction is the usage of clusters for pattern-based aggregation, for they neither suffer from explosive number of patterns nor redundancy. The development of such a framework is one of the topic of our undergoing research.

In this thesis, we have focussed more on the perennial objects and how to store them efficiently. However, the main load in the relational stream comes from the streams of ephemeral objects. These streams may under go change in speed [5], the rate at which the concept changes may vary[Bifet and Gavaldà, 2007]. This asks for either a size reassignment as proposed in [Ikonomovska et al., 2011] or incorporation of a mechanism to discard useless or redundant tuples. A possible solution in this regard may be the use of sampling to reduce the load. However, sampling in a relational stream, where objects are inter-linked is a difficult problem on its own and usage of caches from which objects move in and out makes it even more challenging. This is also a topic of our on going research.

[4] We have formulated this problem in Chapter 6

[5] frequency of objects arriving at each timepoint

During the experiments we encountered perennial objects that were of varying sizes (i.e., the amount of objects that feed them) and ages (i.e., the amount of time for which they have been active). In financial dataset, we experienced this phenomenon: new customers arrive regularly and the difference in cumulative number of transactions done by the older and the newer ones is big. Distance functions simply return a value based on the current states of the objects to be compared and overlook their ages, sizes and the evolution that these objects may have gone through overtime. Solutions to these (i.e., implications introduced by ages and sizes) would not only result in better models but they may be helpful in a situation, where prediction of next state of an object is required, e.g., in bankruptcy prediction or prediction of some disease outbreak. We have already developed two preliminary frameworks in this regard; one uses Kalman filters [Krempl et al., 2011], while the other uses a combination of clustering and a *mixture* of Markov Chains [Siddiqui et al., 2012] for tracking objects and predicting their next states. This problem setting can also be adopted for the supervised case, where the true labels for the objects arrive with a significant delay. For example, the financial standing of a customer or a company is evaluated yearly. Prediction can be used in predicting the true label for such an object as data arrives for it in the other streams.

Bibliography

Adomavicius, G. and A. Tuzhilin (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering 17*(6), 734–749.

Aggarwal, C., J. Han, J. Wang, and P. Yu (2003). A framework for clustering evolving data streams. In J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer (Eds.), *Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003*. VLDB Endowment.

Agrawal, R. and R. Srikant (1994). Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo (Eds.), *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1994*, San Francisco, CA, USA, pp. 487–499. Morgan Kaufmann Publishers Inc.

Alfred, R. (2008a). Dara: Data summarisation with feature construction. In *Proceedings of the Asia International Conference on Modelling and Simulation,*, AICMS 2008, pp. 830–835. IEEE Computer Society.

Alfred, R. (2008b). Dynamic aggregation of relational attributes based on feature construction. In P. Atzeni, A. Caplinskas, and H. Jaakkola (Eds.), *Advances in Databases and Information Systems, 12th East European Conference, ADBIS 2008, Proceedings*, Volume 5207 of *Lecture Notes in Computer Science*, pp. 2–13. Springer-Verlag.

Alfred, R. (2009). Discovering knowledge from multi-relational data based on information retrieval theory. In R. Huang, Q. Yang, J. Pei, J. Gama, X. Meng, and X. Li (Eds.), *Proceedings of the 5th International Conference Advanced Data Mining and Applications*, Volume 5678 of *ADMA 2009*, pp. 409–416. Springer.

Alfred, R. (2011). Feature transformation: A genetic-based feature construction method for data summarization. *Computational Intelligence 27*(3), 315–335.

Alfred, R. and D. Kazakov (2007). Discretization numbers for multiple-instances problem in relational database. In Y. E. Ioannidis, B. Novikov,

and B. Rachev (Eds.), *Advances in Databases and Information Systems, 11th East European Conference, ADBIS 2007, Proceedings*, Volume 4690 of *Lecture Notes in Computer Science*, pp. 55–65. Springer.

Anderson, G. and B. Pfahringer (2007). Clustering relational data based on randomized propositionalization. In H. Blockeel, J. Ramon, J. W. Shavlik, and P. Tadepalli (Eds.), *Proceedings of the International Conference on Inductive Logic Programming*, Volume 4894 of *ILP 2007*, pp. 39–48. Springer.

Anderson, G. and B. Pfahringer (2008). Exploiting propositionalization based on random relational rules for semi-supervised learning. In T. Washio, E. Suzuki, K. M. Ting, and A. Inokuchi (Eds.), *Proceedings of the 12th European Conference on Principles of Data Mining and Knowledge Discovery, PAKDD 2008*, Volume 5012 of *Lecture Notes in Computer Science*, pp. 494–502. Springer-Verlag.

Anderson, G. and B. Pfahringer (2009). Relational random forests based on random relational rules. In C. Boutilier (Ed.), *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI 2009, pp. 986–991.

Aydin, T. and H. A. Güvenir (2004). Learning interestingness of streaming classification rules. ISCIS 2004, pp. 62–71. Springer-Verlag.

Aydin, T. and H. A. Güvenir (2005). Modeling interestingness of streaming classification rules as a classification problem. In F. A. Savaci (Ed.), *Artificial Intelligence and Neural Networks, 14th Turkish Symposium, TAINN 2005. Revised Selected Papers*, Volume 3949 of *Lecture Notes in Computer Science*, pp. 168–176. Springer.

Babcock, B., M. Datar, and R. Motwani (2002). Sampling from a moving window over streaming data. In *Proceedings of the 13th Annual ACM SIAM Symposium on Discrete Algorithms, SODA 2002*, Philadelphia, PA, USA, pp. 633–634. ACM Press.

Beringer, J. and E. Huellermeier (2006). Online clustering of parallel data streams. *Data and Knowledge Engineering 58*(2), 180–204.

Bifet, A. and R. Gavaldà (2007). Learning from time-changing data with adaptive windowing. In *Proceeding of the SIAM International Conference on Data Mining, SDM 2007*. ACM Press.

Bifet, A., G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009, pp. 139–148. ACM Press.

Bifet, A., G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl (2010, September). MOA: massive online analysis, a framework for stream classification and clustering. In *Proceedings of International Workshop on 'Handling Concept Drift in Adaptive Information Systems' at ECML PKDD 2010*, HACDAIS 2010, Barcelona, Spain.

Blockeel, H. and L. D. Raedt (1998). Top-down induction of first-order logical decision trees. *Journal of Artificial Intelligence 101*(1-2), 285–297.

Bloom, B. H. (1970, July). Space/time trade-offs in hash coding with allowable errors. *ACM Communications 13*(7), 422–426.

Boettcher, M., M. Spott, and R. Kruse (2009). A condensed representation of itemsets for analyzing their evolution over time. In W. L. Buntine, M. Grobelnik, D. Mladenic, and J. Shawe-Taylor (Eds.), *Proceedings of the European Conference on Machine Learning and Principles of Knowledge Discovery, ECML PKDD 2009*, Volume 5781 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, pp. 163–178. Springer-Verlag.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition 30*(7), 1145–1159.

Bradley, P. S., U. M. Fayyad, and C. Reina (1998). Scaling clustering algorithms to large databases. In R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro (Eds.), *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge discovery and data mining, KDD 1998*, pp. 9–15. ACM Press.

Breiman, L. (2001, October). Random forests. *Machine Learning 45*(1), 5–32.

Cao, F., M. Ester, W. Qian, and A. Zhou (2006). Density-based clustering over an evolving data stream with noise. In *Proceeding of the SIAM International Conference on Data MiningSDM 2006*, pp. 328–339. ACM Press.

Catlett, J. (1991). *Megainduction: Machine Learning on Very Large Databases*. Ph. D. thesis, University of Sydney, Sydney, Australia.

Ceci, M., A. Appice, C. Loglisci, C. Caruso, F. Fumarola, C. Valente, and D. Malerba (2009). Relational frequent patterns mining for novelty detection from data streams. In P. Perner (Ed.), *Machine Learning and Data*

Mining in Pattern Recognition, 6th International Conference, MLDM 2009. Proceedings, Volume 5632 of *MLDM 2009*, pp. 427–439. Springer.

Chaudhuri, S., R. Motwani, and V. Narasayya (1999). On random sampling over joins. In *Proceedings of the 1999 ACM International Conference on Management of Data, SIGMOD 1999*, New York, NY, USA, pp. 263–274. ACM Press.

Cheung, D. W.-L., J. Han, V. Ng, and C. Y. Wong (1996). Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the 12th IEEE International Conference on Data Engineering*, ICDE 1996, Washington, DC, USA, pp. 106–114. IEEE Computer Society.

Cheung, D. W.-L., S. D. Lee, and B. Kao (1997). A general incremental technique for maintaining discovered association rules. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications*, DASFAA 1997, pp. 185–194. World Scientific Press.

Chi, Y., H. Wang, P. S. Yu, and R. Muntz (2004). Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Proceedings of the 4th IEEE International Conference on Data Mining*, ICDM 2004. IEEE Computer Society.

Chi, Y., H. Wang, P. S. Yu, and R. R. Muntz (2006, October). Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowledge Information System 10*(3), 265–294.

Chuang, K.-T., H.-L. Chen, and M.-S. Chen (2009, January). Feature-preserved sampling over streaming data. *ACM Transactions on Knowledge Discovery from Data 2*(4), 15:1–15:45.

Csernel, B., F. Clerot, and G. Hbrail (2007). Summarizing a 3 way relational data stream. In *Workshop on Data Stream Analysis*, WDSA 2007. Wikipedia.

Dagpunar, J. (1988). *Principles of random variate generation*. Oxford Science Publications. Clarendon Press.

Dai, B.-R., J.-W. Huang, M.-Y. Yeh, and M.-S. Chen (2006). Adaptive clustering for multiple evolving streams. *IEEE Transactions on Knowledge and Data Engineering 18*(9), 1166–1180.

Das, A., J. Gehrke, and M. Riedewald (2003). Approximate join processing over data streams. In *Proceedings of the 2003 ACM International Conference on Management of Data, SIGMOD 2003*. ACM Press.

Dehaspe, L. and H. Toivonen (1999). Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery 3*(1), 7–36.

Dehaspe, L. and H. Toivonen (2001). Discovery of relational association rules. In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, pp. 189–212. Springer-Verlag.

Domingos, P. and G. Hulten (2000). Mining high-speed data streams. In R. Ramakrishnan, S. J. Stolfo, R. J. Bayardo, and I. Parsa (Eds.), *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD 2000, New York, New York, USA, pp. 71–80. ACM Press.

Du, N., C. Faloutsos, B. Wang, and L. Akoglu (2009). Large human communication networks: patterns and a utility-driven generator. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009*, pp. 269–278. ACM Press.

Du, N., H. Wang, and C. Faloutsos (2010). Analysis of large multi-modal social networks: Patterns and a generator. In J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag (Eds.), *Proceedings of the European Conference on Machine Learning and Principles of Knowledge Discovery, ECML PKDD 2010*, Volume 6321 of *Lecture Notes in Computer Science*, Barcelona, Spain, pp. 393–408. Springer-Verlag.

Dzeroski, S. (2003, July). Multi-relational data mining: an introduction. *SIGKDD Explorer Newsletter 5*(1), 1–16.

Dzeroski, S. and N. Lavrač (1993). Inductive learning in deductive databases. *IEEE Transactions on Knowledge and Data Engineering 05*(6), 939–949.

Emde, W. and D. Wettschereck (1996). Relational instance based learning. In L. Saitta (Ed.), *Proceedings of the 13th International Conference on Machine Learning*, ICML 1996, pp. 122 – 130. Morgan Kaufmann Publishers Inc.

Ester, M., H. peter Kriegel, S. Jörg, and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In

Proceedings of the 13th National Conference on Artificial Intelligence, AAAI 1996, pp. 226–231. AAAI Press.

Farnstrom, F., J. Lewis, and C. Elkan (2000). Scalability for clustering algorithms revisited. *SIGKDD Explorer Newsletter 2*, 51–57.

Ferrer-Troyano, F. J., J. S. Aguilar-Ruiz, and J. C. R. Santos (2005). Incremental rule learning and border examples selection from numerical data streams. *Journal UCS 11*(8), 1426–1439.

Fumarola, F., A. Ciampi, A. Appice, and D. Malerba (2009). A sliding window algorithm for relational frequent patterns mining from data streams. In *Proceedings of the 12th International Conference on Discovery Science, DS 2009*, pp. 385–392. Springer-Verlag.

Gama, J. and P. Kosina (2011). Learning decision rules from data streams. In T. Walsh (Ed.), *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI 2011, pp. 1255–1260. IJCAI/AAAI.

Gama, J., P. Medas, G. Castillo, and P. P. Rodrigues (2004). Learning with drift detection. In A. L. C. Bazzan and S. Labidi (Eds.), *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, Proceedings*, Volume 3171 of *Lecture Notes in Computer Science*, pp. 286–295. Springer-Verlag.

Gama, J., P. Medas, and R. Rocha (2004). Forest trees for on-line data. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, New York, NY, USA, pp. 632–636. ACM.

Gama, J., R. Rocha, and P. Medas (2003). Accurate decision trees for mining high-speed data streams. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos (Eds.), *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2003, New York, NY, USA, pp. 523–528. ACM Press.

Gama, J., R. Sebastião, and P. P. Rodrigues (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009*, pp. 329–338. ACM Press.

Gratch, J. (1996). Sequential inductive learning. In *Proceedings of the 13th National Conference on Artificial Intelligence*, AAAI 1996, pp. 779–786. AAAI Press.

Guha, S., A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan (2003). Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering 15*(3), 515–528.

Gupta, A., V. Bhatnagar, and N. Kumar (2005). Incremental classification rules based on association rules using formal concept analysis. In *MLDMPR*. Springer-Verlag.

Hidber, C. (1999). Online association rule mining. In *Proceedings of the 1999 ACM International Conference on Management of Data, SIGMOD 1999*, New York, NY, USA, pp. 145–156. ACM Press.

Holmes, G., R. Kirkby, and B. Pfahringer (2005). Tie breaking in hoeffding trees. In *Proceedings of the Second International Workshop on Knowledge Discovery from Data Streams, Porto, Portugal, 2005*.

Hörmann, W. and G. Derflinger (1996, July). Rejection-inversion to generate variates from monotone discrete distributions. *Transactions on Modeling and Computer Simulation 6*, 169–184.

Hulten, G., L. Spencer, and P. Domingos (2001). Mining time-changing data streams. In D. Lee, M. Schkolnick, F. J. Provost, and R. Srikant (Eds.), *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge discovery and data mining*, KDD 2001. ACM Press.

Ikonomovska, E., K. Driessens, S. Dzeroski, and J. Gama (2011, December). Adaptive windowing for online learning from multiple inter-related data streams. In M. Spiliopoulou, H. Wang, D. J. Cook, J. Pei, W. Wang, O. R. Zaïane, and X. Wu (Eds.), *Proceedings of International Workshop on 'Learning and Data Mining for Robots' at the 11th IEEE ICDM Workshops*, LEMIR 2011, Vancouver, Canada, pp. 697–704. IEEE Computer Society.

Ikonomovska, E. and S. Dzeroski (2011). Regression on evolving multi-relational data streams. In S. Müller-Feuerstein, B. Volz, K. Orsborn, and S. Stefanova (Eds.), *EDBT/ICDT Ph.D. Workshop*, EDBT/ICDT 2011, pp. 1–7. ACM Press.

Jin, R. and G. Agrawal (2003). Efficient decision tree construction on streaming data. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos (Eds.), *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2003, New York, New York, USA, pp. 576. ACM Press.

Johnson, N., A. Kemp, and S. Kotz (2005). *Univariate discrete distributions*. Wiley Inc.

Johnson, T., S. Muthukrishnan, and I. Rozenbaum (2005). Sampling algorithms in a stream operator. In *Proceedings of the 2005 ACM International Conference on Management of Data, SIGMOD 2005*, New York, NY, USA, pp. 1–12. ACM Press.

Kirsten, M., S. Wrobel, and T. Horváth (2001). Distance based approaches to relational learning and clustering. In *Relational Data Mining*, pp. 213–230. Springer-Verlag.

Kleinberg, E. M. (2000, May). On the algorithmic implementation of stochastic discrimination. *IEEE Transaction on Pattern Analysis Machine Intelligence 22*(5), 473–490.

Knobbe, A. J., M. de Haas, and A. Siebes (2001). Propositionalisation and aggregates. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery, PKDD 2001*, pp. 277–388. Springer-Verlag.

Kramer, S. and G. Widmer (2001). Inducing classification and regression trees in first order logic. In *Relational Data Mining*, pp. 140–156. Springer-Verlag.

Krempl, G., Z. F. Siddiqui, and M. Spiliopoulou (2011, Sept.). Online clustering of high-dimensional trajectories under concept drift. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis (Eds.), *Proceedings of the European Conference on Machine Learning and Principles of Knowledge Discovery, ECML PKDD 2011*, Volume 6912 of *Lecture Notes in Computer Science*, Athens, Greece. Springer-Verlag.

Kroegel, M. A. (2003). *On Propositionalization for Knowledge Discovery in Relational Databases*. Ph. D. thesis, Otto-von-Guericke-University Magdeburg, Germany.

Krogel, M.-A. and S. Wrobel (2002). Feature selection for propositionalization. In *Proceedings of the 5th International Conference on Discovery Science, DS 2002*, London, UK, UK, pp. 430–434. Springer-Verlag.

Lachiche, N. (2005). Good and bad practices in propositionalisation. In S. Bandini and S. Manzoni (Eds.), *AI*IA*, Volume 3673 of *AI*IA 2005*, pp. 50–61. Springer.

Lavrač, N. and P. Flach (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic 2*(4), 458–494.

Law, Y.-N. and C. Zaniolo (2007). Load shedding for window joins on multiple data streams. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, ICDEW 2007, Washington, DC, USA, pp. 674–683. IEEE Computer Society.

Liu, B., W. Hsu, and Y. Ma (1998). Integrating classification and association rule mining. In R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro (Eds.), *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge discovery and data mining, KDD 1998*. ACM Press.

McGovern, A., N. C. Hiers, M. W. Collier, D. J. G. II, and R. A. Brown (2008). Spatiotemporal relational probability trees: An introduction. In *Proceedings of the 8th IEEE International Conference on Data Mining*, ICDM 2008, pp. 935–940. IEEE Computer Society.

Metwally, A., D. Agrawal, and A. E. Abbadi (2005). Efficient computation of frequent and top-k elements in data streams. In T. Eiter and L. Libkin (Eds.), *ICDT*, Volume 3363 of *ICDT 2005*, pp. 398–412. Springer.

Metwally, A., D. Agrawal, and A. E. Abbadi (2006, September). An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems 31*, 1095–1133.

Mierswa, I., M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler (2006). Yale: Rapid prototyping for complex data mining tasks. In L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad (Eds.), *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge discovery and data mining*, KDD 2006, pp. 935–940. ACM Press.

Muggleton, S. and L. D. Raedt (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming 19/20*, 629–679.

Nasraoui, O., C. C. Uribe, C. R. Coronel, and F. Gonzalez (2003). Tecnostreams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, ICDM 2003, Washington, DC, USA, pp. 235. IEEE Computer Society.

O'Callaghan, L., N. Mishra, A. Meyerson, S. Guha, and R. Motwani (2001). Streaming-data algorithms for high-quality clustering. In *Proceedings of*

the 17th IEEE International Conference on Data Engineering, ICDE 2001, pp. 685. IEEE Computer Society.

Pasquier, N., Y. Bastide, R. Taouil, and L. Lakhal (1999, March). Efficient mining of association rules using closed itemset lattices. *Information System 24*(1), 25–46.

Perlich, C., P. Melville, Y. Liu, G. Swirszcz, R. D. Lawrence, and S. Rosset (2008). Breast cancer identification: Kdd cup winner's report. *SIGKDD Explorations 10*(2), 39–42.

Perlich, C. and F. J. Provost (2003). Aggregation-based feature invention and relational concept classes. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos (Eds.), *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 2003, pp. 167–176. ACM Press.

Perlich, C. and F. J. Provost (2006). Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning 62*(1-2), 65–105.

Quinlan, J. R. (1983). Learning from noisy data. In *Proceedings of the Second International Machine Learning Workshop*, MLW 1983, Urbana-Champaign, IL, pp. 58–64.

Quinlan, J. R. (1986, March). Induction of decision trees. *Machine Learning 1*, 81–106.

Rymon, R. (1993). An se-tree based characterization of the induction problem. In *Proceedings of the 10th International Conference on Machine Learning*, ICML 1993, pp. 268–275. Morgan Kaufmann Publishers Inc.

Schlimmer, J. C. and J. R. H. Granger (1986). Incremental learning from noisy data. *Machine Learning 1*(3), 317–354.

Siddiqui, Z. F., M. Oliveira, J. Gama, and M. Spiliopoulou (2012). Where are we going? predicting the evolution of individuals(to appear). In *Proceedings of the 11th International Symposium on Intelligent Data Analysis, IDA 2012*, Lecture Notes in Computer Science. Springer-Verlag.

Siddiqui, Z. F. and M. Spiliopoulou (2009a). Combining multiple interrelated streams for incremental clustering. In *Proceedings of 21st International Conference on Scientific and Statistical Database Management, SSDBM 2009*. Springer-Verlag.

Siddiqui, Z. F. and M. Spiliopoulou (2009b). Stream clustering of growing objects. In *Proceedings of the 12th International Conference on Discovery Science, DS 2009*. Springer-Verlag.

Siddiqui, Z. F. and M. Spiliopoulou (2010). Tree induction for perennial objects. In *Proceedings of 22nd International Conference on Scientific and Statistical Database Management, SSDBM 2010*. Springer-Verlag.

Siddiqui, Z. F. and M. Spiliopoulou (2011). Classification rule mining for a stream of perennial objects. In *Proceedings of the 5th International Symposium on Rules: Research Based and Industry Focused, RuleML 2011*. Springer-Verlag.

Siddiqui, Z. F., M. Spiliopoulou, P. Symeonidis, and E. Tiakas (2011). A data generator for multi-stream data. In *Proceedings of 2nd Workshop on Mining Ubiquitous and Social Environments at ECML PKDD 2011*, MUSE 2011.

Srivastava, R. and M. Gore (2008). A micro-clustering based method for multi-class classification in multi-relational databases. pp. 14–20. CSREA Press.

Srivastava, U. and J. Widom (2004). Memory-limited execution of windowed stream joins. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer (Eds.), *Proceedings of the 30th International Conference on Very Large Data Bases, VLDB 2004*, pp. 324–335. VLDB Endowment.

Supinie, T. A., A. McGovern, J. Williams, and J. Abernathy (2009). Spatiotemporal relational random forests. In Y. Saygin, J. X. Yu, H. Kargupta, W. Wang, S. Ranka, P. S. Yu, and X. Wu (Eds.), *Proceedings of ICDM Workshops 2009, the 9th IEEE International Conference on Data Mining Workshops, Miami, Florida, USA, 6 December 2009*, ICDMW 2009, pp. 630–635. IEEE Computer Society.

Symeonidis, P., A. Nanopoulos, A. Papadopoulos, and Y. Manolopoulos (2008). Collaborative recommender systems: Combining effectiveness and efficiency. *Expert Systems with Applications 34*(4), 2995–3013.

Symeonidis, P., E. Tiakas, and Y. Manolopoulos (2010). Transitive node similarity for link prediction in social networks with positive and negative links. In *Proceedings of the 4th ACM International Conference on Recommender systems, RecSys 2010*, pp. 183–190. ACM Press.

Tan, P.-N., M. Steinbach, and V. Kumar (2004). *Introduction to Data Mining*. Wiley Inc.

Thomas, S., S. Bodagala, K. Alsabti, and S. Ranka (1997). An efficient algorithm for the incremental updation of association rules in large databases. In D. Heckerman, H. Mannila, and D. Pregibon (Eds.), *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge discovery and data mining, KDD 1997*, pp. 263–266. ACM Press.

Utgoff, P. E. (1988). Id5: An incremental id3. In *Proceedings of the 5th International Conference on Machine Learning*, ICML 1988, pp. 107–120. Morgan Kaufmann Publishers Inc.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Journal of Machine Learning 4*.

Veloso, A., J. W. Meira, M. Carvalho, B. Possas, S. Parthasarathy, and J. Zaki (2002). Mining frequent itemsets in evolving databases. In *Proceeding of the SIAM International Conference on Data MiningSDM 2006*. ACM Press.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*, 37–57.

Vitter, J. S. (1987). An efficient algorithm for sequential random sampling. *ACM Transactions on Mathematical Software 13*(1), 58–67.

Wang, K., S. Zhou, and Y. He (2000). Growing decision trees on support-less association rules. In R. Ramakrishnan, S. J. Stolfo, R. J. Bayardo, and I. Parsa (Eds.), *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD 2000. ACM Press.

Widmer, G. and M. Kubat (1996). Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning 23*(1), 69–101.

Wikipedia (2012a). Mean absolute error.

Wikipedia (2012b). Root mean square deviation.

Witten, I. H. and E. Frank (2005). *Data Mining: Practical machine learning tools and techniques* (2nd ed.). Morgan Kaufmann Publishers Inc.

Xie, J., J. Yang, and Y. Chen (2005). On joining and caching stochastic streams. In *Proceedings of the 2005 ACM International Conference on Management of Data, SIGMOD 2005*, New York, USA, pp. 359–370. ACM Press.

Yan, F., M. Zhang, G. Liu, W. Wang, and Z. Deng (2011). Evolution of social networks: New patterns and a new generator. In *The Ninth International Conference on Creating, Connecting and Collaborating through Computing (C5)*, CCCCC 2011, pp. 3–10. IEEE: IEEE Computer Society.

Yeh, M.-Y., B.-R. Dai, and M.-S. Chen (2007, October). Clustering over multiple evolving streams by events and correlations. *IEEE Transactions on Knowledge and Data Engineering 19*(10), 1349–1362.

Yin, X. and J. Han (2005). Cross-relational clustering with user's guidance. In R. Grossman, R. J. Bayardo, and K. P. Bennett (Eds.), *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge discovery and data mining*, KDD 2005, pp. 344–353. ACM Press.

Yin, X., J. Han, J. Yang, and P. S. Yu (2006, June). Efficient classification across multiple database relations: A crossmine approach. *IEEE Transactions on Knowledge and Data Engineering 18*, 770–783.

Yu, P. S. and Y. Chi (2009). Association rule mining on streams. In *Encyclopedia of Database Management Systems*.

Zaki, M. J. and C.-J. Hsiao (2005, April). Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering 17*(4), 462–478.

APPENDIX A

Additional Material

A.1 Generating Zipf Distribution using Python

```
def generateZipFan2(n=100, drift=5, b=50):
    # drift: no of drifts
    # n: no of vals in each drifts
    # b: no of batches
    import random

    n=n*4
    stream = []
    for i in range(drift):
        a = 1.01
        s = generateZipFan(a,n)
        s = np.array(s)

        max_v = int(100000*a)
        s = s[s<max_v]
        max_v = int((np.random.uniform()+1)*1000000)%max_v
        min_v = int(max_v*0.98)
        for j in range(len(s)):
            s[j] = max_v-s[j]

        s = s[s>min_v]
        if (i%2==0):
            for j in range(len(s)):
                s[j] = s[j]+(s[j]%(1))*10000

        s = list(s)
        random.shuffle(s)
        stream.extend(s[:n/4])
    return stream
```

A.2 Generating Dataset with Multi-Gen

In Table A.1 the complete parameters for generating the synthetic datasets used in the main section of the thesis.

A.3 Experiment with Caching Strategies

In this section we report some further experiments, which we could not place in the main section of this thesis.

A.3.1 Financial Dataset with an Adjusted Baseline

We report some more experiments on the Financial dataset. We have modified the baseline slightly so that it still knows one timepoint into future, but rather than utilising that knowledge of future directly, it utilises it via the counts it maintains for each object, i.e., it first updates its counters from the data at t_{i+1} and then decides on basis of those counts what to store in the cache. Under these conditions, even the baseline suffers performance degradation. These experiments are reported in Figures A.1 to A.3.

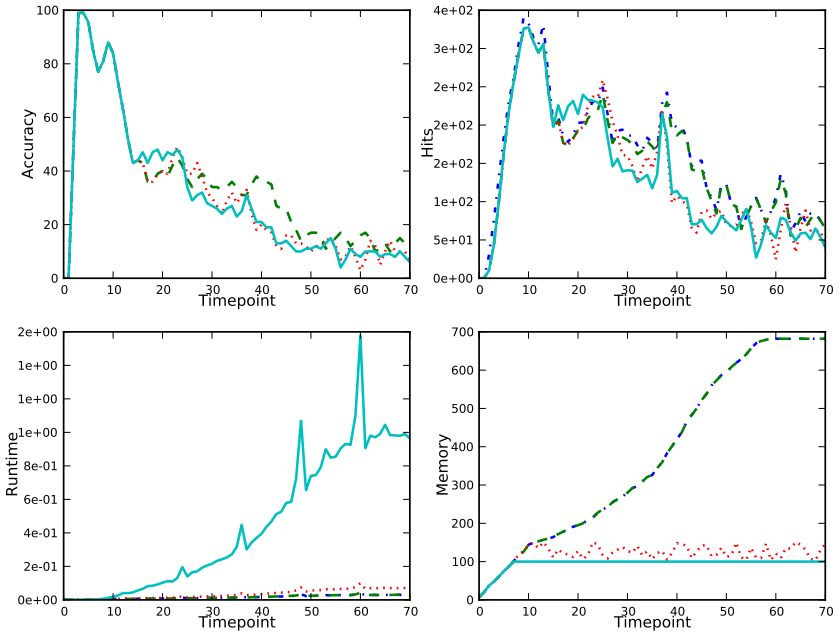

Figure A.1: Plots with an Adjusted Baseline for Financial Dataset: $N^{\mathcal{T}}$=50, M=100, β=1

Table A.1: Parameters for various dataset generated from Multi-Gen.

Dataset	N^i	n^i	v^i	N^u	n^u	v^u	τ_d	$\mathcal{L}$	s, v	PKS	UP2IP
Ratings2	5	200	4	5	600	1	4	10	21.5, 10000	0.8	0.75
Ratings	20	20×100	4	5	600	1	2	10	21.5, 10000	0.8	0.75
									0 to η^a		
Ratings 1500r	8	8×40	5	6	600	1	1	50	0 to 5	0.2	0.5
Ratings 6500r	8	8×40	5	6	600	1	1	100	0 to 21	0.8	0.2
Drift 150u	8	8×40	5	6	150	1	2	30,50	0 to 21	0.75	0.2
Drift 1000u	8	8×40	5	6	1000	1	2	50,30	0 to 21	0.5	0.2

[a] In an earlier version of the generator, the number of generated ratings were generated using the uniform distribution per timepoint instead of Zipf distribution. An unpublished article describing this version is available at: http://omen.cs.uni-magdeburg.de/itikmd/mitarbeiter/zaigham-faraz-siddiqui.html

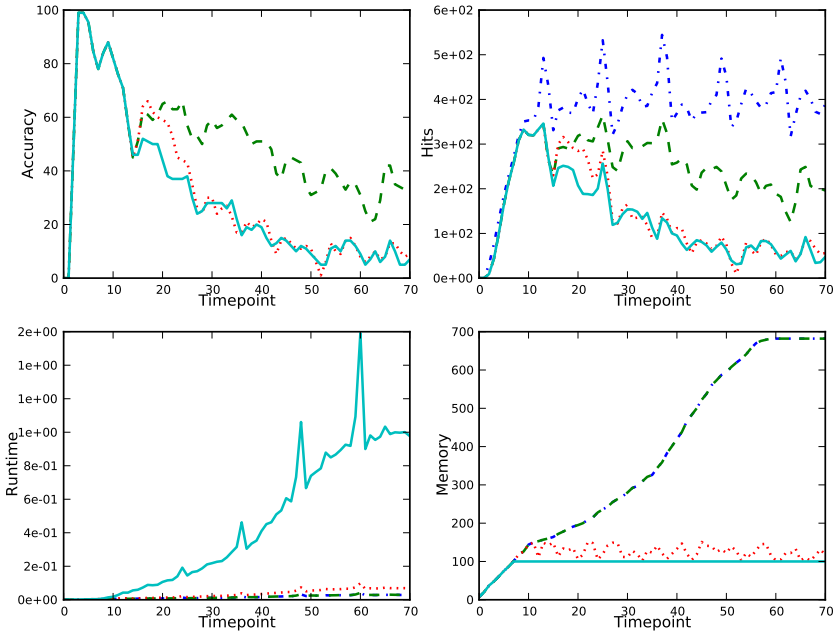

Figure A.2: Plots with an Adjusted Baseline for Financial Dataset: $N^{\mathcal{T}}$=50, M=100, β=2

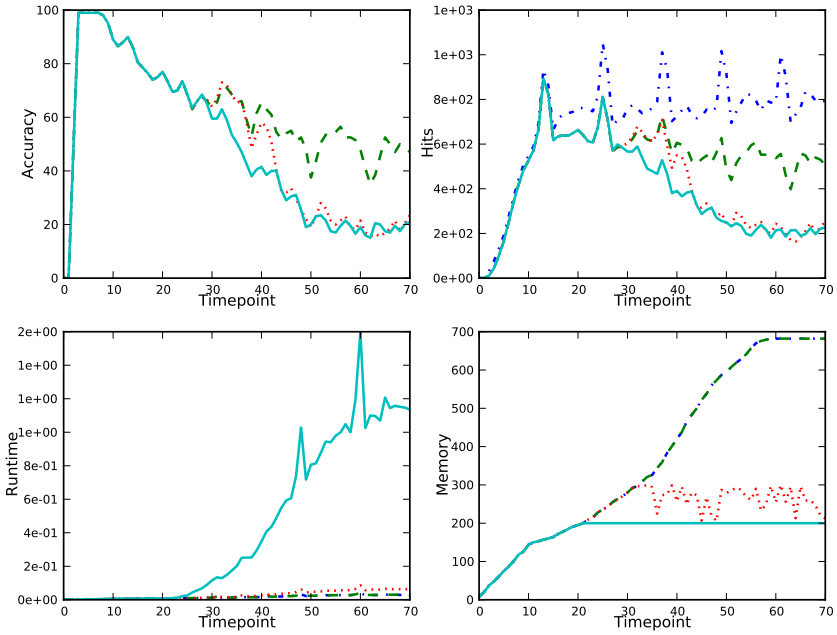

Figure A.3: Plots with an Adjusted Baseline for Financial Dataset: $N^{\mathcal{T}}$=100, M=200, β=2

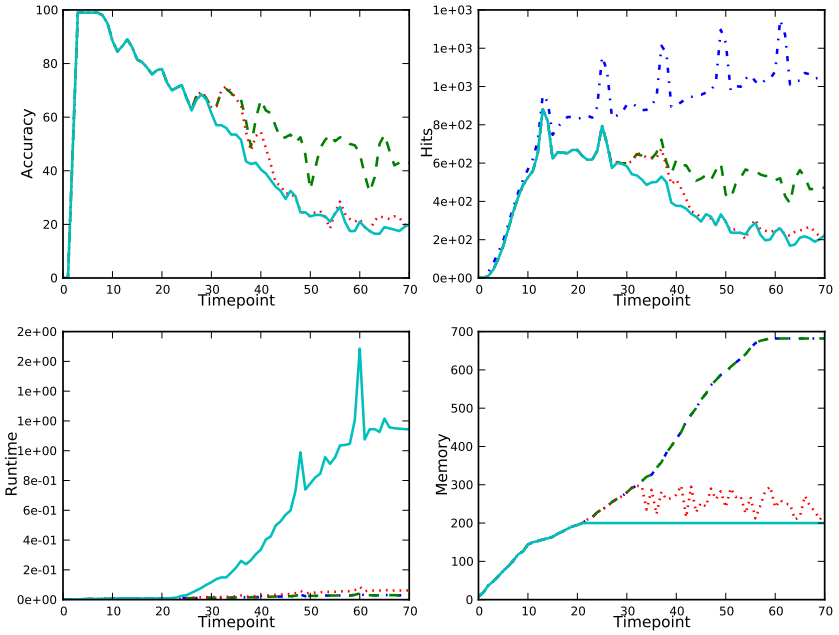

Figure A.4: Plots for Financial Dataset: $N^{\mathcal{T}}$=100, M=200, β=5

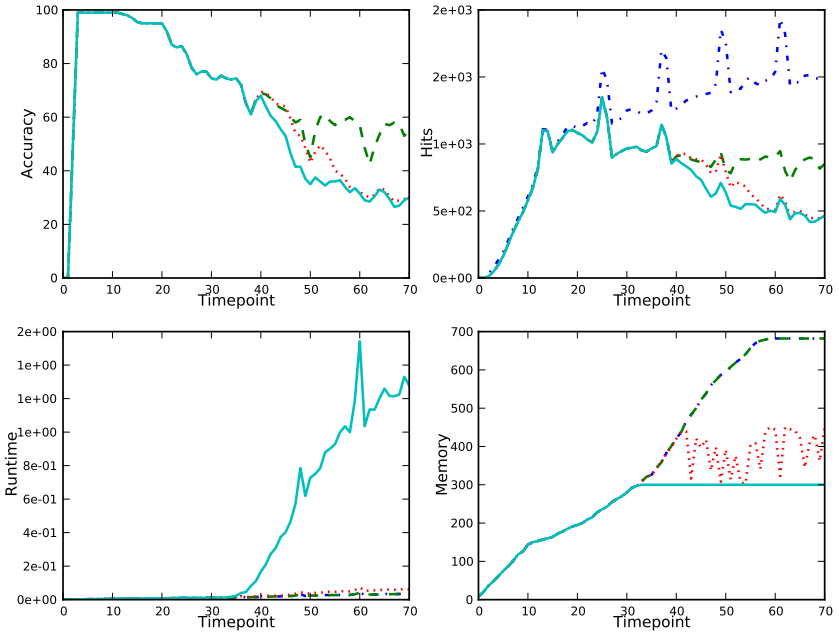

Figure A.5: Plots for Financial Dataset: $N^{\mathcal{T}}$=150, M=300, β=5

Declaration

I hereby declare that I, the undersigned, have conducted this work without the undue help of others, have not made use of any tools other than specified, and ideas taken directly or indirectly from other sources are referenced.

In particular, I have not taken the help of any commercial doctoral consultant. No third party has received monetary benefits, either directly or indirectly, for the work presented in the submitted dissertation.

This work has not been submitted for review at any other place, either at home or abroad, in this form or a similar form. This work as a whole has not yet published.

Place, Date

Unterschrift

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Ort, Datum

Signature