Hashing for Large-Scale Structured Data Classification

Lianhua Chi

A thesis submitted for the Degree of

Doctor of Philosophy

Faculty of Engineering and Information Technology

University of Technology, Sydney 2015

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate:

Date:

ACKNOWLEDGEMENTS

I would like to express my earnest thanks to my principal supervisor, Dr. Ling Chen, co-supervisor, Professor Xingquan Zhu, and Dr. Bin Li who have provided tremendous support and guidance for my research. Their comprehensive guidance has covered all aspects of my PhD study, including research methodology, research topic selection, experiments, academic writing skills and thesis writing, and even the sentence structure and formulas. Their critical comments and suggestions have strengthened my study significantly. Their strict academic attitude and respectful personality have benefited my PhD study and will be a great treasure throughout my life. Without their excellent supervision and continuous encouragement, this research could not have been finished on time. Thanks to you all for your kind help.

I am grateful to all members of the centre for Quantum Computation and Intelligent Systems (QCIS) for their careful participation in my presentation and valuable comments for my research. I especially thank Professor Chengqi Zhang, and PhD students Mr Ting Guo, Mr Shirui Pan, Mr Meng Fang, Mr Zhibing Hong, Ms Hongshu Chen and Mr Junyu Xuan, and other students for their contributions and help during my PhD study.

I am grateful to FEIT Travel fund, Vice-Chancellor's Postgraduate Conference Fund, and the ARC Discovery Scholarship.

Last but not least, I would like also to thank my family members. Thanks to my mother and father for their constant encouragement and generous support. Thanks to my husband for his unconditional support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With the rapid development of the information society and the wide applications of networks, almost incredibly large numbers bytes of data are generated every day from the social networks, business transactions and so on. In such cases, hashing technology, if done successfully, would greatly improve the performance of data management. The goal of this thesis is to develop hashing methods for large-scale structured data classification.

First of all, this work focuses on categorizing and reviewing the current progress on hashing from a data classification perspective.

Secondly, new hashing schemes are proposed by considering different data characteristics and challenges, respectively. Due to the popularity and importance of graph and text data, this research mainly focuses on these two kinds of structured data:

1) The first method is a fast graph stream classification method using Discriminative Clique Hashing (DICH). The main idea is to employ a fast algorithm to decompose a compressed graph into a number of cliques to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are employed to compress the original edge set of the graph stream and map the unlimitedly increasing clique-patterns onto a fixed-size feature space, respectively. DICH essentially speeds up the discriminative clique-pattern mining process and solves the unlimited clique-pattern expanding problem in graph stream mining;

2) The second method is an adaptive hashing for real-time graph stream classification (ARC-GS) based on DICH. In order to adapt to the concept drifts of the graph stream, we partition the whole graph stream into consecutive graph chunks. A differential hashing scheme is used to map unlimited increasing features (cliques) onto a fixed-size feature

space. At the final stage, a chunk level weighting mechanism is used to form an ensemble classifier for graph stream classification. Experiments demonstrate that our method significantly outperforms existing methods;

3) The last method is a Recursive Min-wise Hashing (RMH) for text structure. In this method, this study aims to quickly compute similarities between texts while also preserving context information. To take into account semantic hierarchy, this study considers a notion of "multi-level exchangeability", and employs a nested-set to represent a multi-level exchangeable object. To fingerprint nested-sets for fast comparison, Recursive Min-wise Hashing (RMH) algorithm is proposed at the same computational cost of the standard min-wise hashing algorithm. Theoretical study and bound analysis confirm that RMH is a highly-concentrated estimator.

CHAPTER 1 INTRODUCTION

1.1 MOTIVATION

With the rapid development of information society and the wide applications of network, almost incredibly large numbers bytes of data are generated every day from the social networks, business transactions, sensors, and entertainment industry and so on. The characteristics of these data mainly include large volume, quickly moving, different types (such as relation data, semi-structured data (XML), images, texts, videos and graphs). It is called as "big data".

Nowadays, mining of massive data [114] has become one of the most important research trends in the era of big data. The "4V" (volume, velocity, variety and veracity) nature of big data subverts the traditional learning paradigm because, in big data scenarios, the volume and the dimensionality of instances are usually unpredictable and increase rapidly. In such cases, even enumerating complete features to compute a similarity has become an intractable problem. For example, in document similarity search, the underlying feature space can easily exceed 108 dimensions if 5-shingles (5 continuous characters) are considered [114]; and the feature space can be much higher if a vocabulary of words as features is considered. Thus, it becomes urgent to develop fast approximate algorithms to address the storage and computation problems for big data.

However, due to the limitation of traditional data analysis tools, it is very difficult to store, process and analyse these big data. For the users, they hope to effectively process and analyse all available data and get more accurate and timely business intelligence. In order to meet the needs of users, attempts can be made to better manipulate and control the data. In such cases, the hashing technology can efficiently compress data for better

management. Research on hashing has attracted more and more attention since the idea of hashing started. Many hashing methods were developed to solve the problems of the curse of dimension and finding nearest neighbours. The hashing technology can learn binary-code representation for data in the hash code space and preserve the neighbourhood structure in the original feature space by mapping similar points in the original feature space to nearby binary codes in the hash code space. The compact representation in hashing can effectively save storage and achieve fast query in large scale data.

1.2 PROBLEM STATEMENT AND SOLUTIONS

The core goal of this research is to develop a set of effective hashing methods for large-scale structured data classification. Accordingly, a literature review is first carried out to survey existing hashing methods. In order to utilize hashing methods for efficient data management and classification, this research will place an emphasis on graph and text data, and will propose algorithms for hashing and classifying structured data from static sets and dynamic streams.

1.2.1 PROBLEM STATEMENT AND SOLUTIONS FOR GRAPH STRUCTURED DATA

The emergence of complex networks has led to a surge of research in graph data mining [87]. Graph classification is an important graph data mining task that aims to learn a discriminative model from training examples to predict class labels of test examples, where both training and test examples are graphs. Many real-world applications involve graph-represented data, such as chemical compounds, XML documents and program flows.

Fig. 1.1 shows an example of graph classification where two sets of compound graphs: “Known Graphs” and “Unknown Graphs”. In the “Known Graphs” are given as training examples. In the “Known Graphs” set, there are a graph labelled as “Organic Compound” and another graph labelled as “Non-organic Compound”. The task of graph classification

is to learn classification models, from the training graphs, in order to effectively predict the unknown graphs, by analysing the features or structures in the known graphs.

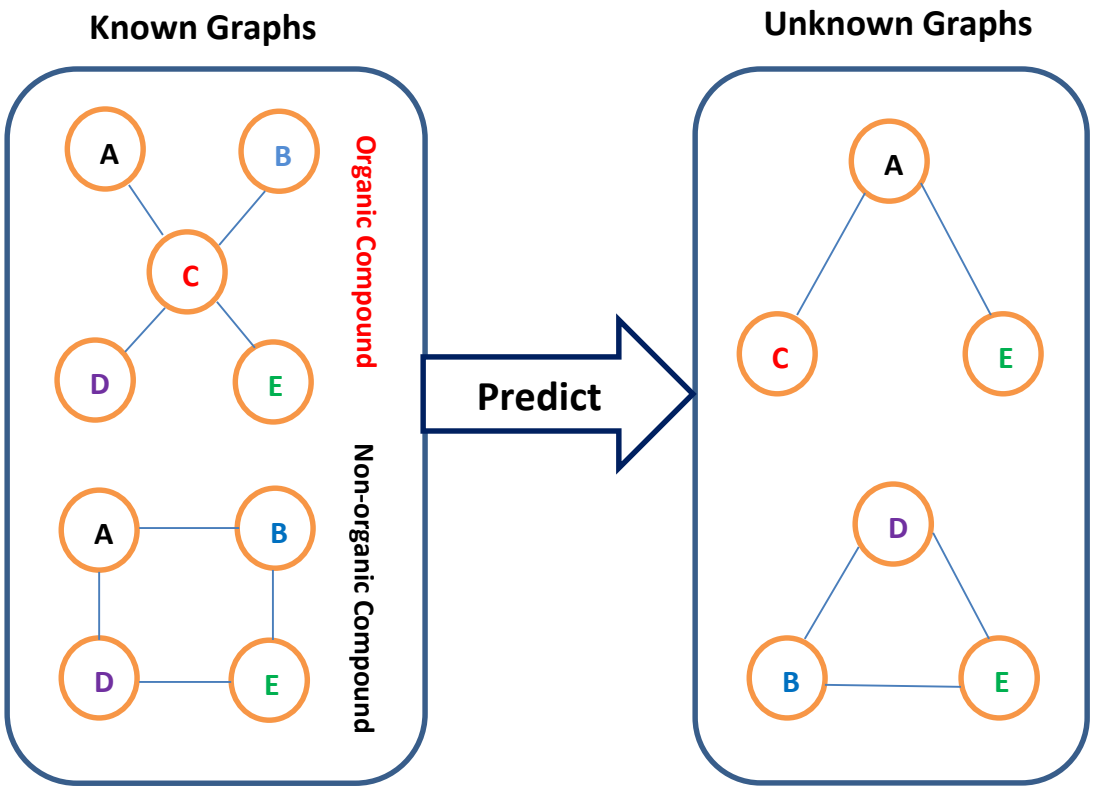


Figure 1.1: Graph classification

The essential challenge for graph classification is to extract features from graphs and represent graph data in instance-feature format to support model training. A variety of studies on substructure extraction (e.g., walks [88, 89], paths [90], and subtrees [91, 92]) for describing graphs have been proposed in the past decade. However, most of them only consider the learning problem of graph classification in batch mode (all data are available for training), which limits their applicability to large-scale and stream scenarios. In fact, dynamic networked data are often presented with increasing volumes and change over time in many real-world scenarios. For example, a social network is made up of a population of individuals, where the interactions among them keep generating, disappearing, and changing over time. A transportation network is a complex network made up of numerous interconnected routes, where the traffic flows over them are generated dynamically over time. Due to the streaming nature of many real-world

complex networks, graph stream classification has recently attracted increasing research interest [93-96]. However, the hashing for graph stream classification on a complex network with massive nodes is challenging, because

- **How to handle data stream:** The volumes of graph data are continuously growing, so graph streams can usually be accessed only once. The hashing for graph stream classification must be able to tackle dynamically increasing graph volumes and generate hashing values with high speed.

- **How to adapt to the changing feature distributions:** The marginal distributions of subgraph-patterns (features) may continuously change over the graph stream (that is, the concept-drift problem [96]). The hashing methods are required to effectively deal with the changing feature space.

In order to address these challenges, this study proposes a fast graph stream classification method using DIscriminative Clique Hashing (DICH) to address the aforementioned challenges. The main idea is to decompose a compressed graph into a number of cliques (fully connected subgraphs) to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are employed to compress the original edge set of the graph stream and map the unlimitedly increasing clique-patterns onto a fixed-size feature space, respectively. The hashed cliques are then used to update an "in-memory" fixed-size pattern-class table, which will be finally used to generate a rule-based classifier.

Meanwhile, motivated by the aforementioned challenges and the limitations of the existing approaches, in this thesis, there is a further proposal for an adaptive real-time classification method for graph stream using stochastic learning, differential hashing techniques and a chunk level weighting mechanism to address these problems. The detailed solutions for the aforementioned challenges and the limitations are highlighted as follows:

1. This study proposes an approximate method for fast graph feature extraction by detecting cliques from the compressed graphs via hashing. The method

significantly improves the efficiency of feature extraction and classifier learning online to satisfy the "real-time" requirement.

2. This study proposes a graph feature reduction method by mapping unlimitedly expanding clique patterns onto a fixed-size compatible feature space via differential hashing. This can avoid a pre-scan of graphs to further speed up the learning process, satisfying the "one-pass" requirement and adapting to the concept drifting.

3. This study adopts the stochastic learning strategy to incrementally train a graph classifier online, which can satisfy the "real-time" requirement and achieve better classification performance than the majority voting used in [95] [18].

Combined with a differential hashing scheme, a chunk level weighting mechanism is adopted to form a weighted classifier ensemble for graph stream classification, which can effectively adapt to concept drifting and achieve better performance than instance level weighting mechanism [103].

1.2.2 Problem Statement and Solutions for Text Structured Data

There have been a number of approximate algorithms for big data similarity computation. Since many high-dimensional data can be represented as bags of words, min-wise hashing [10] has been naturally applied to them for fast approximating set similarities without scanning and comparing the complete sets. However, all the current algorithms are based on the bag-of-words representation for its exchangeability that can facilitate random projection and hashing. A limitation of such *flat-set* representation is that context information and semantic hierarchy may be lost. Thus, a more expressive bag-of-words representation needs to be explored to relieve this problem.

In this thesis, the aim is to fast compute similarities between bag-of-words represented objects while also preserving context information inside the objects. The process still follows the random algorithm approach to this end. Relational learning and structural

patterns are not be considered to capture context information which might be unrealistic in big data scenarios. To take into account semantic hierarchy, a notion of multi-level exchangeability is considered which can be applied at word-level, sentence-level and paragraph-level. Then, a nested-set is employed to represent a multi-level exchangeable object, say "nested bag-of-words". For example, {{a, b, c, d}, {b, d, e}, {e, f, g}} represents a paragraph with three sentences, each of which further comprises several words. In this example, the top-level exchangeable elements are sentences while the bottom-level exchangeable elements are words. In such nested-set representations, context information and semantic hierarchy are preserved yet the resulting form is still simple for random algorithms. To fast compute a similarity between nested-sets, a Recursive Min-wise Hashing (RMH) algorithm is proposed for sketching nested-sets. The advantage of RMH is two-fold: 1) it accounts for multiple levels of exchangeabilities; 2) it enables a probabilistic comparison of sub-sets instead of hard matching.

1.3 CONTRIBUTIONS

This thesis focuses on exploring, developing and utilizing hashing scheme to solve large-scale structured data classification problems.

Our contributions to solve these problems are listed below:

- **Surveying the hashing methods:** a comprehensive overview of hashing methods is given in this thesis, which is the theoretical basis of our thesis for the large-scale structured data classification. This comprehensive overview not only becomes a helpful resource and guidance for further research on hashing methods and related research fields, but also provides theoretical guidance for the large-scale structured data classification. Our contributions are as follows:

  1: Summarize the development history of hashing and provide the theoretical background of hashing, which can help us establish a better understanding of the hashing methods.

2: Review the current hashing methods including data-independent hashing and data-dependent hashing methods, which provide a clear framework of current popular hashing methods.

3: Introduce the main applications of hashing, including the objective-oriented applications and data domain-oriented applications, which help us understand the importance of hashing methods.

- **Exploring the hashing scheme for graph structured data classification:** Motivated by the aforementioned challenges and the limitations of the existing approaches, in-depth research on graph structured data classification is done with the guidance of hashing theory. Our contributions are as follows:

  1: Propose a fast graph stream classification method using DIscriminative Clique Hashing (DICH) to address the aforementioned challenges.

  2: Propose an approximate method for fast graph feature extraction by detecting cliques from the compressed graphs via hashing. The method significantly improves the efficiency of feature extraction and classifier learning online to satisfy the "real-time" requirement.

  3: Propose a graph feature reduction method by mapping unlimitedly expanding clique patterns onto a fixed-size compatible feature space via differential hashing. This can avoid a pre-scan of graphs to further speed up the learning process, satisfying the "one-pass" requirement and adapting to the concept drifting.

  4: Adopt the stochastic learning strategy to incrementally train a graph classifier online, which can satisfy the "real-time" requirement and achieve better classification performance than the majority voting used in [95] [18].

  5: Combined with the differential hashing scheme, a chunk level weighting mechanism is adopted to form a weighted classifier ensemble for graph stream classification, which can effectively adapt to concept drifting and achieve better performance than an instance level weighting mechanism [103].

- **Exploring the hashing scheme for text structured data classification:** To fast compute a similarity for text data, a Recursive Min-wise Hashing (RMH) algorithm is proposed for sketching nested-sets. By virtue of RMH, two multi-level exchangeable objects can be compared with the same computational cost of the standard min-wise hashing algorithm while preserving context information as a plus. A theoretical bound to RMH is also provided to show it is a highly-concentrated estimator. Our contributions are as follows:

  1: Account for multiple levels of exchangeabilities;

  2: Enable a probabilistic comparison of sub-sets instead of hard matching.

1.4 THESIS STRUCTURE

The framework of the whole thesis is as follow:

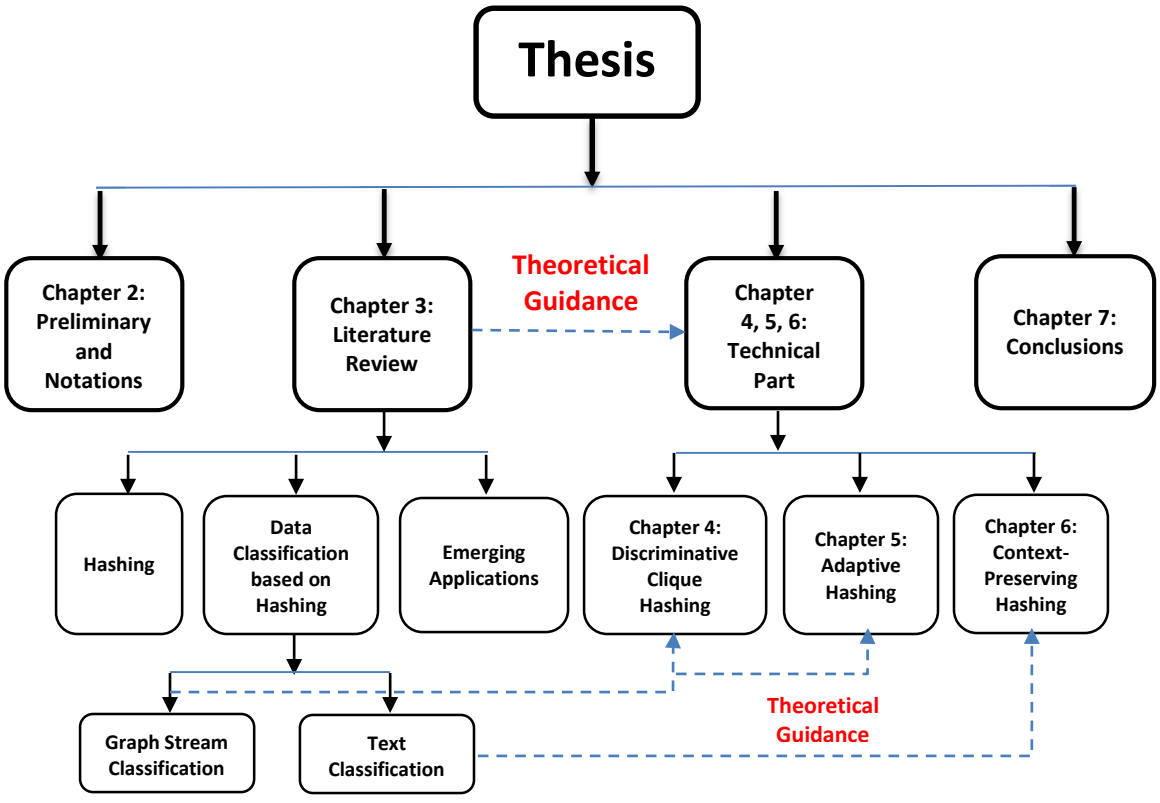


Figure 1.2: Thesis structure

The content of each chapter in this thesis is summarized as follows:

Chapter 2: This chapter provides preliminary concepts and definitions for the proposed models. It also summarizes major notations in the thesis.

Chapter 3: This chapter is a literature review that surveys existing works on hashing scheme from a data classification perspective. It summarizes major approaches in the field, along with their technical strengths/weaknesses, followed by a simple discussion about emerging hashing applications and challenges therein.

Chapter 4: This chapter presents a discriminative clique hashing for fast graph stream classification (DICH). It provides algorithm details, theoretical processes, and comparative experiments to valid its superiority to state-of-art methods.

Chapter 5: This chapter describes an adaptive hashing for real-time graph stream classification (ARC-GS) in detail. It explains the motivations and the principles of graph stream classification, provides the theoretical basis and interpretations for the proposed work, and shows comparative experimental results with baseline methods and benchmark data sets.

Chapter 6: This chapter introduces a context-preserving hashing for fast text classification (RMH). It explains the motivations and the principles of fast text classification, provides the theoretical basis, interpretations and time complexity for the proposed work. Details for the analysis of the comparisons results of experiments performed is also presented in this chapter.

Chapter 7: This chapter concludes this thesis and outlines the direction for future work.

1.5 PUBLICATIONS

Below is a list of the refereed international journal and conference papers associated with my PhD research that have been submitted, accepted and published:

1) Lianhua Chi, Bin Li, Xingquan Zhu: Context-Preserving Hashing for Fast Text Classification. SDM 2014: 100-108.

2) Lianhua Chi, Bin Li, Xingquan Zhu: Fast Graph Stream Classification Using Discriminative Clique Hashing. PAKDD (1) 2013: 225-236

3) Ting Guo, Lianhua Chi, Xingquan Zhu: Graph hashing and factorization for fast graph stream classification. CIKM 2013: 1607-1612

4) Zongmin Cui, Hong Zhu, Lianhua Chi: Lightweight key management on sensitive data in the cloud. Security and Communication Networks 6(10): 1290-1299 (2013)

5) Zongmin Cui, Hong Zhu, Jie Shi, Lianhua Chi, Ke Yan: Lightweight Management of Authorization Update on Cloud Data. ICPADS 2013: 456-461

6) Bin Li, Xingquan Zhu, Lianhua Chi, Chengqi Zhang: Nested Subtree Hash Kernels for Large-Scale Graph Classification over Streams. ICDM 2012: 399-408

7) Lianhua Chi, Yucai Feng, Hehua Chi, Ying Huang: Face image recognition based on time series motif discovery. GrC 2012: 72-77

8) Yucai Feng, Lianhua Chi, Hehua Chi, Chuanlu Liu, Zhuo Liu: QoM: An effective querying method for time series database. GrC 2012: 129-134

9) Juebo Wu, Hehua Chi, Lianhua Chi: A Cloud Model-based Approach for Facial Expression Synthesis. Journal of Multimedia 6(2): 217-224 (2011)

10) Lianhua Chi, Hehua Chi, Yucai Feng, Shuliang Wang, Zhongsheng Cao: Comprehensive and Fast Discovery of Time Series Motifs. Journal of Zhejiang University SCIENCE C (JZUS-C) 2011.

CHAPTER 2 PRELIMINARY CONCEPTS AND NOTATIONS

2.1 DEFINITIONS

2.1.1 DEFINITIONS FOR GRAPH STRUCTURED DATA

Suppose there exists a complex network which comprises a massive universe of nodes. Edges connecting these nodes are denoted by an edge set ε. The stream of graphs $\{\cdots, G_{n-1}, G_n, G_{n+1}, \cdots\}$ are presented sequentially as the subsets of ε (all are connected graphs), where the subscript n denotes the receiving order of the graph in the stream. An example of such graph streams is a coauthor network. All the papers (graphs of connected authors) on the coauthor network with different time-stamps form a graph stream.

Definition 1. ***Connected Graph*****:** *A graph is represented as* $G = \{\mathcal{V}, \mathcal{E}, \mathcal{L}\}$ *where* $\mathcal{V} = \{v_1, \ldots, v_V\}$ *is the set of vertices,* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *is the set of edges, and* $\mathcal{L}$ *is the label*[1] *set of the vertices and edges. A connected graph is a graph in which there exists a path between any pair of vertices.*

Definition 2. ***Graph Stream*****:** *A graph stream* $\mathcal{S} = \{G_{1,\ldots}, G_{n-1}, G_n, G_{n+1,\ldots}\}$ *is a sequence of graphs which arrive one after another in a stream fashion.*

[1] Note that the labels of nodes and edges are different notations from the class labels of graphs.

Definition 3. *Clique*: *A clique* $\mathcal{C} = \{\mathcal{V}', \mathcal{E}', \mathcal{L}'\}$ *in a graph* $G = \{\mathcal{V}, \mathcal{E}, \mathcal{L}\}$ *is a subgraph that satisfies* $\mathcal{V}' \subseteq \mathcal{V}$ *and any pair of vertices in* $\mathcal{V}'$ *are connected by an edge (i.e., a complete graph).*

Clique is widely used as a fundamental unit for structural analysis and knowledge discovery in graphs. Although finding maximum clique is NP-complete [104], many algorithms for finding cliques have been developed in exponential time and even polynomial time for certain type of graphs. Because graphs do not have features available for representation, in this thesis, we propose to use cliques as features to represent graphs. While many methods exist to represent graphs by using frequent subgraph patterns or using whole graphs (e.g., graph kernels). Cliques have the following two major advantages for graph representation:

(1) For dynamic graph streams, frequent subgraph patterns are rapidly changing, which makes the feature space exponentially grows and drastically changes. As a result, a classifier built from historical graphs might not be used to accurately represent and classify future graphs because they have different subgraph feature space. In comparison, cliques are basic graph structures remaining relatively stable for graph streams. By using cliques as features to represent graphs, historical and future graphs can be ensured to have shared common feature space for learning;

(2) Finding cliques is much more efficient than finding frequent subgraph patterns (our clique finding details will be elaborated in the next section), so our method can rapidly discover graph features for learning.

(3) Compared to frequent subgraph patterns or a whole graph, clique is a relatively small structure unit, which implies that clique might not be sufficiently accurate for graph representation (or incur more information loss). Nevertheless, our research will show that although graphs are complex in structures, a graph can often be decomposed into a small number of structure units. This allows our research to use a relatively large number of small cliques to represent a large set of graphs, without resulting in severe information loss.

Definition 4. ***Clique Features*****:** *Let* $\mathcal{C} = \{C_1, \dots C_M\}$ *denote a set of clique patterns (or clique features). For a graph* G_n*, a vector* $x^{(n)} = \left[{x_1}^{(n)}, \dots {x_M}^{(n)}\right]^T$ *is used to represent* G_n *in the clique-feature space spanned by* $\mathcal{C}$*, where* ${x_m}^{(n)}$ *is the number of clique pattern* C_m *found in* G_n*.*

Specifically, the edge set of G_n is denoted by $\varepsilon_n = \{e_1, \cdots, e_E\} \subset \varepsilon$. Each graph G_n has a class label $l_n \in \{1, \cdots, L\}$. Each received graph G_n is represented in the form of $\langle n, e_1, \cdots, e_E, l_n \rangle$. In this chapter, it is assumed that each edge in a graph has a numerical weight w_{ij}, where i and j are the indices of the two vertices of the edge (for simplicity, edge labels are not considered).

2.1.2 DEFINITIONS FOR TEXT STRUCTURED DATA

Definition 5. ***Bag-of-Words*****:** *A bag of words is the unordered collection of words in a text* d*.*

It is a simplified representation disregarding the structural information. Due to its simplicity, bag-of-words has been accepted as a standard model in information retrieval and text mining, especially in massive data scenarios. For text classification, there are two commonly used bag-of-words representations:

(1) *Term Frequency (TF)*, which counts the occurrence of each word in d and let the count be the value of the corresponding feature dimension. The resulting form is a feature vector $x \in R^V$ whose dimensions are spanned by V terms in a predefined vocabulary. It is common to use inverse document frequency (IDF) to weight TF for emphasizing uncommon terms [114].

(2) *(Multi-) Set*, which views all words in d as a set S; if the same word is allowed to appear multiple times, it is a multi-set. This representation is easier than TF since no predefined vocabulary (feature space) is required, hence it is more popular in high-dimensional data scenarios.

Definition 6. ***Min-wise Hashing*****:** *The min-hash scheme [10] is an approximate method for measuring the similarity of two sets, say S_a and S_b.*

K hash functions (random permutations) $\{\pi_k\}_{k=1}^{K}$ are applied to the elements in S_* and $\min(\pi_k(S_*))$ is an min-hash of S_*. An advantageous property of min-hash is that the probability of S_a and S_b to generate the same min-hash value is exactly the Jaccard similarity of S_a and S_b:

$$Pr(\hbar(S_a) = \hbar(S_b)) = \frac{|S_a \cap S_b|}{|S_a \cup S_b|} = J(S_a, S_b) \tag{2.1}$$

where $\hbar(\cdot) = min(\pi(\cdot))$ is written as a shorthand. In practice, multiple independent random permutations are used to generate min-hashes to approach the expected probability. The similarity between the two sets based on the K min-hashes is calculated by

$$\hat{J}(S_a, S_b) = \frac{\sum_{k=1}^{K} 1(\hbar(S_a) = \hbar(S_b))}{K} \tag{2.2}$$

where 1(*state*) = 1, if *state* is true; and 1(*state*) = 0, otherwise. As $\rightarrow \infty$, $\hat{J}(S_a, S_b) \rightarrow J(S_a, S_b)$; that is

$$E[1(\hbar_k(S_a) = \hbar_k(S_b))] = J(S_a, S_b) \tag{2.3}$$

Definition 7. ***Feature Hashing*****:** *Feature hashing [116] provides an unbiased and highly-concentrated estimator of the inner product of high-dimensional feature vectors. It is closely related to the random projection [9, 1]. The difference is that the projection matrix* $\boldsymbol{P}$ *only comprises values in* $\{1, -1\}$*, that is.,* $\boldsymbol{P} \in \{1, -1\}^{K*V}$ *where V is the original dimensionality and K is the new,* $\ll V$ *. A constraint on* $\boldsymbol{P}$ *is that each column is allowed to have only one non-zero entry. The positions of non-zero entries and its signs are randomly generated.*

Given a feature vector $x \in R^V$ (that is, TF), the hashed feature vector gives $\bar{x} = \boldsymbol{P}x \in R^K$. The intuition of this operation is to randomly partition the features into groups and sum up the signed features in the same group, where the sign is added to eliminate bias (a biased version without signs is [115]).

In practice, it is not necessary to explicitly define the projection matrix. Two random hash functions can be directly applied to the terms $\{w_1, w_2, \dots\}$ in d to calculate the hashed TF feature vector

$$\bar{x} = [\bar{x}_1, \dots, \bar{x}_K]^T, and\ \bar{x}_K = \sum_{i:\xi(w_i)=k} x_i\, \delta(w_i) \tag{2.4}$$

where $\xi: str \mapsto \{1, \dots, K\}$ and $\delta: str \mapsto \{1, -1\}$ are two random hash functions. Due to its implicit projection property, feature hashing is extremely useful in big data scenarios where data may have infinite features. It has been adapted to many applications, such as multi-task learning [116], collaborative filtering [113], and graph stream classification [17].

Definition 8. ***Multi-Level Exchangeable Representations*****:** *The multi-level exchangeable representation of a text is a nested set* $S^{(R)} = \{S_1^{(r-1)}, \dots, S_{|S_*^{(r)}|}^{(r-1)}\}$, *where*

$$S_*^{(r)} = \{S_1^{(r-1)}, \dots, S_{|S_*^{(r)}|}^{(r-1)}\} \tag{2.5}$$

for $r = 2, \dots, R-1$. $S_*^{(1)}$ *denotes a set of words as atomic elements and* $S^{(R)}$ *denotes a set of the highest-level exchangeable objects (that is, paragraphs).*

2.2 NOTATIONS

Major notations used in this thesis are summarized in Table 2.1.

Table 2.1: Notations used in the thesis

Symbols	Explanations
$\mathcal{E}$	the edge set
$\{G_1, G_2, \dots, G_i, \dots\}$	the stream of graphs
i	the receiving order in the graph stream
$\{E_1, \dots, E_e\} \subset \mathcal{E}$	the edge set of G_i
e	the number of edges in G_i
$L_i \in \{1, \dots, M\}$	the class label of G_i
$\langle i, E_1, \dots, E_e, L_i \rangle$	the receiving form of G_i
G_{test}	the test graph
$\bar{\mathcal{E}}$	the compressed edge set
$\bar{G}_i$	the compressed graph

$\{\max(\bar{G}_i), \ldots, \min(\bar{G}_i)\}$	the descending order of a number of weight levels
$\max(\bar{G}_i)$	the largest edge weights in $\bar{G}_i$
$\min(\bar{G}_i)$	the smallest edge weights in $\bar{G}_i$
Δ	the "in-memory" $P \times M$ pattern-class table
P	the indices of hashed clique-patterns
M	all the classes of the graphs
C_i	the clique set of G_i
$C_{i,j}$	each clique in C_i
$h(\cdot)$	a random hash function
$H_{i,j} \in \{1, \ldots, P\}$	an index of $C_{i,j}$
θ	a threshold parameter of discriminative capability
$G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$	A graph
$\mathcal{V} = \{v_1, \ldots, v_V\}$	the set of vertices
$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$	the set of edges
$\mathcal{L}$	the label of set of the vertices and edges
$C = (\mathcal{V}', \mathcal{E}', \mathcal{L}')$	a clique (complete subgraph) in a graph
$\mathcal{C} = \{C_1, \ldots C_M\}$	$G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$
n	a set of clique patterns
$\mathbf{x}^{(n)} = [x_1^{(n)}, \ldots x_M^{(n)}]^\top$	the receiving order of the graph in the stream
	a representation of G_n in the clique-feature space spanned by $\mathcal{C}$, where $x_m^{(n)}$ is the number of clique pattern C_m found in G_n
$\mathcal{E}_n = \{e_1, \ldots, e_E\} \subset \mathcal{E}$	the edge set of G_n
$l_n \in \{1, \ldots, L\}$	a class label of each graph G_n
$\langle n, e_1, \ldots, e_E, l_n \rangle$	the form of each received graph G_n
$\bar{w}_{ij}$	the weight of each edge in $\bar{G}_n$
g	a subgraph in G
K	the number of chunk classifiers in an ensemble
M	the range of clique hash values
R	a ratio value
C_{new}	the new cliques set
C_{old}	the old cliques set
$M * R$	the size of hash space of new cliques C_{new}
$M * (1 - R)$	the size of hash space of old cliques C_{old}
$C_{new,k}$	each clique in C_{new}
$C_{old,k}$	each clique in C_{old}
$\mathbf{X} \in \mathbb{R}^{M \times N}$	the feature matrix, where N columns correspond to N graphs and M rows correspond to M hashed clique patterns.
$\mathbf{x}^{(n)}$	each feature vector in X
$\mathbf{Y} \in \{0, 1\}^{L \times N}$	the label matrix, where L rows correspond to L classes
$\mathbf{W} \in \mathbb{R}^{L \times M}$	the weight matrix
α	the step size
$\{S_1, S_2, \ldots, S_n\}$	a sequential chunks
G_{test}	a test graph
$\mathbf{y}^{test}$	the class-label distribution vector of G_{test}

S_T $\{L_1, L_2, \ldots, L_K\}$	a testing chunk K predicted labels for G_{test}
d $x \in R^V$ S_* $\min(\pi_k(S_*))$ $\boldsymbol{P} \in \{1, -1\}^{K*V}$ $\xi: str \mapsto \{1, \ldots, K\}$ $\delta: str \mapsto \{1, -1\}$ S_n $h_n \in Z^K$ $S^{(R)} = \left\{S_1^{(r-1)}, \ldots, S_{\lvert S_*^{(r)} \rvert}^{(r-1)}\right\}$ R K $\mathrm{O}(\lvert S^{(R)} \rvert K^R)$ $\mathrm{O}(\prod_{r=1}^{R} \lvert S^{(R)} \rvert K)$	a text a feature vector, where V is the original dimensionality a set a min-hash of S_* the projection matrix, where V is the original dimensionality and K is the new, $K \ll V$ a random hash function a random hash function a set of terms (bag-of-words) a fingerprint *a nested set* the number of levels in a nested set the number of min-hash functions at each level the time complexity of the top-level recursion, where $\mathrm{O}(K^{R-1})$ is for the number of reorganized sets and $\mathrm{O}\lvert S^{(R)} \rvert K$ for K min-wise hashing procedures on $S^{(R)}$ the time complexity of the bottom-level of recursion, where $\mathrm{O}(\prod_{r=1}^{R} \lvert S^{(R)} \rvert)$ is for the number of the bottom-level sets and $\mathrm{O}\lvert S_*^{(1)} \rvert K$ for K min-wise hashing procedures on $S_*^{(1)}$.

CHAPTER 3
LITERATURE REVIEW

This chapter presents a discussion of relevant work in connection with this research. Section 3.1 reviews the research on hashing. Section 3.2 reviews the research on how existing hashing methods classify large-scale structured data. Section 2.3 reviews the emerging applications of the research. Our main objective is to (1) summarize and categorize hashing methods to provide a big picture for large-scale structured data classification; (2) summarize and categorize the large-scale structured data classification methods based on hashing; and (3) introduce the wide applications of the aforementioned methods.

3.1 HASHING

3.1.1 HASHING INTRODUCTION

In the application of databases, the "Key" is often used to uniquely identify a record in a table. It will be easier and more effective to find a record in the millions of records. Currently, with the increase of data volumes and complex data structure, large scale data are waiting to be processed. These data may be extremely larger than the available memory. In view of this situation, a fundamental research challenge appears: how to accurately and efficiently retrieve items from a large data collection. Naturally, less information we will try to use to represent these extremely large data. Under this requirement, the idea of hashing begins to be proposed. Hashing is the transformation of a set of data into shorter fixed-length values or bucket addresses that can represent the original data. This transformation can be achieved by a function h. With the help of a

hash function, the insertion, deletion, and lookup on the data can be done in almost constant time. However, some different data will be mapped into the same hashing value. The collision will be inevitable if there are more data than hashing values (See in Fig. 3.1). So the hashing methods firstly need to face two questions: 1) how to design a better hash function to minimize the collisions or increase the accuracy on the basis of high efficiency? 2) When the collisions occur, how to deal with them?

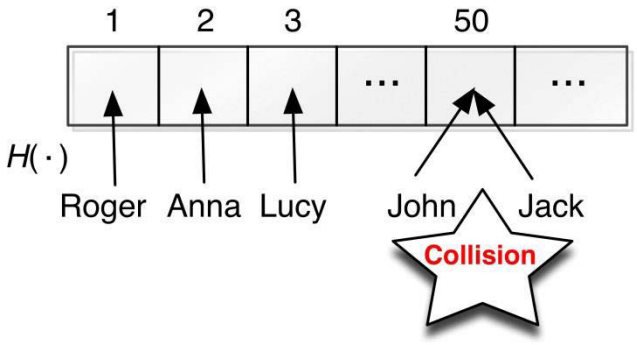


Figure 3.1: A basic hashing example

In the Fig 3.1, the black strings represent different names of people; the numbers represent the hash values of the grids; the $H(\cdot)$ represents the hash function that maps the black string into corresponding number; and the red "Collision" reflects the phenomenon that two different strings are mapped into the same grid. From Fig. 3.1, we can also see an example that how hashing is used for efficient data access. For the strings "Roger", "Anna", "Lucy", "John", "Jack", they are mapped by hash function $H(\cdot)$ into a set of numeric values, and there are corresponding hashing values for these records. When searching the record containing "Lucy", we just need to rehash this "Lucy", and this directly yields the hashing value "3" to this record, and then we finish searching. It is much more efficient than searching through all the records till the matching record is found.

Currently, the hashing methods mainly include data-independent hashing and data-dependent hashing. If the hashing function set is defined uniquely and independently from the data to be processed, we can classify it as a data-independent hashing method. Otherwise, it is classified as a data-dependent hashing method. Among the data-

independent hashing methods, one popular hashing method is Locality Sensitive Hashing (LSH) [1, 3]. With the help of the simple random projections, two objects within a smaller distance will be more likely to have the same hash code. This good property of LSH guarantees the collision probability between similar data. Another popular data-dependent hashing method is Spectral Hashing (SH) [36] which seeks balanced and uncorrelated compact binary codes of data-points for approximate nearest neighbour (ANN) search.

Many examples can reflect that the hashing technology can be extremely beneficial for text classification [19, 86], image search [11, 24, 72], and multimedia search [83~85]. In image search, the goal is to develop efficient image search and fast scene matching methods with very little memory. However, the actual size of image databases is often very large, and it is difficult or even impossible to save all this image information together in memory. On the other hand, the direct comparison between any two images is time-consuming. If we want to search a certain image in the whole image database, it will be extremely time-consuming. In such cases, it would be beneficial if we can compress the size of data and speed up the searching process. The hashing technology can learn binary-code representation for data in the hash code space and preserve the neighbourhood structure in the original feature space by mapping similar points in the original feature space to nearby binary codes in the hash code space. The compact representation in hashing can effectively save the storage and achieve fast query in large scale data.

3.1.2 HASHING OVERVIEW

3.1.2.1 KEY TERMINOLOGIES

Hashing Function: any function $h(\cdot)$ that can be used to map an arbitrary size of data to a fixed interval $[0, m]$. Given a dataset containing n data points $X = [x_1, x_2, \dots x_n] \in \mathbb{R}^D$, and a hashing function $h(\cdot)$, the $h(X) = [h(x_1), h(x_2), \dots, h(x_n)] \in [0, m]$ can be called the hash values, hash codes, hash sums, or simply hashes of data points $X = [x_1, x_2, \dots x_n] \in \mathbb{R}^D$.

One practical use is a data structure called a hash table, which has been widely used for rapid data lookup.

Hamming Distance: a set of all $2N$ binary strings of length N:

The Hamming distance between two equal length binary strings is the number of positions for which the bits are different.

$$H_{dis}(0011000, 0111001) = 2$$

$$H_{dis}(1011000, 1011000) = 0$$

Nearest Neighbour (NN): Given a set of n data points $X = [x_1, x_2, \dots x_n] \epsilon \mathbb{R}^D$, preprocess X and efficiently find a point in X closest to a query point $x \in X$.

The Nearest Neighbour Search (NNS) is an optimization problem to find the closest or the most similar data points. It can also be called a proximity search, similarity search or closest point search.

Approximate Nearest Neighbour (ANN): Given a set of n data points $X = [x_1, x_2, \dots x_n] \epsilon \mathbb{R}^D$, preprocess X and find a point $y \in X$ that is a ε-approximate nearest neighbor of the query point $x \in X$ in that for all $y' \in X$, the distance between y, x $d(y, x) \leq (1 + \varepsilon) d(y', x)$.

Curse of dimensionality: various phenomena that arise when analyzing and organizing data in high-dimensional spaces (such as hundreds or thousands of dimensions).

The curse of dimensionality does not occur in low-dimensional settings such as the two or three-dimensional space. The common problem in high-dimensional spaces is that when the dimensionality increases, the volume of the space increases very rapidly so that the available data becomes sparse.

3.1.2.2 THE FRAMEWORK FOR CATEGORIZING HASHING METHODS

We can see the overall classification for hashing methods from Fig. 3.2. According to the data dependency, we divide the hashing methods into two big classes: Data-independent hashing and Data-dependent hashing. In the Data-independent hashing methods, the methods that use a randomized process are classified into three kinds of classes: Random Hashing, Locality Sensitive Hashing, and Learning for Hashing. The methods that use deterministic structuring are classified into two kinds of classes: Tree or Space Filling Curves. In the data-dependent hashing methods, the hashing methods are classified into three kinds of classes: Unsupervised hashing, Semi-supervised hashing, and Supervised Hashing according to the properties of the training data.

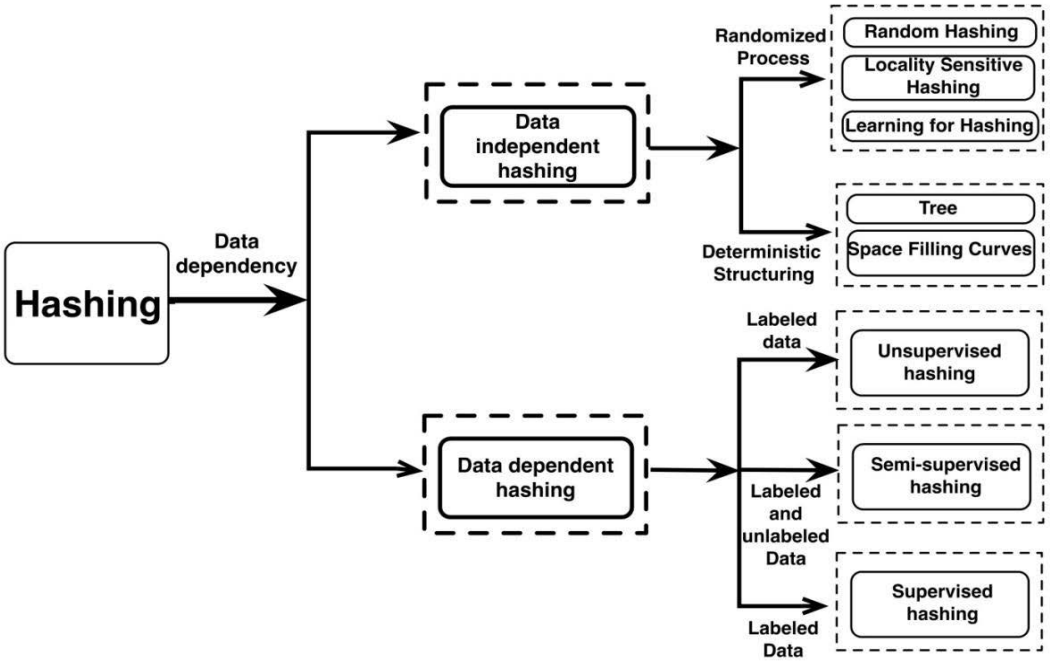


Figure 3.2: The Framework for Categorizing Hashing methods: the contents above the arrows represent the classification rules; the contents in the dotted box represent different classes of hashing methods

3.1.3 HASHING METHODS

Based on the projection dependency of hashing methods, we class all hashing methods across two main categories: Data-Independent Hashing functions and Data-Dependent Hashing functions.

3.1.3.1 DATA-INDEPENDENT HASHING METHODS

If the hashing function set is defined uniquely and independently from the data to be processed, we can classify it as a data-independent hashing method. The data independent hashing functions can be further divided into four classes based on the projection modes: Random Projection, Locality sensitive projection, Learning for hashing and Structured Projection. For locality sensitive hashing, we divide these hashing methods into three classes according to three different similarity metrics: Euclidean distance, Lp distance, Mahalanobis distance. The Min-hash is another kind of locality sensitive hashing. In the Structured Projection, the hashing methods mainly use tree and space filling curves to do the hashing. We can use Fig. 3.3 to summarize these data independent hashing functions.

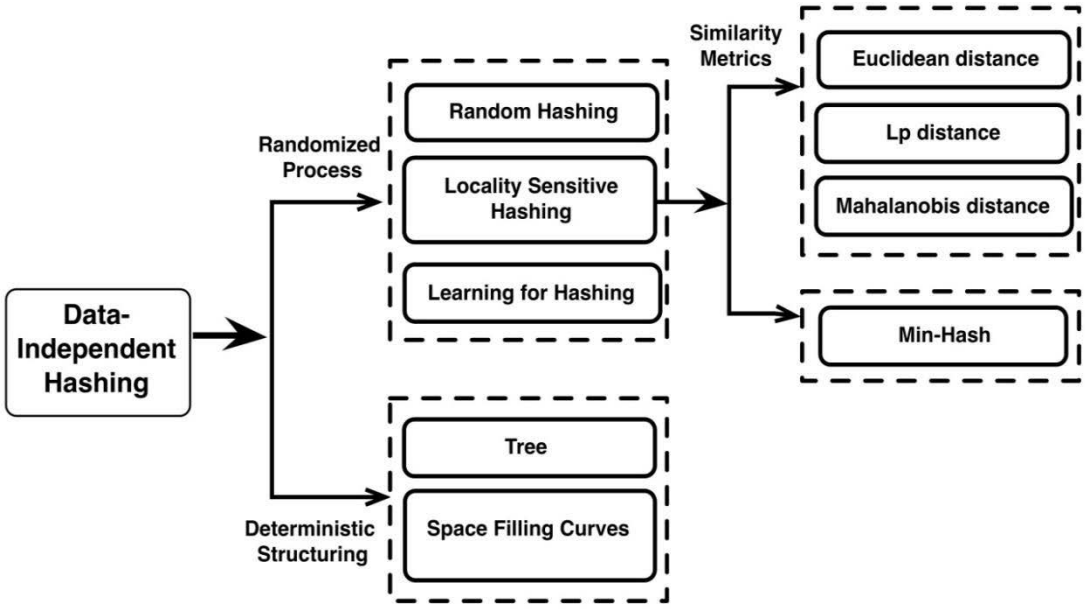


Figure 3.3: The classification of data-independent hashing methods: the contents above the arrows represent the classification rules; the contents in the dotted box represent different classes of hashing methods

We will first introduce the most simple projection method: Random Projection, and then advance to the locality sensitive projection to preserve the locality characteristics of the data. Next, we will introduce learning hashing projection and Structured Projection.

3.1.3.1.1 RANDOM HASHING

Random Projection hashing is a general data reduction technique, which randomly projects the original high-dimensional data into a lower-dimensional subspace. For example, we set the original data as d-dimensional data, and we can get the k-dimensional ($k \ll d$) subspace after random projection hashing.

According to the record in Kunth [26], the initial random projection hashing method was originated in 1953 by H. P. Luhn. The basic idea is to use a hash function, called a random function $h: U \to V$. In this way, a random value $h(x)$ can be generated in domain V (corresponding to the k-dimensional data) and associated with each data in the domain U (corresponding to the d-dimensional data). In this random projection, a random function requires $d \lg k$ bits to represent, which leads to the infeasibility of storing a randomly chosen function. With the deepening of the research, some researchers began to use fixed functions in the random projection. In [27], the authors proposed a universal hashing. The core idea was to choose hashing functions at random from a small family of functions, not all functions. This point guaranteed a provable performance and achieved feasible and succinct storage of hash functions. For example, the [6] chose the hash functions uniformly from some family F of hash functions:

$$\{x \to ((ax + b) mod\ p) mod\ v | 0 < a < p, 0 \le b < p\} \tag{3.1}$$

The whole family is defined by the parameters p and v, and a particular hashing function is defined by the parameters a and b.

In this universal hashing, each set of n elements (n: depending on the setting of a particular application) in U is uniformly projected to random and independent values, and the corresponding family F is n-*wise independent*. In [28], the authors proposed such function families where a random function can use $n \lg d$ bits of space to store. For a long while, the time complexity of all n-wise independent families to evaluate a hash function was $O(n)$ when the space usage was nontrivial. However, [29] made an important breakthrough in time complexity. In [29], the hash families were relatively small and highly independent which can be evaluated in constant time on a RAM.

However, the main drawback of random projection is high instability. In other words, different random hash functions may lead to totally different hashing values. On the other hand, if two elements have a one bit difference, they will be projected to two completely different random values. We can see that the pure random projection hashing cannot achieve good performance. In order to preserve the data characteristics in the original feature space, locality sensitive hashing was introduced.

3.1.3.1.2 LOCALITY SENSITIVE HASHING

The most widely known data-independent method with randomized projection is locality sensitive hashing (LSH) [1, 3]. This good property of LSH is that it guarantees the collision probability between similar data points. Despite its advantage, LSH still has an unavoidable disadvantage and that is the inefficiency of the hash codes. First, the random generation of hash functions and independency of data in LSH cannot guarantee the efficiency; second, it usually needs long codes in each hash table to guarantee an acceptable accuracy, which heavily increases the requirement of storage, especially for very large scale applications. So, many recent research works focus on how to generate short compact hash codes, and data dependent hashing methods have attracted attention recently.

With the help of the simple random projections, two objects within a smaller distance will be more likely to have the same hash code in the LSH. For similarity measures in LSH, [4, 6, 23, 24, 25] separately extended it to p-norm distances for $p \in (0, 2]$ [4], Mahalanobis distance [6], angular similarity [23, 25] and kernel similarity [24]. The basic idea of LSH is to choose a random hyperplane at the outset and use the hyperplane to hash input vectors. The hyperplanes are often used to partition the data points into two sets in the original data space or a kernel space, and two different binary codes are assigned based on the set each data point is assigned to. For the hyperplane, in order to more optimally allocate a variable number of bits per LSH hyperplane, two papers [20, 21] successively proposed dubbed Neighbourhood Preserving Quantization (NPQ) [20] and dubbed Variable Bit Quantisation (VBQ) [21]. The NPQ assigns multiple bits per hyperplane based upon adaptively learned thresholds, and the VBQ provides a data-

driven non-uniform bit allocation across hyperplanes. Based on the randomized projection of LSH, [22] proposed a scheme named Distribution-Aware LSH (DALSH) which generated a series of data-adaptive projections to address the problem of a lack of adaptation to real data.

Another LSH-related hashing technique is named Min-Hash. The full name of Min-Hash is the min-wise independent permutations locality sensitive hashing scheme. It is an important hashing technique to estimate the similarity between two sets. Here, we use an example to demonstrate the work process of Min-wise hashing in Fig. 3.4:

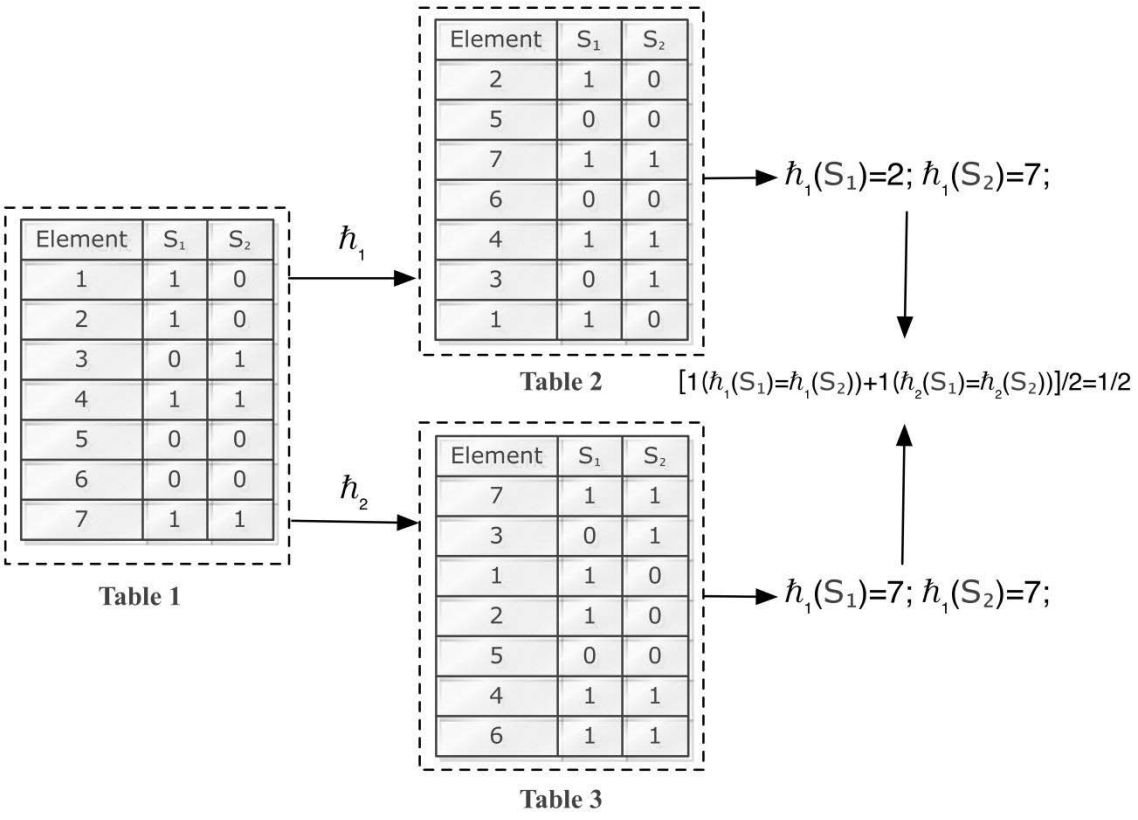


Figure 3.4: An example of the Min-Hash

In the last two columns of the tables of Fig 3.4, the "1" or "0" represent that whether the corresponding element is in the set (S_1 or S_2) or not, and the $\hbar_1$ and $\hbar_2$ represent two different hash functions. We set S_1={1,2,4,7}, S_2={3,4,7}, and two independent random element permutations $\hbar_1$=[2,5,7,6,4,3,1] and $\hbar_2$ =[7,3,1,2,5,4,6]. For S_1, the minimum $\hbar_1$ hash value is 2, and the minimum $\hbar_2$ hash value is 1. For S_2, the minimum $\hbar_1$hash

value is 7, and the minimum $\hbar_2$hash value is 1. Lastly we can get that the similarity between S_1, S_2: $\hat{j}(S_1, S_2) = \frac{\sum_{k=1}^{2} 1(\hbar_k(S_1)=\hbar_k(S_2))}{2} = \frac{1}{2}$.

This hashing scheme was initially introduced in [9] and firstly used in [10] to detect duplicate web pages and eliminate them from search results in the AltaVista search engine. It has also been applied in large-scale clustering documents [9], near duplicate Image detection [11] and large-scale text classification [19] which proposed a Recursive Min-wise Hashing (RMH) to preserve the context information. In order to save storage space, [13, 14, 15, 16] developed this min-hash technique to a b-bit Min-wise hashing by changing the traditional 64 bits used to store each hashed value in Min-wise hashing methods.

Another few related random projection hashing techniques include Shift Invariant Kernel based Hashing (SIKH) [6, 12], Nested Subtree Hashing (NSH) [17] and Discriminative Clique Hashing (DICH) [18]. In [6, 12], based on random projections, authors introduced a simple distribution-free encoding scheme, which could relate the expected Hamming distance between the binary codes of two vectors to the value of a shift-invariant kernel between the vectors. Then, as many data mining applications involve networked data with dynamically increasing volumes, [17, 18] respectively proposed a new hashing scheme to address the problem of large-scale graph classification over streams. In [17], the authors proposed the Nested Subtree Hashing (NSH) method to recursively project the multi-resolution subtree patterns of different chunks onto a set of common low-dimensional feature spaces. In [18], two random hashing schemes were employed in Discriminative Clique Hashing (DICH) to speed up the discriminative clique-pattern mining process and address the unlimitedly clique-pattern expanding problem.

3.1.3.1.3 LEARNING FOR HASHING

In [7] the researchers proposed an algorithm named BoostMap which is a data-independent machine learning method for the construction of Euclidean embeddings. It is an efficient approximate nearest-neighbour method, and can be applied to arbitrary

distance measure, metric or nonmetric. Before introducing this BoostMap, we first introduce some basic methods for constructing Euclidean embeddings, such as Lipschitz embeddings [30], Bourgain embeddings [30, 31], FastMap [32], and MetricMap [33].

The basic idea of Lipschitz embeddings is to embed metric spaces into other ones with low distortion. In Lipschitz embeddings, an object $x \in X$ (a space) is transformed into an n-dimensional vector $V = (v_1, v_2, \dots v_n)$ such that each element v_i corresponds to the distance of object $o \in X$ to a predefined reference set [30]. The Bourgain embeddings are a special type of Lipschitz embeddings.

The key point of the learning process in BoostMap is to see the embeddings as classifiers used to estimate the distance of any three data objects, and to use AdaBoost [34] to combine all previous lower-dimensional embeddings into one higher-dimensional embeddings for higher accurate similarity rankings. The process is: after identifying a large family of $1D$ embeddings P based on a reference object or a pair of pivot objects, we can see each $1D$ embedding P as a continuous output binary classifier and a *weak classifier* [34]. Each weak classifier estimates, for triples (q, x_1, x_2) of objects in X, if q is closer to x_1 or x_2. The BoostMap will use AdaBoost [34] to combine many $1D$ embeddings P into a multidimensional embedding that behaves as a *strong classifier* that has relatively higher accuracy than a *weak classifier*. The BoostMap makes full use of the advantage of machine learning techniques to assemble a higher-accuracy embedding from many one-dimensional embeddings.

3.1.3.1.4 STRUCTURED PROJECTION

Although there are many effective hashing methods for low-dimensional spaces, their performance may degrade as the number of dimensions increases which is the *dimensional curse* phenomenon. For a similarity search in High-Dimensional Vector Spaces (HDVSs), the conventional approach to do similarity searches in HDVSs is to use a multidimensional index structure which needs to do a data space partition.

In the structured projection hashing methods, the data space will be divided along the defined or predefined lines regardless of data features, and the lines are defined or predefined by the different selected structures.

Based on a deterministic structuring, [2, 5, 8] separately proposed data independent hashing schemes with structured projection, including tree [2] and space filling curves [5, 8].

In [2], the authors studied the impact of dimensionality on the nearest-neighbour similarity-search in high-dimensional vector space (HDVSs), and showed that any partitioning scheme and clustering technique must degenerate to a sequential scan through all their blocks if the number of dimensions is sufficiently large. In [5, 8], the authors both studied the content-based copy identification by space filling curves projection. In [5] the authors mainly proposed a novel strategy dedicated to pseudo-invariant feature retrieval more specifically for content-based copy identification. Furthermore, this paper adopted the Hibert curve as the line of projection and directly mapped the approximate search range onto a Hilbert space-filling curve in order to establish an efficient access to the database. The advantage of the Hilbert curve is that it can guarantee that two cells that are neighbours in the index are also neighbours in the description space [35]. For the Hilbert curve, the disadvantage is that it will be difficult to compute the key in the index starting from the position in description space for high-dimensional spaces and higher-order partitioning. In order to simplify the computation of the keys (cell addresses in the index) and to link it more strongly with a component-wise search process, [8] replace the Hilbert curve by the Z space-filling curve and hierarchically partition the description space into hyper-rectangular cells following the Z-curve.

3.1.3.2 DATA-DEPENDENT HASHING METHODS

In the data-dependent hashing method, the hashing function family is defined uniquely only for a given training dataset and the hash functions usually involve similarity comparisons with some features of the training dataset. The objective of these methods is to closely fit the data distribution in the feature space in order to achieve a better selectivity while preserving locality as much as possible.

We can use Fig. 3.5 to summarize the data dependent hashing functions:

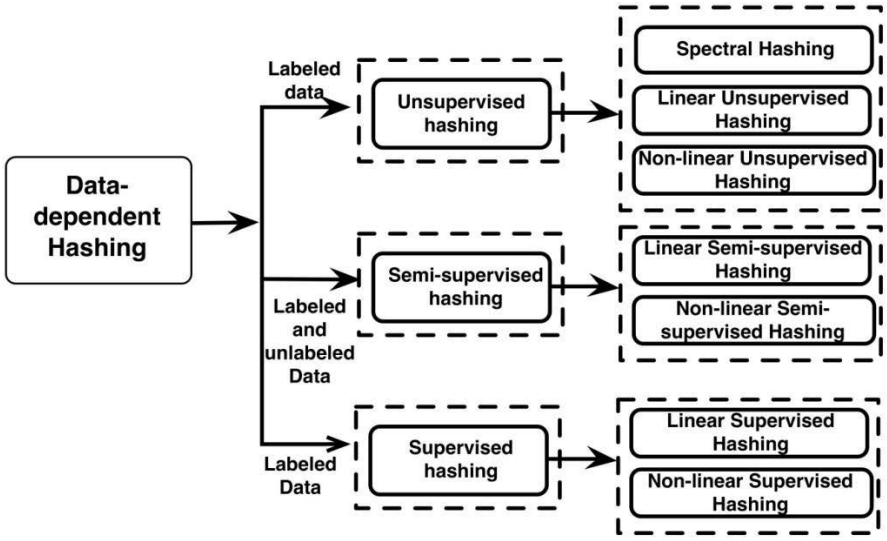


Figure 3.5: The classification for data-dependent hashing methods: the contents above the arrows represent the classification rules; the contents in the dotted box represent different classes of hashing methods

3.1.3.2.1 UNSUPERVISED HASHING

In the unsupervised hashing methods, the pairwise labels are not available. Unsupervised hashing methods use just the unlabelled data to generate binary codes for the given points and try to preserve the similarity in the original feature space.

According to the three forms of hashing functions (Eigenfunctions, Linear functions, and Non-Linear functions), we divide the unsupervised hashing into three corresponding types: Spectral Hashing, Linear Hashing, and Non-Linear Hashing.

(1) Spectral Hashing:

The most popular data-dependent unsupervised hashing is Spectral Hashing (SH) [36]. The SH discusses the problem of learning a code for semantic hashing [38] which designs compact binary codes for a large number of documents so that semantically similar documents are mapped to similar codes within a short Hamming distance. In [36], the authors defined a hard criterion for a good code that is related to graph partitioning and used a spectral relaxation to obtain an eigenvector solution.

(2) Linear Unsupervised Hashing:

Many unsupervised hashing methods use linear functions to do the hashing. There are some related learning-based hashing methods [37, 39~45, 48~52]. Most of these hashing algorithms focus on exploiting the spectral properties of the data affinity matrix for binary coding. Among them, the Anchor Graph Hashing (AGH) [37] is more popular. The AGH is a graph-based hashing method which automatically discovers the neighbourhood structure inherent in the data to learn appropriate compact codes, and a scalable graph-based unsupervised hashing approach which considers the underlying manifold structure of the data to search for the semantic nearest neighbour.

In order to achieve a satisfactory performance, many existing hashing methods use a large number of hash tables (long codewords), and the space cost has become a problem. Some papers [58, 59, 60] had proposed corresponding methods to address this problem. In [58], the authors consider a hardware-friendly scheme for Minimal Perfect Hashing (MPH) via counting Bloom filters to reduce the number of memory accesses to just 1 and also to be space-efficient. In order to perform cost effective and exact pattern matching, the authors in [59] proposed HashMem architecture to combine hashing and memories by using hashing to generate a distinct address for each candidate pattern stored in memory. The authors in [60] developed a hashing algorithm Compressed Hashing (CH) for high dimensional nearest neighbour search by combining the techniques of sparse coding and compressed sensing.

Other important linear unsupervised hashing methods include ANN search algorithm Product Quantization (PQ) [53], Angular Quantization-based Binary Coding (AQBC) [54] for high-dimensional non-negative data that arises in vision and text applications , Spherical Hashing [55] to map more spatially coherent data points into a binary code compared to hyperplane-based hashing functions, Isotropic Hashing (IsoHash) [56] firstly to learn projection functions which could produce projected dimensions with isotropic variances (equal variances), Manhattan hashing (MH) [57] based on Manhattan distance to deal with the destruction of the neighbourhood structure in the original feature in Hamming distance based hashing, Predictable Dual-View Hashing (PDH) [61] to embed proximity of data samples in the original spaces, and Inductive Manifold Hashing (IMH) [62] to connect manifold learning methods and hash function learning. Most

recently, [63] proposed Locally Linear Hashing (LLH) to preserve the locally linear manifold structures of high-dimensional data in a low-dimensional Hamming space. Another latest unsupervised hashing method Topology Preserving Hashing (TPH) was proposed in [64] to preserve neighbourhood relationships and relative neighbourhood proximities.

(3) Non-Linear Unsupervised Hashing:

In order to accommodate arbitrary kernel functions, some papers [24, 43, 46, 47] proposed unsupervised kernel-based hashing method. In [24], the authors widened the accessibility of LSH to generic kernel space and proposed Kernelized LSH (KLSH). The main idea of the KLSH is to construct a random hyperplane hash function in kernel space based on a central limit theorem. According to the central limit theorem, under very mild conditions, the mean of a set of data objects from some underlying distribution will better follow Gaussian distribution in the limit as the number of data objects in the set increase. In this central limit theorem, an approximate random vector will be computed by using data items from the database. Once the random hyperplane hash function is constructed, the KLSH computes a small set of candidate approximate nearest neighbors by the method of Charikar, sorts them to yield a list of hashed nearest neighbors by the kernel function and then uses standard LSH techniques to retrieve nearest neighbors of a query to the database in sublinear time. In KLSH, there were no assumptions about the data distribution or input which could help KLSH directly suitable for image search and other domains.

3.1.3.2.2 SEMI-SUPERVISED HASHING

In semi-supervised hashing, both labeled data and unlabeled data are used to train the model. Representative semi-supervised methods include Semi-Supervised Hashing (SSH) [65, 66], LAbel-regularized Max-margin Partition (LAMP) algorithm [67], Semi-Supervised Discriminant Hashing (SSDH) [68], Bootstrap Sequential Projection Learning for Semi-supervised Nonlinear Hashing (Bootstrap-NSPLH) [69], and Semi-Supervised Topology Preserving Hashing (STPH) [64]. Among these hashing methods, the Semi-Supervised Hashing (SSH) is most popular.

3.1.3.2.3 SUPERVISED HASHING

In supervised hashing, only labelled data, such as similar or dissimilar data, are used to train the model. The goal of supervised hashing is to respect label-based similarity or semantic similarity.

Representative supervised methods include Boosting Similarity Sensitive Coding (BoostSSC) [70], Boltzmann machine based hashing (RBMs) [71], Binary Reconstructive Embedding (BRE) [73], Minimal Loss Hashing (MLH) [74], Kernel-based Supervised Hashing (KSH) [75], and Linear Discriminant Analysis based Hashing (LDAHash) [76]. Most recently, some new supervised methods was proposed which include Similarity Preserving Hashing (SPH) [77], Two-Step Hashing (TSH) [78], Multimodal Similarity-Preserving Hashing (MSPH) [79], Semantic Correlation Maximization (SCM) [80], Latent Factor Hashing (LFH) [81] and FastHash [82].

Overall, compared with unsupervised hashing methods, the main advantages of these supervised hashing methods are the flexibility and adaptability for real-world applications. However, the training efficiency is still a big problem.

3.2 LARGE-SCALE STRUCTURED DATA CLASSIFICATION BASED ON HASHING

3.2.1 GRAPH STREAM CLASSIFICATION BASED ON HASHING

The surge of real-world networked data, such as chemical compounds, biological data, XML documents, and program flows, has led to the rise of graph mining research [87]. Graph classification is an important graph mining task that aims to learn a discriminative model from training examples to predict class labels of test examples,

where both training and test examples are graphs. The essential challenge for graph classification is to extract features from graphs to represent them in feature vectors to facilitate classifier training within a generic machine learning framework. A variety of studies on substructure extraction (e.g., walks [88, 97], paths [89], subtrees [90, 91, 98], and subgraphs [99]) for describing graphs have been proposed in the past decade. However, most of them consider the learning problem of graph classification in batch mode (where all graph data are available for training), which hinders their applicability to large-scale and stream scenarios.

In fact, dynamic networked data are often presented with increasing volumes and change over time in many real-world scenarios. For example, a social network is made up of a population of individuals, where the interactions among them keep generating, disappearing, and changing over time. A transportation network is a complex network made up of numerous interconnected routes, where the traffic flows over them are generated dynamically over time. These evolving networked data can be defined as graph streams. Graph streams not only inherit the features of static graphs but also possess special characteristics such as frequent update and necessary real-time response [102]. To solve these emerging problems, graph stream mining has recently attracted increasing research interest [92], [93], [94], [95], [17], [18].

However, graph stream classification on a complex network with massive nodes is by no means an easy problem because of the following challenges:

- **Expanding Feature Space**: Graph stream is defined on a massive universe of nodes. The continuously received graphs in the stream would lead to an increasing number of subgraph patterns. Due to the "one-pass" nature of the graph stream, we cannot enumerate all the subgraph-patterns in a pre-scan to construct the feature space for all the graphs. Thus we need to design a projection that can map arbitrary subgraph patterns to a fixed-size compatible feature space for all the graphs in the stream.

- **Increasing Graph Volumes**: The volume of graph data are continuously growing with a high speed. Due to the "real-time" requirement of graph stream

classification, we cannot employ any existing subgraph detection method which is designed for off-line accurate subgraph mining. Thus we need to develop an approximate subgraph feature extraction method that is fast enough to tackle fast increasing graph volumes.

In [95], [18], the authors have investigated the graph stream classification problem. Both of them employ hashing techniques to sketch the graph stream for saving computational cost and controlling the size of the subgraph-pattern set. The most relevant work to this topic is [95], which also considers graph stream classification on a complex network. It employs a 2-D hashing scheme to construct an "in-memory" summary for the sequentially received graphs. The first random-hash scheme is used to reduce the size of the edge set. The second min-hash scheme is used to dynamically update a number of hash-codes (i.e., corresponding to random sorting samples), which is able to summarize the frequent patterns of co-occurrence edges in the graph stream observed thus far. Finally, a simple heuristic is used to select a set of most discriminative frequent patterns to build a rule-based classifier. Although [95] has exhibited promising performance on graph stream classification, it has two inherent limitations: (1) The selected subgraph-patterns are composed with disconnected edges, which may have less discriminative capability than connected subgraph-patterns due to a lack of semantic meaning. (2) The computational cost is high because an additional frequent pattern mining procedure is required to perform on the summary table which comprises massive transactions. Our previous work, DIscriminative Clique Hashing (DICH) [18], has addressed these limitations to a certain extent by employing a fast clique detection algorithm from hashed graphs. However, DICH adopts a majority voting classifier whose performance could degrade as the number of classes becomes large. Another critical shortcoming is that the DICH cannot adapt to the concept drifting in graph stream classification. To address the problem of the concept drifting, an instance weighting mechanism has been proposed in the gSLU [103] to adjust the subgraph feature selection module for emerging concept drifting graphs. However, this instance weighting mechanism may be too sensitive to better adapt to the concept drifting.

3.2.2 TEXT CLASSIFICATION BASED ON HASHING

There have been a number of approximate algorithms for big data similarity computation. Since many high-dimensional data can be represented as bags of words, min-wise hashing [10] has been naturally applied to them for fast approximating set similarities without scanning and comparing the complete sets. Recently, [114] further improves the efficiency of min-wise hashing by storing only the lowest b bits of each hashed value. Random projection [112, 107] was proposed to randomly project high-dimensional data onto low-dimensional spaces. For sketching streaming data, count-min sketch [110] was developed to estimate feature occurrences. Recently, feature hashing [115, 116] was employed to estimate inner products of high-dimensional feature vectors. All these approximate algorithms have been found very effective in certain big data problems.

Massive text mining is the most fundamental and essential technology in the era of big data to support various services in almost every field. Massive data mining techniques, including learning to hash [36, 73, 37] and approximate algorithms, are developed rapidly under this circumstance. This work focuses on approximate algorithms, in particular, random hashing techniques, to improve the text similarity estimation quality. The classical approach is a family of locality-sensitive hashing (LSH) algorithms for approximating nearest neighbours in high dimensions [108, 114], in which the most two typical algorithms are min-wise hashing [10] and random projection [112, 107] aforementioned in Section 1. These LSH algorithms have been extensively applied to massive text mining applications as the state-of-the-art techniques. Recently, [15, 16] further improves the efficiency of min-wise hashing by storing only the lowest b bits of each hashed value. Besides, feature hashing [115, 116] was also proposed to estimate inner products of high-dimensional feature vectors. The proposed RMH algorithm for context-preserving hashing can be viewed as a generalization of the standard min-wise hashing scheme [10] for nested sets.

Since we will takes into account the semantic hierarchy in this thesis, we also review some hashing techniques considering structural information. A 2-D hashing scheme is employed in [95] to construct an “in-memory” summary of sequentially received graphs, where the first random-hash scheme is used to reduce the size of the edge set

while the second min-hash scheme is used to summarize the co-occurrence edges in the graph stream observed thus far. Recently, [18] proposes to detect clique patterns from a compressed network obtained through random edge hashing, and the detected clique patterns are further hashed to a fixed-size feature space [116]. The nested subtree hash kernel [17] hashes unlimited node labels into a certain amount and the feature space of the resulting subtree patterns can be constrained. All these methods are aimed to deal with graphs and cannot be applied to sketching nested sets.

We observe that [111] also proposes a min-hash algorithm for hierarchical data objects. However, [111] has two significant differences from the proposed RMH algorithm: 1) Its similarity is based on an assumption that each set is a weighted set and the weights sum to one; while our similarity is directly derived from the Jaccard similarity in the recursive case. 2) It simply views lower-level signatures as an element of the current-level set for min-wise hashing, which will totally discard the similarity information between lower-level sets; while our RMH algorithm reorganizes lower-level min-hashes into a number of sampling-sets for min-wise hashing, which can propagate lower-level similarities to higher levels in probability. This method is mathematically incompatible to the considered problem in this paper; however we did try it in our experiments and its results are only slightly above the random guess. Thus we did not include it in the experimental results.

However, all the aforementioned approximate algorithms are based on the bag-of-words representation for its exchangeability that can facilitate random projection and hashing. A limitation of such *flat-set* representation is that context information and semantic hierarchy may be lost. Thus, a more expressive bag-of-words representation needs to be explored to relieve this problem.

CHAPTER 4
DISCRIMINATIVE CLIQUE HASHING FOR FAST GRAPH STREAM CLASSIFICATION

4.1 INTRODUCTION

In this chapter, we propose to classify large-scale graph streams using hashing methods.

The emergence of complex networks has led to a surge of research in graph data mining [87]. Graph classification is an important graph data mining task that aims to learn a discriminative model from training examples to predict class labels of test examples, where both training and test examples are graphs. Many real-world applications involve graph-represented data, such as chemical compounds, XML documents, and program flows. The essential challenge for graph classification is to extract features from graphs and represent graph data in instance-feature format to support model training. A variety of studies on substructure extraction (e.g., walks [88, 89], paths [90], and subtrees [91, 92]) for describing graphs have been proposed in the past decade. However, most of them only consider the learning problem of graph classification in batch mode (all data are available for training), which limits their applicability to large-scale and stream scenarios.

Due to the streaming nature of many real-world complex networks, such as social networks and sensor networks, graph stream classification has recently attracted

increasing research interest [93-96]. Graph stream classification is defined on a complex network which comprises a massive universe of nodes, where the stream of graphs is represented as sets of edges on the underlying network. For example, co-authorships of research works continuously form graphs on a coauthor network (e.g., DBLP), dynamic communities of interest continuously form graphs on a social network (e.g., Facebook), and traffic flows continuously form graphs on a transportation network. Graph stream classification on a complex network with massive nodes is challenging, because

- **Subgraph feature generation:** Graph stream is defined on a massive universe of nodes, enumerating subgraph-patterns from such a large node set as features is time consuming and memory intensive. We need fast and inexpensive feature generation method for graph stream classification.

- **Increasing stream volumes:** The volumes of graph data are continuously growing, so graph streams can usually be accessed only once. Graph stream classification must be able to tackle dynamically increasing graph volumes and generate discriminative model with high speed.

- **Changing feature distributions:** The marginal distributions of subgraph-patterns (features) may continuously change over the graph stream (i.e., the concept-drift problem [96]), a dynamic updating scheme is required to update the discriminative model.

Few studies have investigated the graph stream classification problem. To the best of our knowledge, only two works [95, 97] may be applied to the considered problem. Both of them employ hashing techniques to sketch the graph stream for saving computational cost and controlling the size of the subgraph-pattern set. In [97], the authors proposed a hash kernel to project arbitrary graphs onto a compatible feature space for similarity computing, but it can only be applied to node-attributed graphs. Recently, Aggarwal [95] proposed a 2-D hashing scheme to construct an "in-memory" summary for sequentially presented graphs and used a simple heuristic to select a set of most discriminative frequent patterns to build a rule-based classifier. Although [95] has exhibited promising performance on graph stream classification, there are two inherent

limitations: (1) The selected subgraph-patterns are composed with disconnected edges, which may have less discriminative capability than connected subgraph-patterns due to a lack of semantic meaning; (2) The computational cost is high because an additional frequent pattern mining procedure is required to perform on the summary table which comprises massive transactions.

In this chapter, we propose a fast graph stream classification method using DIscriminative Clique Hashing (DICH) to address the aforementioned challenges. The main idea is to decompose a compressed graph into a number of cliques (fully connected subgraphs) to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are employed to compress the original edge set of the graph stream and map the unlimitedly increasing clique-patterns onto a fixed-size feature space, respectively. The hashed cliques are then used to update an "in-memory" fixed-size pattern-class table, which will be finally used to generate a rule-based classifier. Since DICH adopts connected subgraphs as features and needs no additional frequent pattern mining procedure, it can achieve very fast training speed for graph stream classification. The experimental results on two real-world graph stream data sets clearly show that DICH outperforms the compared state-of-the-art [95] in both classification accuracy and training efficiency.

The remainder of this chapter is organized as follows. Section 4.2 defines the form of problem and introduces the system overview. The DICH algorithm details are presented in Section 4.3. A case study by use of the proposed DICH algorithm to show its effectiveness is presented in Section 4.4. Finally, the summary is given in Section 4.5.

4.2 DEFINITIONS & METHOD FRAMEWORK

4.2.1 PROBLEM DEFINITION

Suppose there is a complex network which comprises a massive universe of nodes. The dynamically generated edges form a sequence of graphs, and those graphs are independent. The edges connecting these nodes are denoted by the edge set ε. In

particular, the edge set of G_i are denoted by $\varepsilon_i = \{e_1, \cdots, e_E\} \subset \varepsilon$, where e denotes the number of edges in G_i. The stream of graphs $\{\cdots, G_{n-1}, G_n, G_{n+1}, \cdots\}$ are presented continuously as subsets of ε, where the subscript i denotes the receiving order in the graph stream. Each graph G_i has a class label $l_i \in \{1, \cdots, L\}$. We assume G_i is received in the form $\langle i, e_1, \cdots, e_E, l_i \rangle$. In this chapter, we assume that each edge has a default weight 1 for simplicity. The underlying graph stream can only be accessed once and our **goal** is to learn a discriminative model from $\{\cdots, G_{n-1}, G_n, G_{n+1}, \cdots\}$ at a high efficiency to accurately predict the class label of a test graph G_{test} in the future graph stream.

4.2.2 METHOD FRAMEWORK

The corresponding framework, illustrated in Fig. 4.1, comprises three modules. The graphs in the stream are received and processed one by one. The first module is for clique detection from each graph in the stream. The incoming edges of G_i are first randomly hashed to a compressed edge set and then we adopt a fast algorithm to decompose the compressed graph into a number of cliques (fully connected subgraphs) as the features of G_i. Since the number of clique-patterns will unlimitedly increase as new graphs are fed in, the underlying feature space will keep expanding accordingly. Thus, in the second module, a clique hashing scheme is performed to map the unlimitedly emerged clique-patterns onto a fixed-size clique-pattern set. In the last module, an "in-memory" fixed-size pattern-class table is updated using the clique-pattern and class label information of G_i; and a rule-based classifier is constructed based on the pattern-class table by identifying frequent and discriminative clique-patterns associated to each class. To test a graph G_{test} in the future graph stream, G_{test} is processed in the first two modules and the obtained hashed clique-patterns are input to the rule-based classifier for class label prediction. The detailed approaches to the three modules are described in the following section.

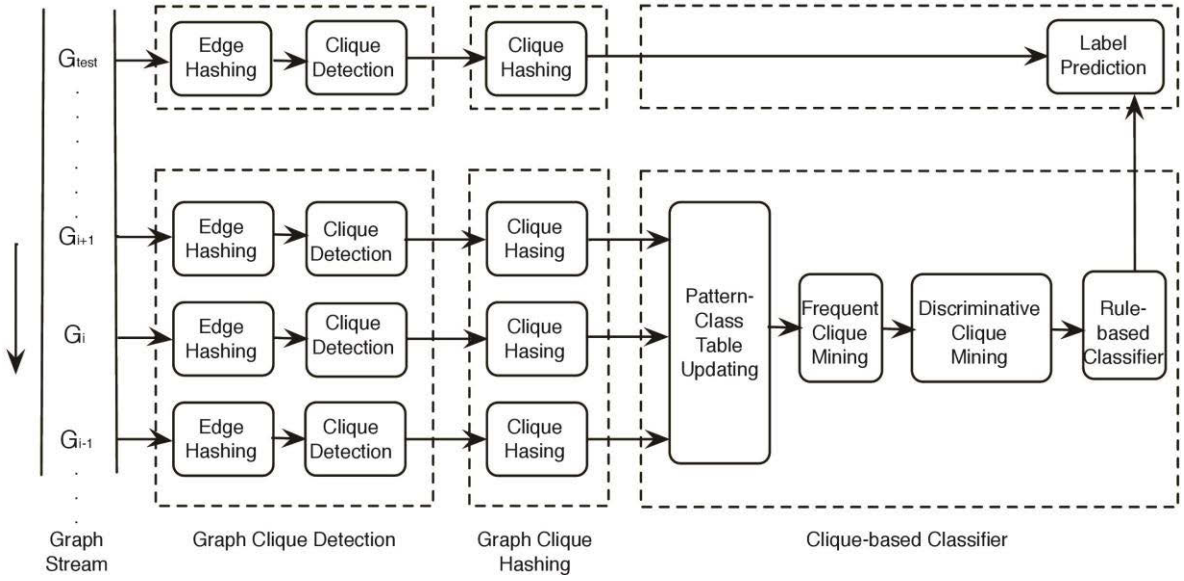


Figure 4.1: The framework of DICH for graph stream classification

4.3 DICH: DISCRIMINATIVE CLIQUE HASHING

4.3.1 GRAPH CLIQUE DETECTION

As introduced in Section 4.2, a graph in the stream is an edge subset of the complex network. Since the edge set of the complex network can be extremely large, it may be infeasible to detect cliques from such a large network in real time. Thus, it is necessary to sketch the graph stream to accelerate the clique detection (feature extraction) process. To this end, we first employ a random hashing scheme to compress each graph G_n on the large network to a small graph $\overline{G}_n$:

$$\bar{G}_n := Graph - Hash(G_n, \bar{V}) \tag{4.1}$$

where $\bar{V}$ denotes the desired number of vertices in the compressed graph $\bar{G}_n$. The graph hashing operation includes two steps: First, we use a random hash function

$$\hbar: \{1, \cdots, V\} \mapsto \{1, \cdots, \bar{V}\} \tag{4.2}$$

to map the indices of the vertices in G_n to $\{1, \cdots, \bar{V}\}$ as the set of vertices in $\bar{G}_n$. Second, for each edge in $\overline{G}_n$, its weight $\overline{w}_{ij}$ is calculated as follows:

$$\overline{w}_{ij} = \sum_{i=\hbar(p), j=\hbar(q)} w_{pq} \tag{4.3}$$

The above graph compression Eq. (4.1) using random hashing has an obvious property: if two graphs G and G' on the original network have a same subgraph g, the compressed graphs $\bar{G}$ and $\overline{\mathrm{G}}'$ also have a same subgraph $\overline{g}$. This property implies that, if two original graphs are similar in terms of subgraphs, the compressed graphs are also similar. We may have collisions that different subgraph patterns in the original graphs get the same subgraph pattern in the compressed graphs. But the probabilities of such cases are very low since the collided edges in the compressed graphs are unlikely to form a connected graph-recall that we only extract cliques as features which are fully connected subgraphs. The leftmost two columns in Fig. 4.2 illustrate this operation.

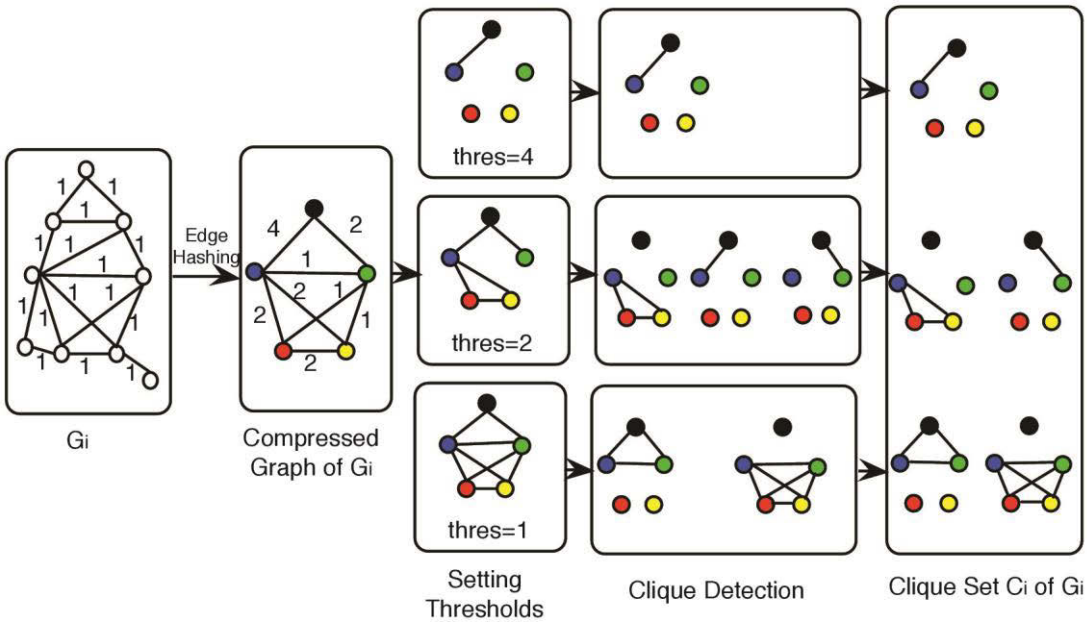


Figure 4.2: Clique detection in a compressed graph

After obtaining the compressed graph $\bar{G}_n$, we will employ a fast algorithm to detect cliques in $\overline{G}_n$. We adapt the graphlet basis estimation algorithm used in [100] to this end. The first step for clique detection is to threshold the compressed graph $\bar{G}_n$ at a number of weight levels in descending order t, say $\{max(\bar{G}_n), \cdots, min(\bar{G}_n)\}$, where $max(\bar{G}_n)$ and

$min(\bar{G}_n)$ denote the largest and the smallest edge weights in $\bar{G}_n$, respectively. We define this graph thresholding operation as

$$\bar{G}_n{}^{(t)} := 1(\bar{G}_n) \geq t \tag{4.4}$$

where $1(\cdot)$ denotes an indicator function, which sets the weight of an edge in $\bar{G}_n{}^{(t)}$ to be 1, i.e., $\bar{w}_{ij}{}^{t} = 1$, if $\bar{w}_{ij} > t$. We use the Bron-Kerbosch algorithm [101] to identify all the cliques from $\bar{G}_n{}^{(t)}$ at each threshold. The Bron-Kerbosch is an algorithm for finding maximal cliques in an undirected graph. The union set of the cliques found in $\left\{\bar{G}_n{}^{(t)}\right\}_{t=min(\bar{G}_n)}^{max(\bar{G}_n)}$ is represented as the clique set for G_n. This procedure is detailed in Algorithm 1.

Algorithm 1 Graph Clique Detection

Input: G_n: a graph in the graph stream; $\bar{V}$: given number of vertices in $\bar{G}_n$

Output: C_n: the clique set detected from $\bar{G}_n$

1: $\bar{G}_n = Graph - Hash(G_n, \bar{V})$;

2: $C_n = \emptyset$;

3: **for** $t = max(\bar{G}_n) : 1 : min(\bar{G}_n)$

4: $\quad \bar{G}_n{}^{(t)} := 1(\bar{G}_n) \geq t$;

5: $\quad C_n^{(t)} := Bron - Kerbosch\left(\bar{G}_n{}^{(t)}\right)$;

6: $\quad C_n := C_n \cup C_n^{(t)}$

7: **end for**

An example of clique detection is illustrated in Fig 4.2. After graph hashing, we obtain the compressed graph of G_n (the 2nd column). Then, four weight thresholds $\{4, 3, 2, 1\}$ are set to generate three graphs $\left\{\bar{G}_n{}^{(1)}, \bar{G}_n{}^{(2)}, \bar{G}_n{}^{(4)}\right\}$ (the 3rd column). Note that $\bar{G}_n{}^{(3)}$ is an empty graph which is not shown. The Bron-Kerbosch algorithm is applied to detect a set of cliques from each graph (the 4th column). Finally, the cliques detected at

the all weight thresholds are merged to form the clique set C_n for G_n as its feature representation (the 5th column).

4.3.2 GRAPH CLIQUE HASHING

As aforementioned, the cliques extracted from each graph are used to represent its features. To learn a classifier from the graph stream, it is required to make the features of all graphs be in the same feature space. In other words, we should count the occurrences of the same set of clique-patterns in all graphs in the stream. Since the number of clique-patterns will increase as new graphs are continuously fed in, the induced feature space will keep expanding accordingly. To address this problem, we adopt a feature hashing scheme to randomly map the unlimitedly emerged clique-patterns onto a fixed-size set. In particular, we use an "in-memory" $P \times M$ pattern-class table Δ, which can be dynamically updated, to count clique-pattern and class label information from the graph stream. In Δ, P rows correspond to the indices of hashed clique-patterns while M columns correspond to all the classes of the graphs.

Given G_i in the graph stream, we first use Algorithm 1 to collect the clique set C_i. Then, for each clique in C_i, say $C_{i,j}$, we apply a random hash function $\hbar(\cdot)$ to the string of ordered edges in $C_{i,j}$ to generate an index $H_{i,j} \in \{1, \cdots, P\}$. If a clique with class label L_i is hashed to an index $H_{i,j}$, we add 1 to the entry $\Delta[H_{i,j}, L_i]$, which means clique-pattern $C_{i,j}$ has a contribution to class L_i. This fixed-size pattern-class table is continuously updated as new cliques are detected over the graph stream. This procedure is detailed in Algorithm 2.

Algorithm 2 Clique Hashing

Input: C_i: the clique set detected from $\overline{G_i}$

Output: Δ: $P \times M$ pattern-class table

1: **for** $j = 1{:}\,1{:}\,size(C_i)$

2: $\quad H_{i,j} := \hbar(C_{i,j})$;

3: $\quad \Delta[H_{i,j}, L_i] := \Delta[H_{i,j}, L_i] + 1$;

4: **end for**

4.3.3 CLIQUE-BASED CLASSIFIER

Given the "in-memory" pattern-class table Δ, we can construct a rule-based classifier by identifying frequent yet discriminative clique-patterns from Δ. To identify frequent clique-patterns, we first sum up the counts in each row of Δ and divide them by the number of graphs received thus far. The result for each row indicates the occurrence frequency of a set of cliques with the same hash value in the graph stream. Then we sort them in a descending order and set a threshold parameter α to select the clique-patterns whose frequencies $>$ α. These selected cliques are frequent clique-patterns which are also the candidates for the subsequent discriminative clique-pattern selection.

Next we can determine whether a frequent clique-pattern is also a discriminative one by comparing its occurrence ratios on the M classes (corresponding to the M columns in Δ). For a candidate clique-pattern, the ratio in column j represents the probability that the clique-pattern belongs to class j. A higher probability on a certain class indicates a better discriminative capability. Similarly, we can set a threshold parameter θ to select the clique-patterns whose maximum ratios $\geq \theta$. Fig. 4.3 gives a toy example for selecting the frequent and discriminative clique-patterns from a pattern-class table Δ.

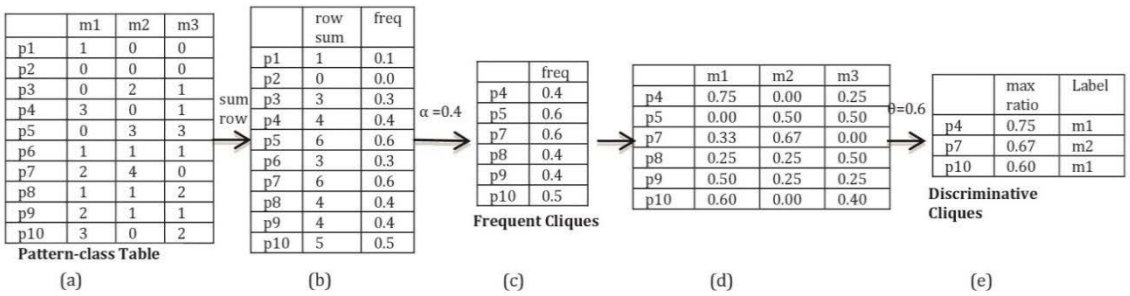


Figure 4.3: A toy example of frequent and discriminative clique-pattern mining

In Fig. 4.3, (a) a pattern-class table with 10 clique-patterns $\{p_1, \dots, p_{10}\}$ and 3 classes $\{m_1, m_2, m_3\}$. (b) the sums of individual clique-patterns in rows and the corresponding occurrence frequencies (i.e., the sums divided by the number of graphs,

say 10 here). (c) the selected frequent clique-patterns whose frequencies are larger than the frequent pattern threshold $\alpha = 0.4$. (d) the occurrence ratios of the selected clique-patterns on the M classes. (e) the selected discriminative clique-patterns whose maximum ratios are larger than the discriminative pattern threshold $\theta = 0.6$.

Finally, based on the selected clique-patterns, we can classify a test graph using majority voting based on the detected cliques in the test graph. In particular, given a test graph G_{test}, we detect its cliques C_{test} using Algorithm 1 and hash its cliques $\{H_{test,j}\}_{j=1}^{size(C_{test})}$ to index its clique-patterns. Each clique corresponding to a discriminative clique-pattern will contributes a class label $L_{test,j} := Find - Rule(\Delta, H_{test,j})$. The class label of the test graph L_{test} is determined by the majority of class labels $\{L_{test,j}\}_{j=1}^{size(C_{test})}$. This procedure is detailed in Algorithm 3.

Algorithm 3 Graph Classification

Input: G_{test}: a test graph

Output: L_{test}: the predicted label of G_{test}

1: $C_{test} = Clique - Detect(G_{test})$

2: **for** $j = 1{:}\,1{:}\,size(C_{test})$

2: $H_{test,j} := \hbar(C_{test,j})$;

3: $L_{test,j} := Find - Rule(\Delta, H_{test,j})$;

4: **end for**

5: $L_{test} := Majority - Vote\left(\{L_{test,j}\}_{j=1}^{size(C_{test})}\right)$;

4.4 EXPERIMENT

In this section, we will test the proposed DICH method for graph stream classification on two real-world data sets. In particular, we will evaluate the effectiveness and efficiency of DICH by comparing it with the 2-D hash compressed stream classifier [95],

which is the only state-of-the-art method applicable to graph stream classification. We use the following data sets in our experiments.

- DBLP Data Set[2]: In this data set, authors are nodes and co-authorship forms edges, and a graph is constituted by the co-authors of a paper. There are three classes in the data set: 1) Database related conferences, 2) Data mining related conferences, and 3) All remaining conferences. Our goal is to classify a test paper into one of three classes. The final data set contains over 5×10^5 authors, 9.75×10^5 edges, and 3.55×10^5 different graphs as the training data. We divide the data set into five splits and choose four splits as the training data and the remaining split as the test data.

- IBM Sensor Data Set[3]: This data set records the information from local traffic constituted by each graph on a sensor network. The IP-addresses are nodes and local traffic flows are edges. Each graph is associated with a particular intrusion type and there are over 300 different intrusion types (classes) in the data set. Our goal is to classify a traffic flow pattern into one of intrusion types. Because the number of classes is extremely large (> 300), we expect the overall accuracy to be relatively low. The data set contains more than 1.57×10^6 graphs. We choose 90% of the data as the training data and the remaining 10% as the test data.

4.4.1 EFFECTIVENESS EVALUATION

In this experiment, we evaluate the effectiveness of DICH by comparing it with the 2-D hash compressed stream classifier proposed in [95]. We will investigate the classification performance and sensitivity of the two methods by varying 1) the frequent pattern threshold α, 2) the discriminative pattern threshold θ, and 3) the size of the compressed edge set N.

[2] http://www.charuaggarwal.net/dblpcl/
[3] http://www.charuaggarwal.net/sens1/gstream.txt

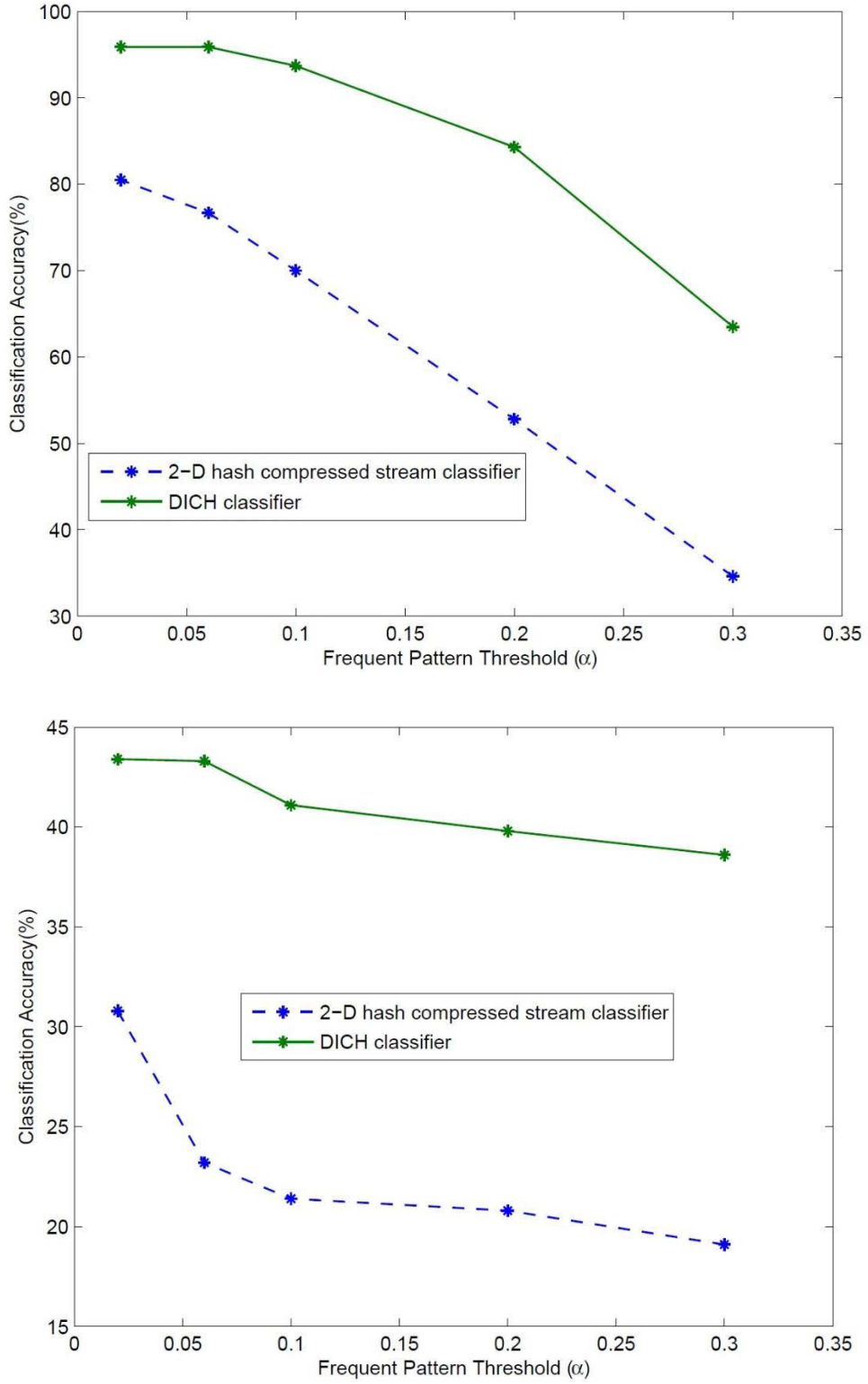


Figure 4.4: Effectiveness evaluation w.r.t. α on the DBLP data set (left) and the IBM sensor data set (right), $N = 5000$ and $\theta = 0.4$

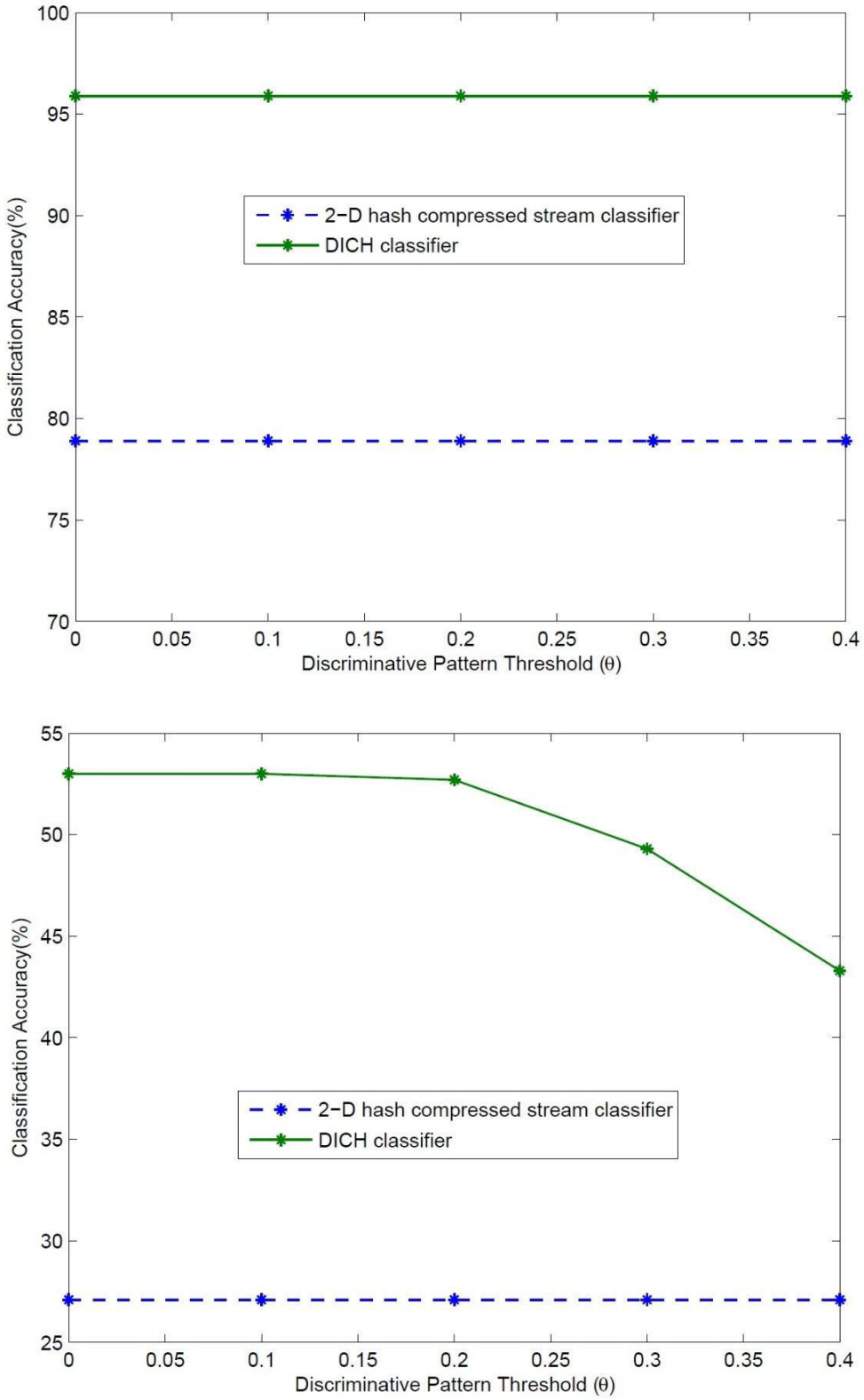


Figure 4.5: Effectiveness evaluation w.r.t. θ on the DBLP data set (left) and the IBM sensor data set (right), $N = 5000$ and $\alpha = 0.05$

First, we adjust the frequent pattern threshold[4] α for performance evaluation and fix the other two parameters by setting $N = 5000$ and $\theta = 0.4$. Fig. 4.4 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. α ($\mathcal{X}$-axis) on the two data sets. We can see that the classification performance of DICH is much higher than the 2-D hash compressed stream classifier on both data sets and in all values of α. The performance of both classifiers trends to decline as α becomes larger since more graph features will be eliminated and such information loss will affect classification performance. By comparing the curve slopes of two classifiers, the 2-D hash compressed stream classifier is more sensitive to α. In the case of $\alpha = 0.3$, the classification accuracy of the 2-D hash compressed stream classifier is much lower. From this experiment, we can validate the effectiveness of DICH, which can clearly outperform the 2-D hash compressed stream classifier and is more insensitive to the frequent pattern threshold.

Second, we adjust the discriminative pattern threshold θ for performance evaluation and fix the other two parameters by setting $N = 5000$ and $\alpha = 0.05$. Fig. 4.5 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. θ ($\mathcal{X}$-axis) on the two data sets. On the DBLP data set, DICH was significantly superior to the 2-D hash compressed classifier in classification accuracy. The classification performance of both methods was insensitive to θ, which may be due to the fact that the two classes (Database related conferences and Data mining related conferences) in DBLP data are extremely rare, the identified frequent patterns have already had relatively high discriminative capability. On the IBM sensor data set, although DICH is somewhat more sensitive to θ than the 2-D hash compressed classifier, it has much higher classification accuracy in all cases. This experiment further demonstrates that DICH has higher effectiveness than the 2-D hash compressed classifier in terms of the discriminative pattern threshold.

[4] In [95], the frequent pattern threshold α is used to screen out subgraph patterns.

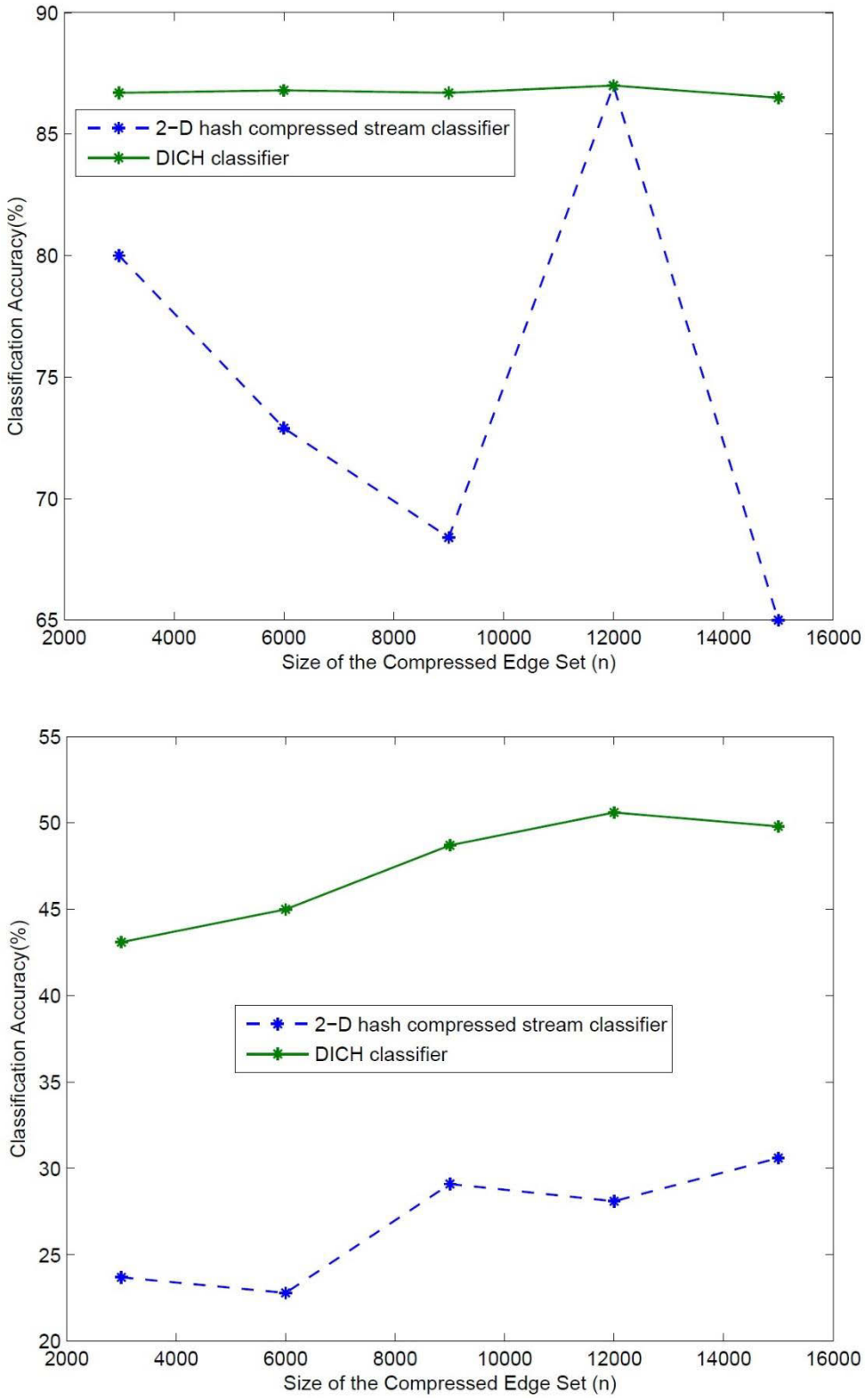
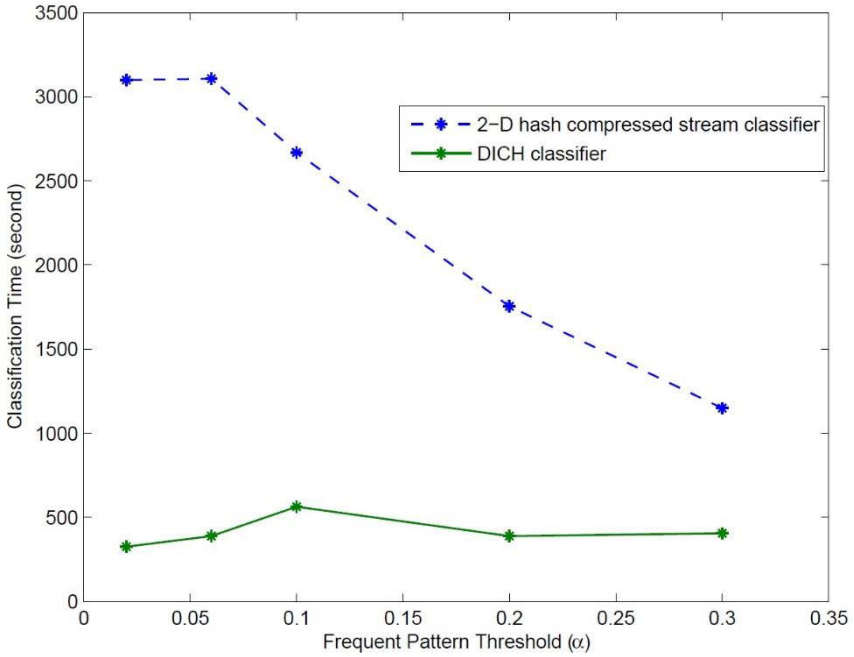


Figure 4.6: Effectiveness evaluation w.r.t. the edges compression size N on the DBLP data set (left) and the IBM sensor data set (right), $\alpha = 0.06$ and $\theta = 0.3$

Third, we adjust the size of the compressed edge set N for performance evaluation and fix the other two parameters by setting $\alpha = 0.06$ and $\theta = 0.3$. Intuitively, the classification performance of both classifiers will increase at the expense of more space. Fig. 4.6 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. N ($\mathcal{X}$-axis) on the two data sets. We can see that the 2-D hash compressed classifier is very sensitive to N, especially on the DBLP data set; while DICH is steadier on the DBLP data set but no clear performance improvement can be observed as N becomes larger. On the IBM sensor data set, the performance of both classifiers is improved as N becomes larger. Again, DICH outperforms the 2-D hash compressed classifier in all cases.

4.4.2 Efficiency Evaluation

In this experiment, we evaluate the efficiency of the two compared methods on the DBLP data set by adjusting the frequent pattern threshold α, the discriminative pattern threshold θ, and the size of the compressed edge set N. The settings of these parameters are the same as those in the above effectiveness evaluation. All the experiments are conducted on a Linux Cluster which comprises 24 nodes with 3.33GHz Intel Xeon CPU (64bit). Both DICH and the 2-D hash compressed stream classifiers are implemented using R studio.



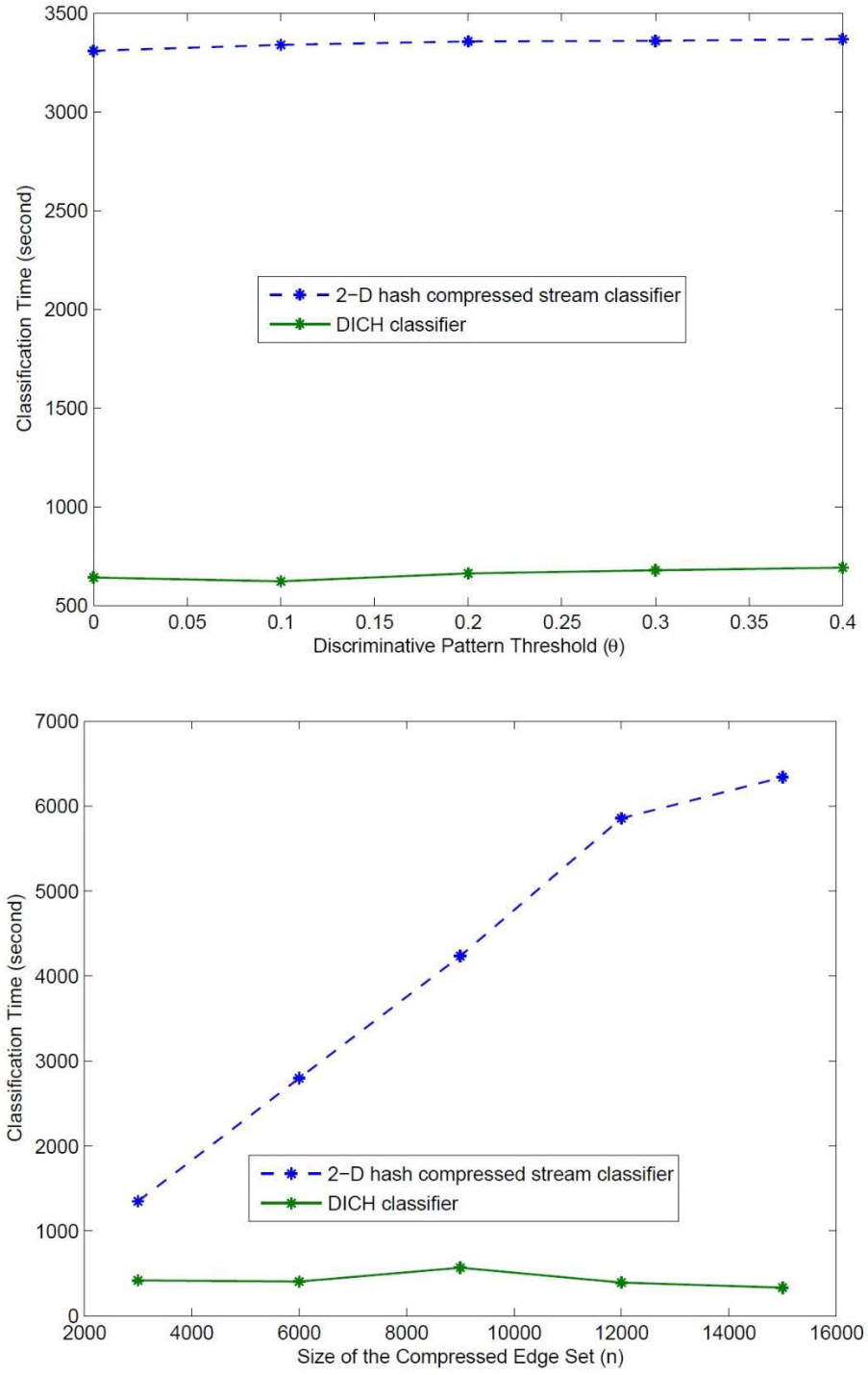


Figure 4.7: Efficiency evaluation (1) w.r.t. α, $N = 5000$ and $\theta = 0.4$ (left); (2) w.r.t. θ, $N = 5000$ and $\alpha = 0.05$ (right); and (3) w.r.t. N, $\alpha = 0.06$, $\theta = 0.3$ (down), on the DBLP data set

Fig. 4.7 plots the training time curves of the two compared methods w.r.t.α, θ, and N on the DBLP data set. We can see that the training time of DICH is significantly less than the compressed hash-based classifier in all cases. The computational cost of the 2-D hash compressed classifier is much higher because it requires an additional frequent pattern mining procedure to perform on the edge co-occurrence table which comprises massive transactions. In contrast, DICH employs a fast clique detection algorithm, which can directly find cliques (connected subgraphs) from the graph stream as features for classifier construction, such that no additional frequent pattern mining procedure is required to find connected subgraph patterns. This experiment shows that DICH clearly outperforms the 2-D hash compressed classifier in not only classification accuracy but also training efficiency.

4.5 SUMMARY

This chapter focused on graph structured data and proposes a DIscriminative Clique Hashing (DICH) for fast graph stream classification. The main idea is to employ a fast algorithm to decompose a compressed graph into a number of cliques to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are employed to speed up the discriminative clique-pattern mining process and address the unlimitedly clique-pattern expanding problem. The hashed cliques are used to update an "in-memory" fixed-size pattern-class table, which is finally used to construct a rule-based classifier. We test DICH on two real-world graph stream data sets. Because DICH directly extracts cliques (connected subgraphs) from the graph stream as features for classifier training, rather than mining unconnected co-occurrence edge sets as that in the compared state-of-the-art method [8], DICH can significantly outperform [8] in both classification accuracy and learning efficiency.

CHAPTER 5
ADAPTIVE HASHING FOR REAL-TIME GRAPH STREAM CLASSIFICATION

5.1 INTRODUCTION

In this Chapter, in order to further improve the performance of the graph stream classification algorithm based on hashing (DICH) in the Chapter 4 and make the graph stream classification algorithm more adaptive in the streaming situation, we do further research on the graph stream classification based on hashing in this chapter.

Graph stream classification on a complex network with massive nodes is by no means an easy problem because of the following challenges:

- **Expanding Feature Space**: Graph stream is defined on a massive universe of nodes. The continuously received graphs in the stream would lead to an increasing number of subgraph patterns. Due to the "one-pass" nature of the graph stream, we cannot enumerate all the subgraph-patterns in a pre-scan to construct the feature space for all the graphs. Thus we need to design a projection that can map arbitrary subgraph patterns to a size-fixed compatible feature space for all the graphs in the stream.

- **Increasing Graph Volumes**: The volume of graph data are continuously growing with a high speed. Due to the "real-time" requirement of graph stream classification, we cannot employ any existing subgraph detection method which is designed for off-line accurate subgraph mining. Thus we need to develop an

approximate subgraph feature extraction method that is fast enough to tackle fast increasing graph volumes.

- **Concept Drifts**: As challenging as the increasing data volumes and expanding feature space, however, is the concept drifting in the graph stream, which implies that the data distributions and the decisions for classification may continuously evolve and change. Accordingly, graph stream classification needs to ensure that classification models can quickly discover changes in the stream, and adapt to the changes for accurate classification.

In [95], [18], the authors have investigated the graph stream classification problem. Both of them employ hashing techniques to sketch the graph stream for saving computational cost and controlling the size of the subgraph-pattern set. Aggarwal [95] proposed a 2-D hashing scheme to construct an "in-memory" summary for sequentially presented graphs and used a simple heuristic to select a set of most discriminative frequent patterns to build a rule-based classifier. Although [95] has exhibited promising performance on graph stream classification, it has two inherent limitations: (1) The selected subgraph-patterns are composed of disconnected edges, which may have less discriminative capability than connected subgraph-patterns due to a lack of semantic meaning. (2) The computational cost is high because an additional frequent pattern mining procedure is required to perform on the summary table which comprises massive transactions. Our previous work, DIscriminative Clique Hashing (DICH) [18], has addressed these limitations to a certain extent by employing a fast clique detection algorithm from hashed graphs. However, DICH also has two major disadvantage in handling graph stream: (1) the hashing space of DICH is fixed, so it cannot adapt to changing nodes and structures in the graph streams; (2) the prediction of decision DICH is based on simple decision rules, which makes it inefficient in handling concept drift in the graph streams. To address the problem of the concept drifting, an instance weighting mechanism has been proposed in the gSLU [103] to adjust the subgraph feature selection module for emerging concept drifting graphs. However, gSLU is based on a frequent subgrpah pattern mining based framework, so it cannot achieve real-time response for graph stream classification. In addition, the instance weighting module employed in

gSLU is too time consuming for graph streams, because it needs to iteratively tune the weight values for each single graph.

Motivated by the aforementioned challenges and the limitations of the existing approaches, in this chapter we propose an adaptive real-time classification method for graph stream using stochastic learning, differential hashing techniques and a chunk level weighting mechanism to address these problems. In particular, for better adapting to concept drifting, the whole graph stream is partitioned into a number of non-overlapping graph chunks each containing the same number of graphs. For each chunk, we employ a random hashing scheme to compress the original node set of the graph stream for fast feature detection. To tackle the concept drifting, a differential hashing scheme is used to map unlimitedly increasing features (cliques) onto a size- fixed feature space. The distribution of the hashed cliques in each received graph is used as a feature vector for stochastic learning of the real-time chunk classifier. Then, a chunk level weighting mechanism is used to form an ensemble for graph stream classification. The proposed method substantially speeds up the graph feature extraction process, solves the unlimited graph feature expanding problem, and effectively adapts to the concept drifting in graph stream classification. The contributions of this chapter are highlighted as follows:

1. We propose an approximate method for fast graph feature extraction by detecting cliques from the compressed graphs via hashing. The method significantly improves the efficiency of feature extraction and classifier learning online to satisfy the “real-time” requirement.

2. We propose a graph feature reduction method by mapping unlimitedly expanding clique patterns onto a fixed-size compatible feature space via differential hashing. This can avoid a pre-scan of graphs to further speed up the learning process, satisfying the “one-pass” requirement and adapting to the concept drifting.

3. We adopt the stochastic learning strategy to incrementally train a graph classifier online, which can satisfy the “real-time” requirement and achieve better classification performance than the majority voting used in [95] [18].

4. Combined with differential hashing scheme, we adopt a chunk level weighting mechanism to form a weighted classifier ensemble for graph stream classification, which can effectively adapt to the concept drifting and achieve better performance than instance level weighting mechanism [103].

We conduct extensive experiments on two real-world graph stream data sets and one synthetic graph stream data set. The experimental results demonstrate that the proposed method can clearly outperform the compared state-of-the-arts [95], [18], [103] in classification accuracy, training efficiency and concept drifting.

The remainder of the chapter is organized as follows. Section 5.2 presents problem definition and system overview. The ARC-GS algorithm (Adaptive Real-time Classification for Graph Stream) is proposed in Section 5.3. The approach has been tested using two real-world graph streams and one synthetic graph stream. The experimental evaluations and results are given in Section 5.4. Finally, the summary is given in Section 5.5.

5.2 DEFINITIONS & SYSTEM OVERVIEW

5.2.1 PROBLEM DEFINITION

Suppose there exists a complex network which comprises a massive universe of nodes. The edges connecting these nodes are denoted by the edge set ε. The stream of graphs $\{\cdots, G_{n-1}, G_n, G_{n+1}, \cdots\}$ are presented sequentially as the subsets of ε (all are connected graphs), where the subscript n denotes the receiving order of the graph in the stream. An example of such graph streams is a coauthor network. All the papers (graphs of connected authors) on the coauthor network with different time-stamps form a graph stream.

Specifically, the edge set of G_n is denoted by $\varepsilon_n = \{e_1, \cdots, e_E\} \subset \varepsilon$. Each graph G_n has a class label $l_n \in \{1, \cdots, L\}$. We represent each received graph G_n in the form of $\langle n, e_1, \cdots, e_E, l_n \rangle$. In this chapter, we assume that each edge in a graph has a numerical

weight w_{ij}, where i and j are the indices of the two vertices of the edge (for simplicity, we don't consider edge labels).

Our **goal** is to learn a graph classification model from the graphs $\{\cdots, G_{n-1}, G_n, G_{n+1}, \cdots\}$ observed from the stream thus far, in an efficient way, to accurately predict the class label of a test graph $G_{test}(test > n)$ in the future graph stream.

5.2.2 SYSTEM OVERVIEW

The framework of our method for graph stream classification is shown in Fig. 5.1, which includes four modules: graph clique detection, differential graph clique hashing, clique-based chunk classifier learning, and weighted chunk classifier ensemble. The graphs in the stream are received one by one. Each received graph G_n is fed into the pipeline in each chunk for processing in the following stages:

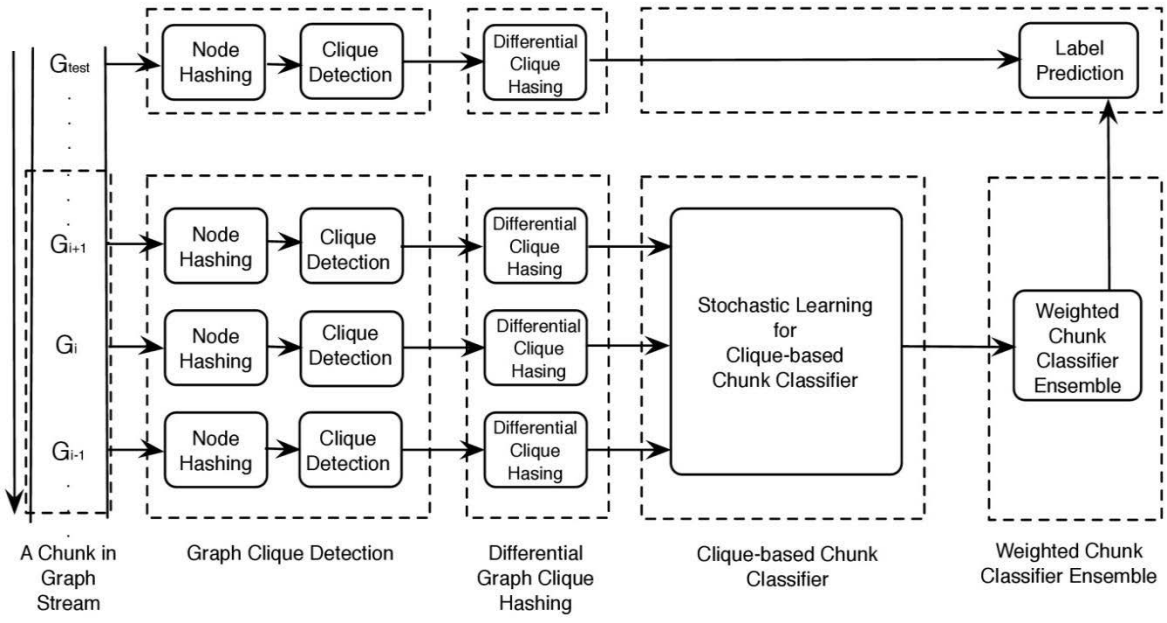


Figure 5.1: The framework of the proposed adaptive real-time hashing for graph stream classification method

1. ***Graph Clique Detection***. Each received graph G_n from the stream is first compressed into a small graph $\bar{G}_n$ using a random hashing scheme. Then we

employ a fast algorithm to decompose the compressed graph into a number of cliques as the features of G_n.

2. ***Differential Graph Clique Hashing***. Since the number of clique patterns (features) will unlimitedly increase as new graphs are fed in, the underlying feature space will keep expanding accordingly. Thus, in this stage, a clique hashing scheme is performed to map the unlimitedly emerged clique patterns onto a fixed-size clique-pattern set, denoted by $\mathcal{C} = \{\bar{C}_1, \cdots, \bar{C}_M\}$. Now all the graphs in the stream can be represented in this compatible space, for G_n, we have $x^n = \left[x_1^{(n)}, \cdots, x_M^{(n)}\right]^{\mathsf{T}}$.

3. ***Clique-based Chunk Classifier Learning***. For each chunk, the hashed clique-pattern representation χ^n of each received graph and its corresponding class label l_n are used to incrementally update the underlying chunk classifier online, using a stochastic learning algorithm.

4. ***Weighted Chunk Classifier Ensemble***. In order to adapt to the concept-drifting in graph streams, we propose the weighted chunk classifier ensemble to reduce the impact of concept drifting on the classification performance. In this module, multiple weighted chunk classifiers are adopted to form an ensemble to predict future graph stream. Combined with the differential hashing scheme, we design this ensemble to make our method more adaptive.

To classify a test graph G_{test} in the future graph stream, G_{test} is processed in the first two modules and the obtained hashed clique-pattern representation χ^{test} is input to the ensemble classifier for class label prediction. The detailed approaches used in the four modules are introduced as follows.

5.3 ARC-GS: ADAPTIVE REAL-TIME CLASSIFICATION FOR GRAPH STREAM

We now discuss technical details on adaptive real-time classification for graph stream.

5.3.1 GRAPH CLIQUE DETECTION

This section is the same as the Section 4.3.1.

5.3.2 DIFFERENTIAL GRAPH CLIQUE HASHING

The cliques extracted from the compressed graph are used to represent the features of the corresponding original graph. To learn a graph classifier, it is required to make the features of all the graphs to be in the same feature space. Since the number of clique patterns (features) will unlimitedly increase as new graphs are fed in over the stream, the underlying feature space will keep expanding accordingly. Meanwhile, in a certain period of time, there may be some abnormal or entirely different feature information called as "concept drifting" [105]. To address these problems, we use the "new clique" and "old clique" to distinguish the new cliques from the existing cliques, and adopt differential feature hashing to constraint the dimensionality of the feature space. Both the "new and old cliques" and "differential feature hashing" can help the algorithm adapt to the changes in the streams and reduce the impact of the concept drifting.

The differential clique hashing scheme is applied on the new and old cliques. The differential clique hashing not only help us control the feature space, but also identify the abnormal features in good time. We can differentially process them and address the concept drifting. Next, we first introduce how to define the "new clique" and "old clique".

The whole stream is partitioned into a set of sequential chunks. The ultimate goal is to constantly train an ensemble classifier to predict the class label of each test graph G_{test} in the future test chunk. An ensemble includes a certain number of weighted chunk classifiers. We set the ensemble size as K which represents the number of chunk classifiers in an ensemble. When detecting a clique from each graph in current chunk, we will check this clique in the previous K chunks. It this clique appears in the previous

K chunks, we set this clique as an "old clique", and otherwise we set it as a "new clique". These K chunks will vary according to the current chunk, so the "old" or "new" is a relative concept.

The following differential clique hashing scheme is performed to map the unlimitedly emerged clique patterns onto a fixed-size clique-pattern set, denoted by $\mathcal{C} = \{\bar{C}_1, \cdots, \bar{C}_M\}$ (M is the range of clique hash values). In order to distinguish new cliques from old cliques in the clique-pattern set, we set a ratio value as R. For the new cliques, the hash space is$M * R$; for the old cliques, the hash space is $M * (1 - R)$. The differential clique hashing scheme will map the new and old cliques onto corresponding fixed-size clique-pattern subset.

Given a graph G_n received from the graph stream, we first use Algorithm 1 to detect its new clique set C_{new} and old clique set C_{old}. Then, for each new clique $C_{new,k}$ in C_{new} and each old clique $C_{old,k}$ in C_{old}, we differentially apply a random hash function $\hbar(\cdot)$ to $C_{new,k}$ and $C_{old,k}$ to generate corresponding index $\hbar_{new,k} \in \{1, \cdots, M * R\}$ and $\hbar_{old,k} \in \{M * R + 1, \cdots, M\}$ as follows

$$\hbar_{new,k} = \hbar\big(str\big(C_{new,k}\big), M * R\big) \tag{5.5}$$

$$\hbar_{old,k} = \hbar\Big(str\big(C_{old,k}\big), M * (1 - R)\Big) \tag{5.6}$$

where $str\big(C_{new,k}\big)$ denotes the string of the ordered node indices of $C_{new,k}$ and $M * R$ constraints the range of hash values in $\{1, \cdots, M * R\}$; the $str\big(C_{old,k}\big)$ denotes the string of the ordered node indices of $C_{old,k}$ and $M * (1 - R)$ constraints the range of hash values in $\{M * R + 1, \cdots, M\}$. We use a vector $x^n = \left[x_1^{(n)}, \cdots, x_M^{(n)}\right]^{\mathrm{T}}$ to represent a graph G_n, where $x_m^{(n)}$ is the frequency of cliques in C_n whose indices are m based on the clique hash function Eq. (5.5) and Eq. (5.6). This procedure is detailed in Algorithm 2. Thus far, all the graphs in the stream can be represented in an M-dimensional compatible space.

Algorithm 2 Differential Graph Clique Hashing

Input: C_n: the all clique set detected from $\bar{G}_n$; C_{new}: the new clique set detected from $\bar{G}_n$; C_{old}: the old clique set detected from $\bar{G}_n$

Output: x^n: the feature vector of G_n

1: $x^n = [0, \cdots, 0]^\intercal$;

2: $C_n = C_{new} + C_{old}$;

3: **for** $k = 1: \text{length}(C_{new})$

4: $\hbar_{new,k} = \hbar\big(str\big(C_{new,k}\big), M * R\big)$;

5: **if** $\hbar_{new,k} = m(1 \le m \le M * R)$

6: $x_m^{(new)} = x_m^{(new)} + 1$

7: **end if**

8: **end for**

9: **for** $k = 1: \text{length}(C_{old})$

10: $\hbar_{old,k} = \hbar\left(str\big(C_{old,k}\big), M * (1 - R)\right)$

11: **if** $\hbar_{old,k} = m\big(1 \le m \le M * (1 - R)\big)$

12: $x_m^{(old)} = x_m^{(old)} + 1$

13: **end if**

14: **end for**

15: $x^n = x^n / \sum_{m=1}^{M} x_m^{(n)}$

5.3.3 CLIQUE-BASED CHUNK CLASSIFIER

In the chunk classifier learning stage, we first construct a feature vector $x^n = \left[x_1^{(n)}, \cdots, x_M^{(n)}\right]^\intercal$ for each received graph G_n in the graph stream using the method introduced above. Then, we will use these continuously obtained feature vectors to learn a chunk classifier for each chunk. This procedure has become straightforward now since we have converted complex graph structures into the same representations, which can be learned using a generic online learning algorithm.

Suppose there are N graphs in the entire stream (N may approach infinity). As each feature vector $x^{(n)}$ is M-dimensional, we assume the entire data set for training is a feature matrix $x \epsilon \mathbb{R}^{M \times N}$, where N columns correspond to N graphs and M rows correspond to M hashed clique patterns. We also construct an label matrix $Y \epsilon \{0,1\}^{L \times N}$ for supervised learning, where L rows correspond to L classes; if G_n belong to the l th class, $Y_{l,n} = 1$; otherwise $Y_{l,n} = 0$.

As the feature vector $\chi^{(n)}$ of G_n actually describes a distribution of the M clique patterns, we thus adopt a regularized ridge regression model to fit the label distribution of G_n

$$\mathrm{Q}(W) = \frac{1}{2}\|WX - Y\|_2^2 + \frac{\lambda}{2}\|W\|_2^2 \tag{5.7}$$

where $W \epsilon \mathbb{R}^{L \times M}$ is the weight matrix. We aim to minimize the following objective to estimate W

$$W^* = \underset{W}{\operatorname{argmin}}\, Q(W) \tag{5.8}$$

which is a linear least-squares problem.

Many methods can be used to solve Eq. (5.8), such as gradient descent and Newton's method. The standard (or "batch") gradient descent method will perform the following iterations

$$W \coloneqq W - \alpha \nabla Q(W) = W - \alpha \sum_{n=1}^{N} \nabla Q_n(W) \tag{5.9}$$

where α is the step size (i.e., learning rate) and

$$Q_n(W) = \frac{1}{2}\left\|W\mathrm{x}^{(n)} - \mathrm{y}^{(n)}\right\|_2^2 + \frac{\lambda}{2N}\|W\|_2^2 \tag{5.10}$$

where $\mathrm{x}^{(n)}$ and $\mathrm{y}^{(n)}$ are the nth columns of X and Y, differentially.

However, in our problem setting, these training pairs of graphs are obtained one by one in sequence, $(\mathrm{x}^{(n)}, \mathrm{y}^{(n)})$ for $n = 1,2,\cdots$, rather than in batch. We cannot simply use the standard gradient descent method to optimize W. Thus, we resort to the stochastic

(incremental) gradient descent method to solve this problem, which updates W using a single example at each iteration, for $n = 1,2, \cdots$

$$W \coloneqq W - \alpha \nabla Q_n(W) \tag{5.11}$$

where

$$\nabla Q_n(W) = W \mathrm{x}^{(n)} \left[\mathrm{x}^{(n)}\right]^{\mathrm{T}} - \mathrm{y}^{(n)} \left[\mathrm{y}^{(n)}\right]^{\mathrm{T}} + \frac{\lambda}{N} W \tag{5.12}$$

Based on the iteration function (Eq. (5.12)), the linear regression model W is updated online at the arrival of each new graph in the stream. In the beginning of the graph stream, W can be initialized based on a chunk of cached graphs using the standard gradient descend method. Every time after finishing updating W for the last graph in a chunk, we use the current W as current chunk classifier. In fact, the real-time update for W can be stopped at any time for testing tasks.

5.3.4 WEIGHTED CHUNK CLASSIFIER ENSEMBLE

Based on the aforementioned modules, we can get the classifier of each chunk. When we predict the label of each graph in the test chunk, we adopt a weighted chunk classifier ensemble.

In our algorithm, we use $\{S_1, S_1, \cdots S_n\}$ to represent sequential chunks, which comprise the same number of graphs. The S_n is the most up-to-date chunk. For each chunk $S_i (1 \leq i \leq n)$, we use the C_i to represent the chunk classifier. We set the ensemble size as K. For the current testing chunk S_T, we will use the previous K weighted chunk classifiers as an ensemble to predict the class label of each graph in current testing chunk S_T separately. The process is as follows:

1. First, a weight W_i is assigned to each individual chunk classifier $C_{T-i} (1 \leq i \leq K)$.

2. Second, each chunk classifier C_{T-i} is used to predict the class label of a graph, and then K predicted labels for the graph in chunk S_T are generated.

3. Third, each of K predicted labels for the graph is assigned the same weight as that of each corresponding chunk classifier C_{T-i},

4. Fourth, we sum up all the weight values of each same label and choose the label with the largest weight as the final label of this graph.

Next, we will introduce how to predict the class label of a graph and assign the weight value for the classifier:

In particular, given a test graph G_{test}, we first detect its cliques using Algorithm 1, and then differentially hash its cliques to construct the clique-pattern feature vector x^{test} using Algorithm 2. At last, we calculate its class-label distribution vector y^{test} using

$$\mathrm{y}_{test} = W x^{test} \tag{5.13}$$

according to which the class label of G_{test} is predicted by

$$l_{test} = \underset{l}{\operatorname{argmin}}\{y_1^{test}, \cdots y_L^{test}\} \tag{5.14}$$

where $\{y_1^{test}, \cdots y_L^{test}\}$ are the values of the L dimensions of y_{test}.

For current testing chunk S_T, according to the hash space ratio R for new and old cliques, the weight value $\{W_{T-1}, W_{T-2}, \cdots, W_{T-K}\}$ of previous K classifiers $\{C_{T-1}, C_{T-2}, \cdots, C_{T-K}\}$ are $\{(1-R)^1, (1-R)^2, \cdots, (1-R)^K\}$. The rule is: the closer to the current testing chunk S_T, the larger the weight is. Suppose the corresponding K predicted labels is $\{L_1, L_2, \cdots, L_K\}$, we will sum up all the weight values of the same predicted label and choose the predicted label with the largest weight as the final label of this graph.

5.4 EXPERIMENT

In this section, we will empirically test the proposed classifier ARC-GS standing for Adaptive Real-time Classification for Graph Stream on two real-world graph streams and one synthetic graph stream. In particular, we will evaluate the effectiveness and efficiency of ARC-GS by comparing it with three baselines: 2-D hash compressed stream

classifier [95] which is a rule-based classifier using a simple heuristic to select a set of most discriminative frequent patterns; gSLU [103] which is a classifier using unique measures to discover informative subgraph features with minimum redundancy and use an instance weighting mechanism for emerging concept drifting graphs; and our previous work DICH [18] which is a majority voting classifier adopting a fast clique detection algorithm from hashed graphs. Then, we will compare the impact of the concept drifts on classification effectiveness of four classifiers. We aim to evaluate the capability of the proposed classifier in handling concept drifting in graph streams, compared with 2-D, gSLU and DICH classifiers. We do each set of experiments three times for each data set, and all the classification results in our experiments are the average performance over these three generated results. The four compared classifiers are implemented in Matlab and all the experiments are conducted on a node of Linux Cluster with 2.90GHz Intel Xeon CPU.

5.4.1 DATA SETS

We use the following graph streams in our experiments.

- **IBM Sensor Data Stream (IBM)**[5]: This data stream records the information from local traffic on a sensor network. The IP-addresses are nodes and local traffic flows are edges. Each graph is associated with a particular intrusion type and there are over 300 different intrusion types (classes) in the data set. Our goal is to classify a traffic flow pattern into one of intrusion types. Because the number of classes is extremely large (> 300), and many of them are rarely observed, we select 50 relative dense classes in our experiments for multi-class classification. The data set contains 1.0×10^6 nodes, 1.25×10^6 edges, which generate a stream of 5.0×10^5 graphs over time.

- **Citation Network Stream (CNS)**[6]: In this Citation Network, each node is a paper associated with rich attributes information (e.g., abstract, title, authors, etc.). In our experiment, we select 16,000 papers with authors and references attribute

[5] http://www.charuaggarwal.net/sens1/gstream.txt
[6] http://arnetminer.org/citation

information from two research areas, artificial intelligence (AI) and computer vision (CV), to generate a graph stream for binary classification. The edges are citations between papers and coauthorships between authors. Our goal is to predict which class a test paper in the stream belongs to. The final data stream contains 7.1×10^4 nodes, 5.8×10^4 edges, and 1.6×10^4 graphs.

- **GTGraph Stream (GTGraph)**[7]: This data stream is a synthetic data set generated by the graph generator GTGraph based on R-MAT model [106]. We choose default values of parameters suggested by the authors during network generation. Our GTGraph network contains 6.0×10^5 nodes and 5.0×10^5 edges, and the edges from the same node are used as a graph for experimental evaluation, which generate a stream of 1.0×10^5 graphs over time.

All the streams are divided into 25 non-overlapping chunks. For the Citation Network Stream (CNS), each of chunk comprises 640 graphs; for the IBM Sensor Data Stream (IBM), each of chunk comprises 20000 graphs; for the GTGraph Stream (GTGraph), each of chunk comprises 4000 graphs.

5.4.2 EFFECTIVENESS EVALUATION

In the following, we evaluate the effectiveness of our ARC-GS classifier by comparing it with the 2-D, gSLU and DICH classifiers on the IBM and CNS data sets. We investigate the classification accuracy of the four methods in terms of 1) the number of features M and 2) the ensemble size K. In the ARC-GS classifier, we set the hash ratio R as 0.2. At the end of this section, we separately investigate the classification accuracy of our ARC-GS classifier in terms of hash ratio R.

For the number of features M: in the ARC-GS and DICH classifiers, the M represents the fixed size of hashed clique-pattern set; in the 2-D classifier, the M represents the number of discriminative patterns in the underlying graph with the use of a 2-dimensional hashing scheme; in the gSLU classifier, the M represents the number of minimum-redundancy subgraph features.

[7] http://www.cse.psu.edu/ madduri/software/GTgraph/

For the ensemble size K: if current testing chunk is S_T, our ARC-GS classifier will use the most recent K weighted chunk classifiers $\{C_{T-1}, C_{T-1}, \cdots C\}$ as an ensemble to predict graphs in S_T; the gSLU classifier will built an ensemble of classifier from the most recent K chunks to predict graphs in the S_T; the 2-D and DICH classifiers will combine the most recent K chunks $\{S_{T-1}, S_{T-1}, \cdots S_{T-K}\}$ as traning data to train a classifier and predict graphs in S_T.

For the hash ratio R: in our ARC-GS classifier, the R represents the proportion to allocate the feature spaces for the new cliques and the old cliques.

Results w.r.t. the number of features M: In this experiment, we fix the ensemble size K ($K = 4$ for the IBM, and $K = 6$ for the CNS) and adjust the number of features M for effectiveness evaluation. For the IBM, we investigate the number of features M in $\{300,1000,3000\}$; for the CNS, we investigate the M in $\{5000,10000,15000\}$.

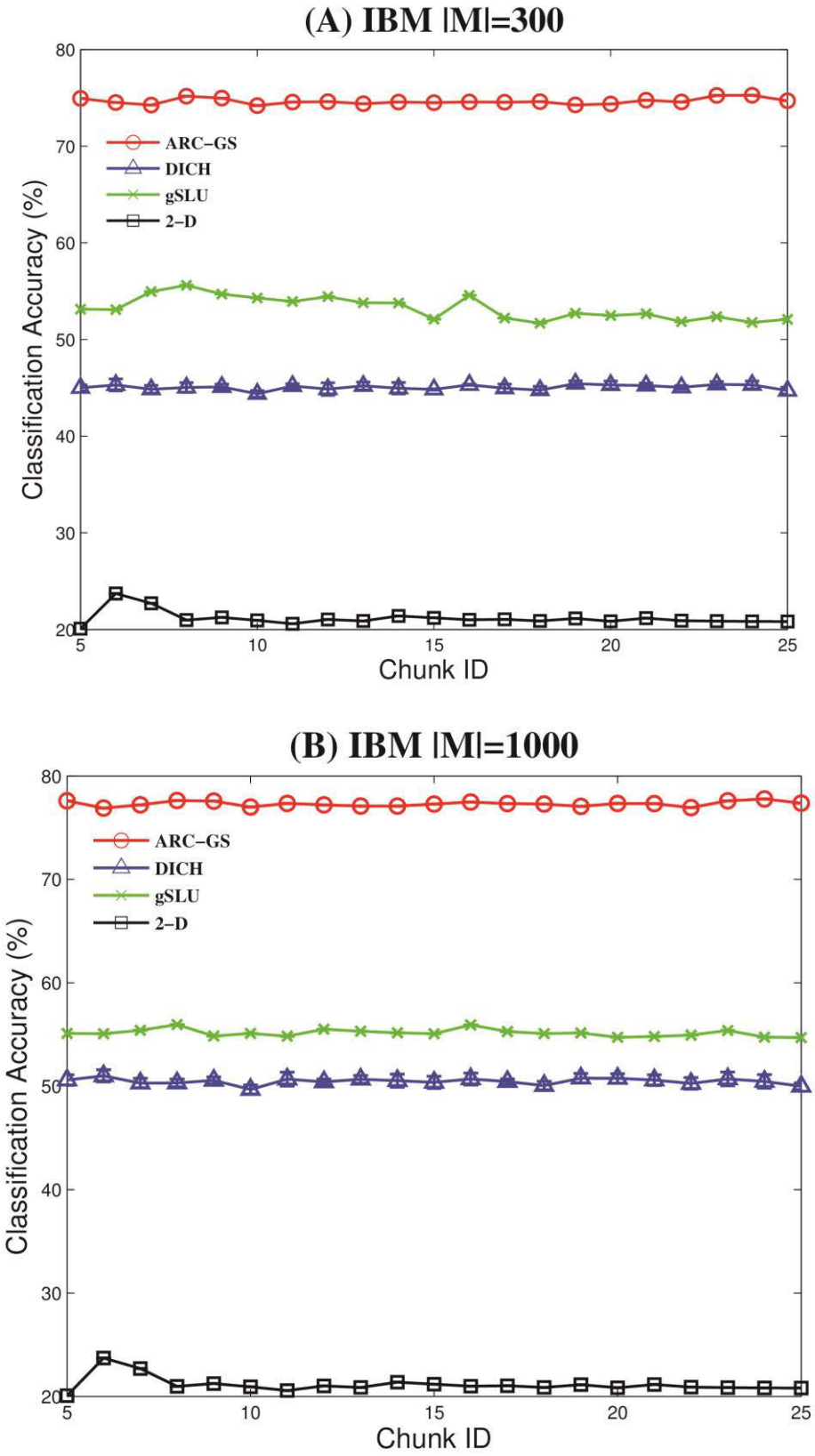
Fig. 5.2 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. the Chunk ID ($\mathcal{X}$-axis) in the IBM and CNS Sensor streams by using different numbers of features. The average classification accuracy over the entire streams is reported separately in Fig. 5.3. We can see that the overall classification performance of the ARC-GS classifier is the best among the four compared methods on the two graph streams under different settings of M across the whole stream. Especially in the IBM, the accuracy of the ARC-GS classifier is always higher than the 2-D, gSLU and DICH classifiers at all chunk IDs. In the CNS, we can see that our classification accuracy is more stable than the other classifiers, especially than the gSLU as the M increase. The reason is that the stochastic learning strategy adopted in our classifier can better satisfy the real-time requirement and achieve better classification. This experiment implies that our ARC-GS classifier can achieve significantly improved accuracy.

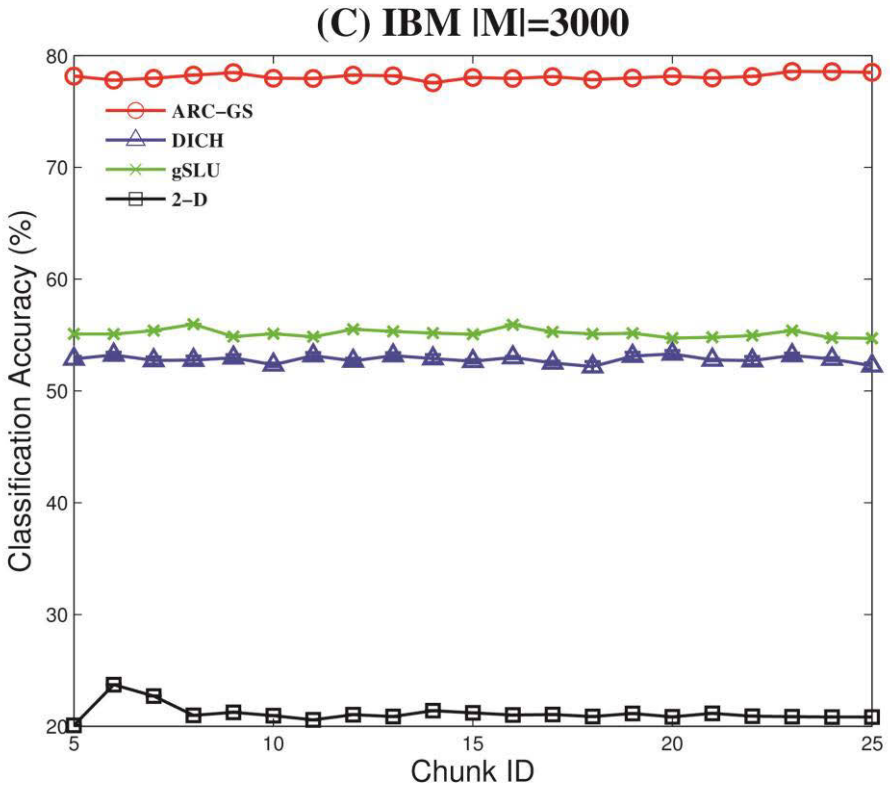
Fig. 5.3 further validates that our ARC-GS classifier outperforms the 2-D, gSLU and DICH classifiers especially in the IBM. Among these classifiers, the effectiveness of the 2-D classifier is the worst. The reason is that the selected subgraph-patterns with disconnected edges may have less discriminative capability than connected subgraph-patterns in the gSLU, DICH and ARC-GS due to a lack of semantic meaning. For the

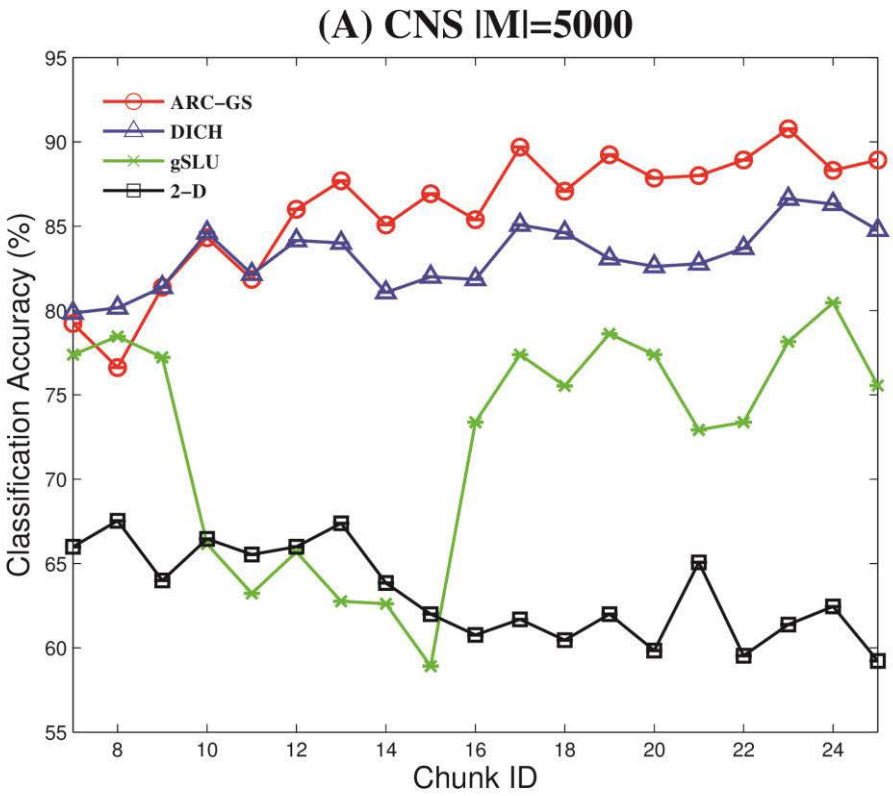
DICH and gSLU classifiers, the DICH classifier outperforms the gSLU classifier in the CNS, otherwise in the IBM. The reason is that the DICH is a majority voting classifier whose performance could degrade as the number of classes becomes large, and the IBM has more number of classes than the CNS. An interesting observation is that a larger number of features can help improve the accuracies of ARC-GS and DICH classifiers in both streams. The reason is that a relatively large number of features would have stronger discriminative capability to classify the graphs. We also find that the increase rate of average accuracy becomes smaller with respect to the increase of the number of features. In the IBM, when the number of features reaches 1000, all the four classifiers almost have enough discriminative capability to classify graphs, and the lager number of features will not significantly help improve the accuracy. For gSLU classifier, in the CNS, when the number of features reaches 10000, the average accuracy falls sharply. The reason is that those selected features are not beneficial and even misleading the classifier. For the 2-D classifier, in the IBM and CNS, when the number of features reaches 300 and 5000 respectively, it already has enough discriminative capability to classify graphs.

Results w.r.t. the ensemble size K: In this experiment, we fix the number of features M ($M = 1000$ for the IBM and $M = 10000$ for the CNS) and adjust the ensemble size K for effectiveness evaluation. For the IBM, we investigate the ensemble size K in $\{2,4,6\}$; for the CNS, we investigate the ensemble size K in $\{4,6,8\}$.

Fig. 5.4 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. the Chunk ID ($\mathcal{X}$-axis) in all these four streams by using different ensemble sizes. The average classification accuracy over the entire streams is reported separately in Fig. 5.5. We can see that the overall classification performance of the ARC-GS classifier is still the best among the four compared classifiers on both two streams under all settings of K. This experiment further implies that our ARC-GS classifier can significantly improve the classification accuracy, compared with the 2-D, gSLU and DICH classifiers.


(A) IBM |M|=300
ARC−GS
DICH
gSLU
2−D
Classification Accuracy (%)
Chunk ID
(B) IBM |M|=1000
ARC−GS
DICH
gSLU
2−D
Classification Accuracy (%)
Chunk ID


(C) IBM |M|=3000
ARC–GS
DICH
gSLU
2–D
Classification Accuracy (%)
Chunk ID
80
70
60
50
40
30
20
5
10
15
20
25


(A) CNS |M|=5000
ARC–GS
DICH
gSLU
2–D
Classification Accuracy (%)
Chunk ID
95
90
85
80
75
70
65
60
55
8
10
12
14
16
18
20
22
24

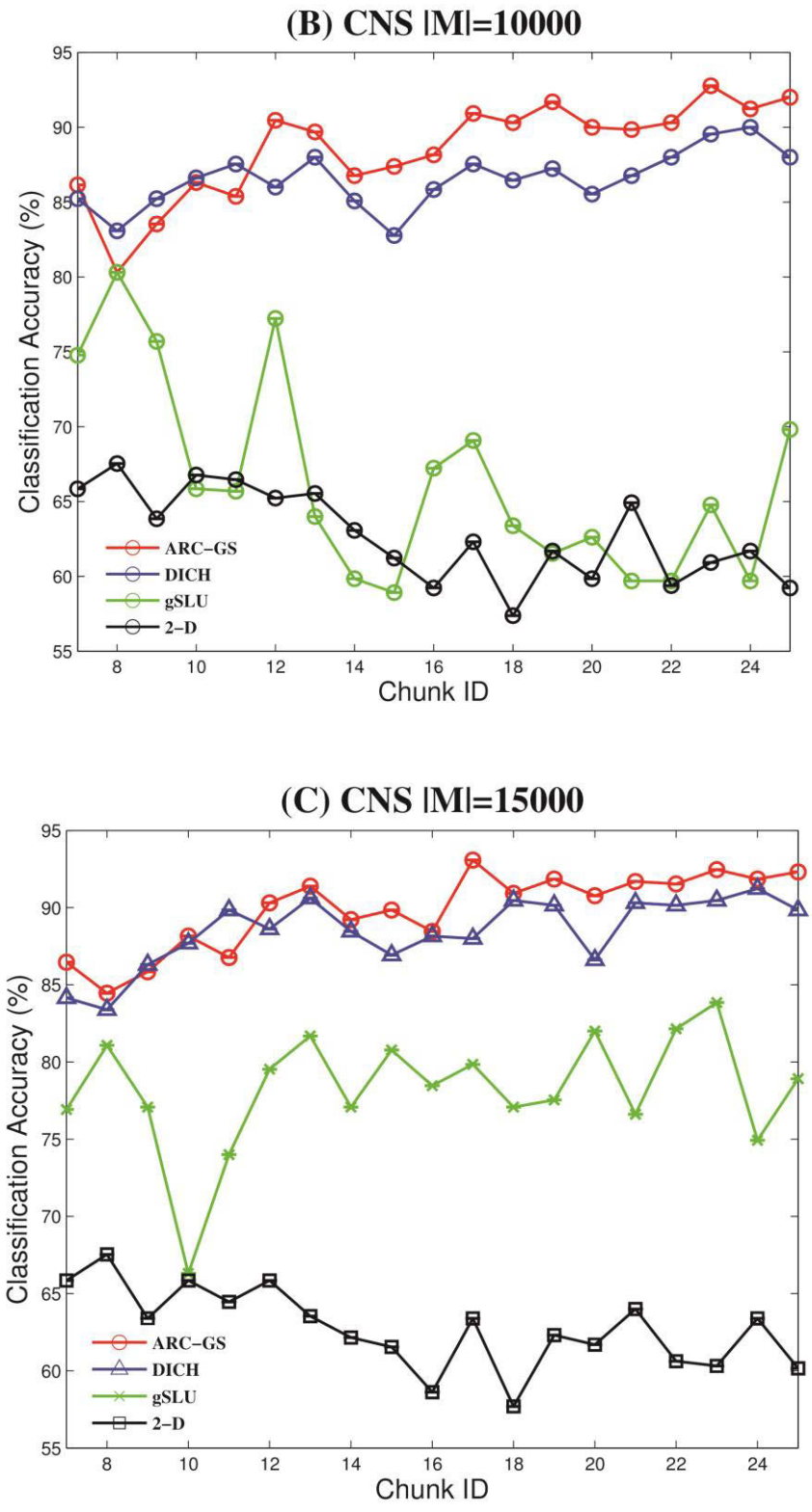


Figure 5.2: Classification accuracy on the IBM (ensemble size $K = 4$), and the CNS (ensemble size $K = 6$) with different numbers of features M

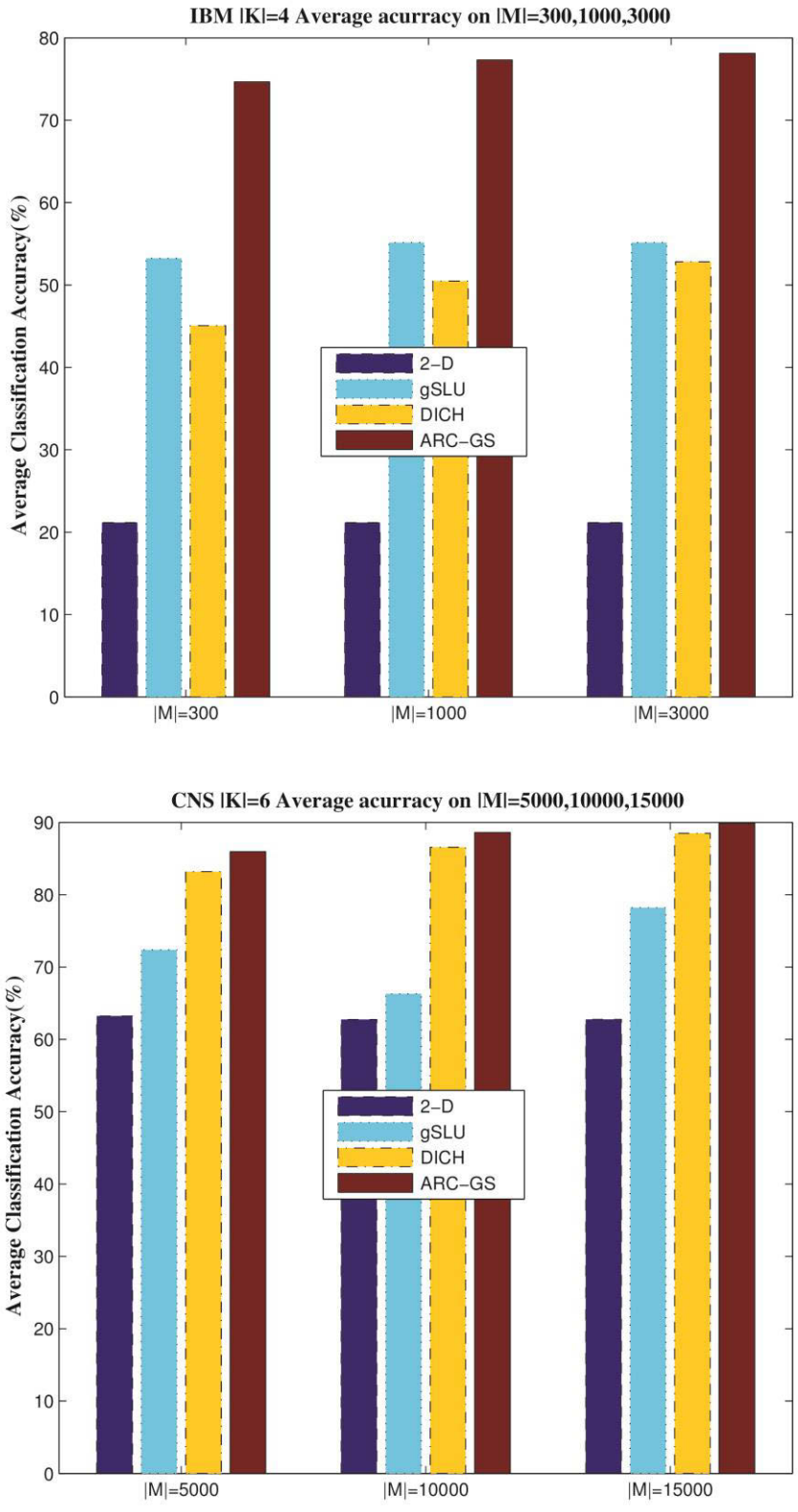
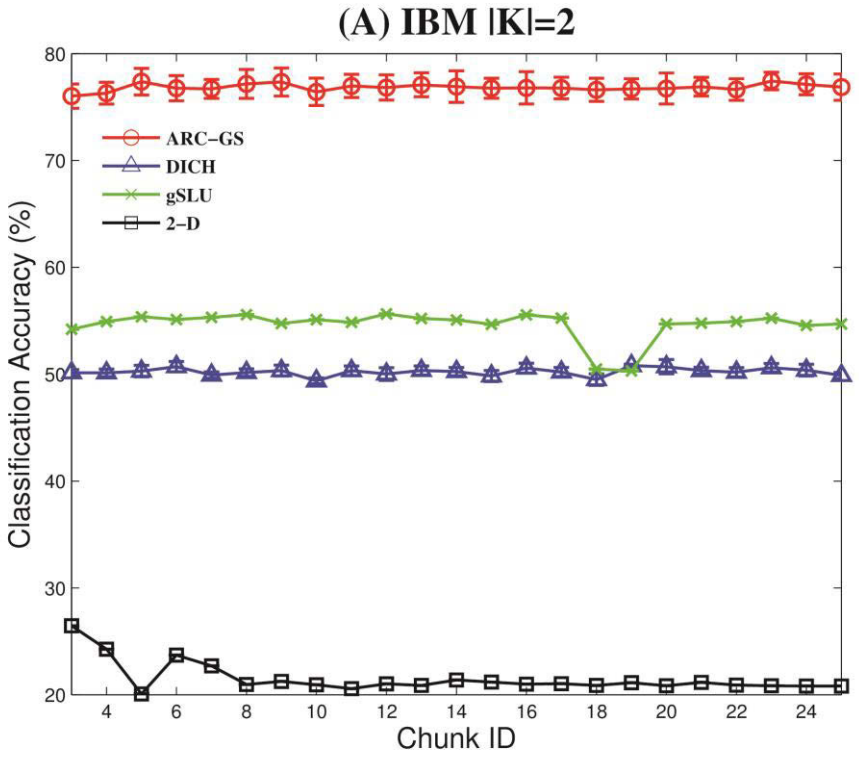
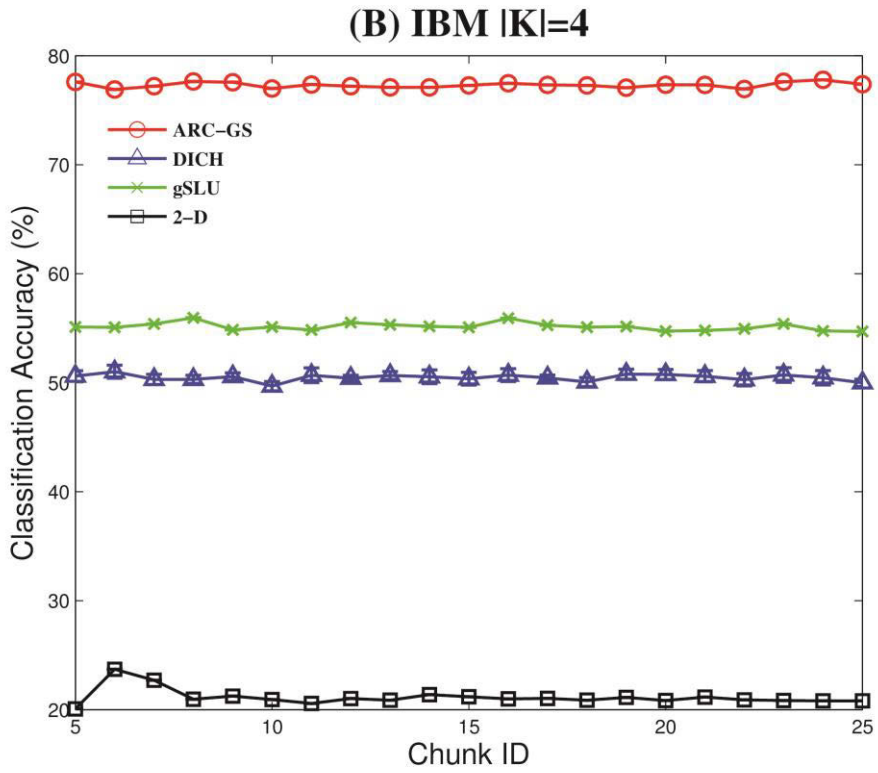
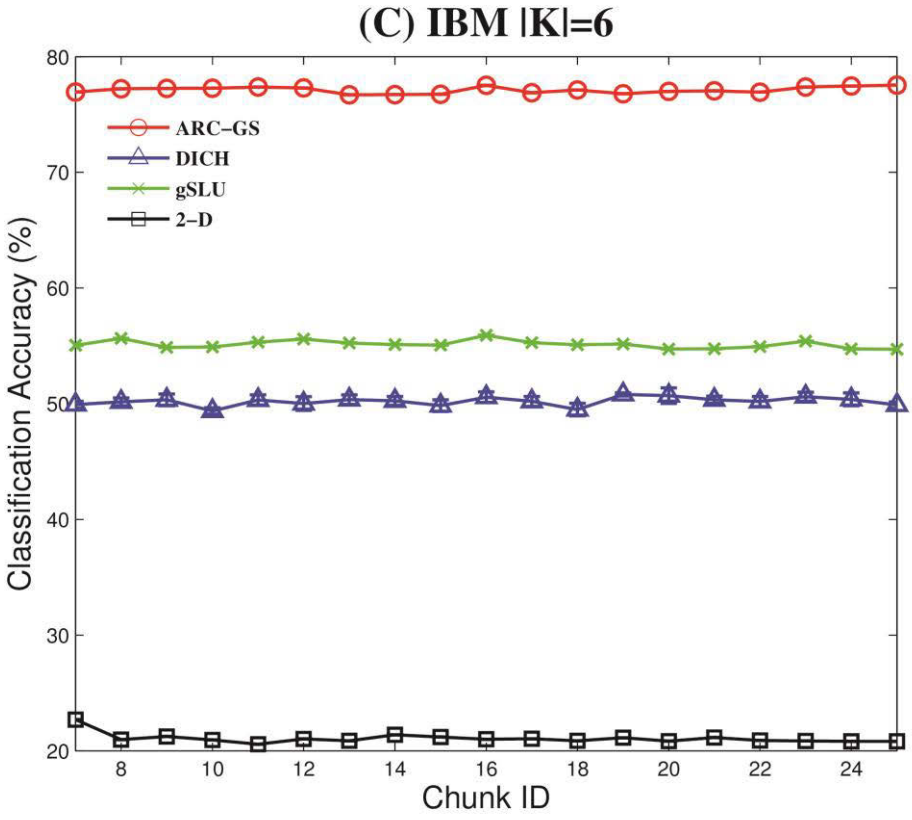


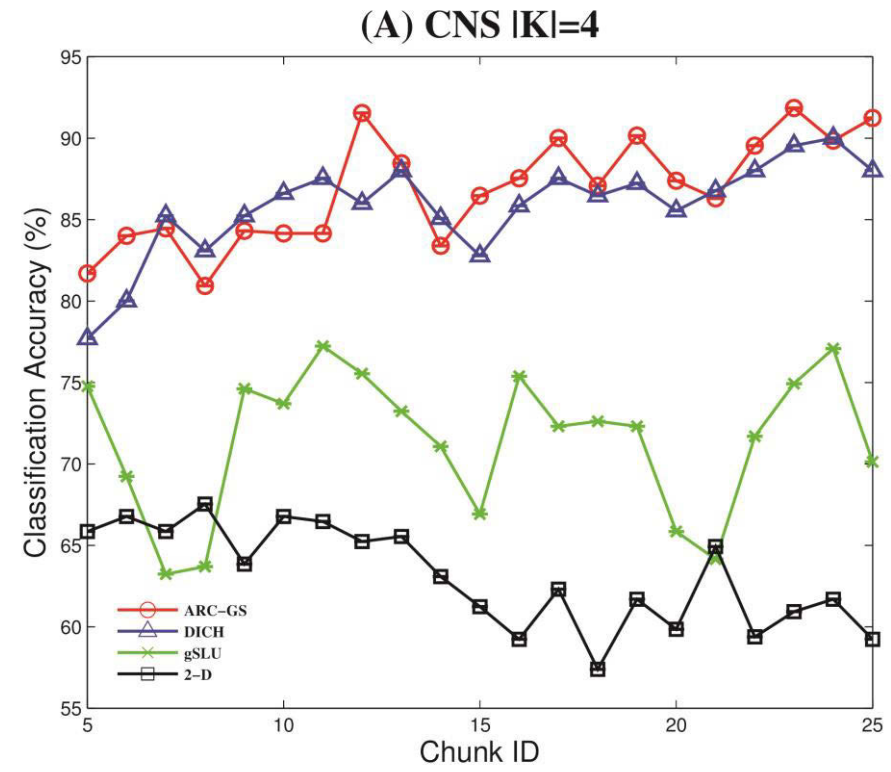
Figure 5.3: Average accuracy on the IBM (left, ensemble size $K = 4$), and the CNS (right, ensemble size $K = 6$) with different numbers of features M

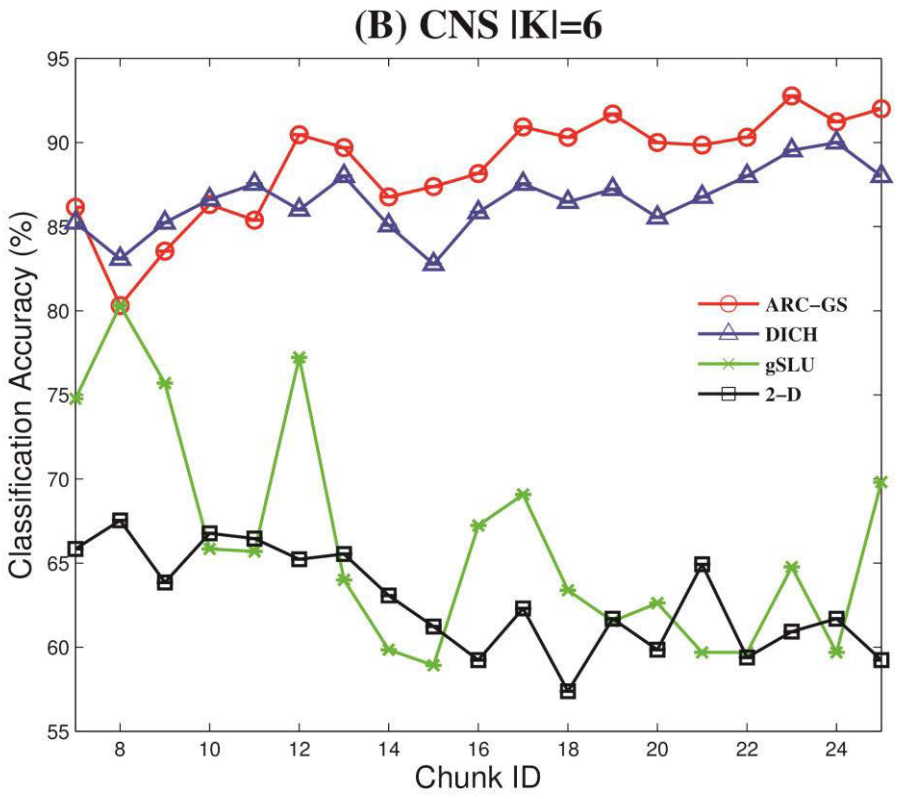
The average performance over the entire IBM and CNS in Fig. 5.5 further demonstrates that our ARC-GS classifier always outperform the DICH and gSLU classifiers and significantly outperform the 2-D hash compressed stream classifier, especially in the IBM Sensor stream. The reason is that a larger number of classes in the IBM can more obviously distinguish the performance of the classifiers than in the CNS. Hence, the results in the IBM can make the advantages of the ARC-GS more convincing. In the two streams especially in the CNS, for the ARC-GS and the DICH classifiers, we can find that a larger ensemble size can help improve the average classification accuracy. The reason may be that a larger ensemble size will make the ARC-GS and the DICH classifiers have more training graphs to generate more discriminative capability to classify graphs especially for the initial classification. However, for the 2-D classifier in the two streams, the fixed number of features may be insufficient for more training graphs to classify graphs, and a larger ensemble size can decrease the average accuracy. For the gSLU classifier in the CNS, the average accuracy is unstable under different ensemble size. The reason should be that the selected features in the gSLU are not stable for the training of new ensemble chunks.




(B) IBM |K|=4
ARC-GS
DICH
gSLU
2-D
Classification Accuracy (%)
Chunk ID
80
70
60
50
40
30
20
5
10
15
20
25


(C) IBM |K|=6
ARC-GS
DICH
gSLU
2-D
Classification Accuracy (%)
Chunk ID
80
70
60
50
40
30
20
8
10
12
14
16
18
20
22
24


(A) CNS |K|=4
Classification Accuracy (%)
Chunk ID
ARC−GS
DICH
gSLU
2−D


(B) CNS |K|=6
Classification Accuracy (%)
Chunk ID
ARC−GS
DICH
gSLU
2−D

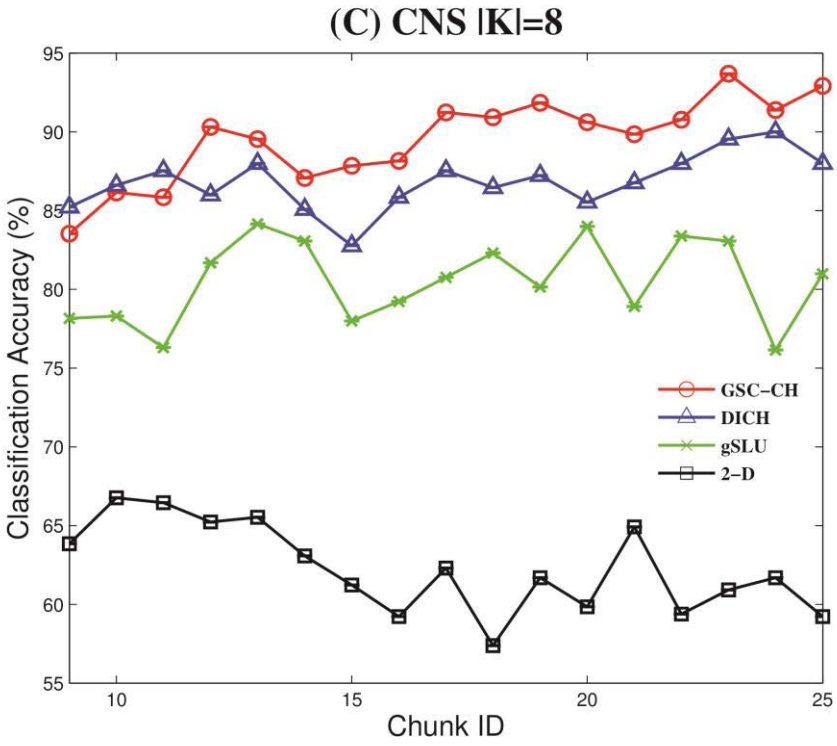
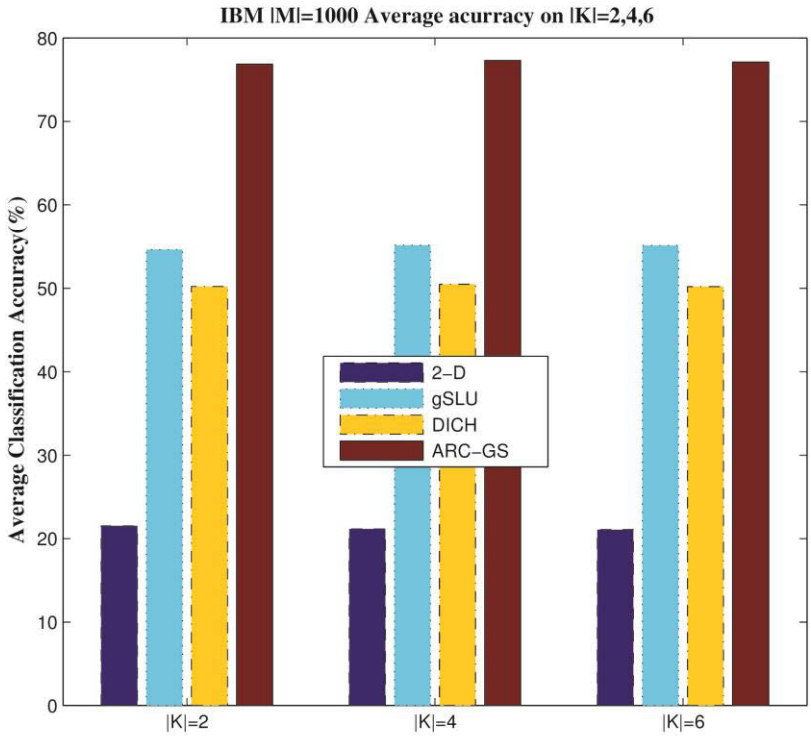


Figure 5.4: Classification accuracy on the IBM (number of features M = 1000), and the CNS (number of features M = 10000) with different ensemble size K



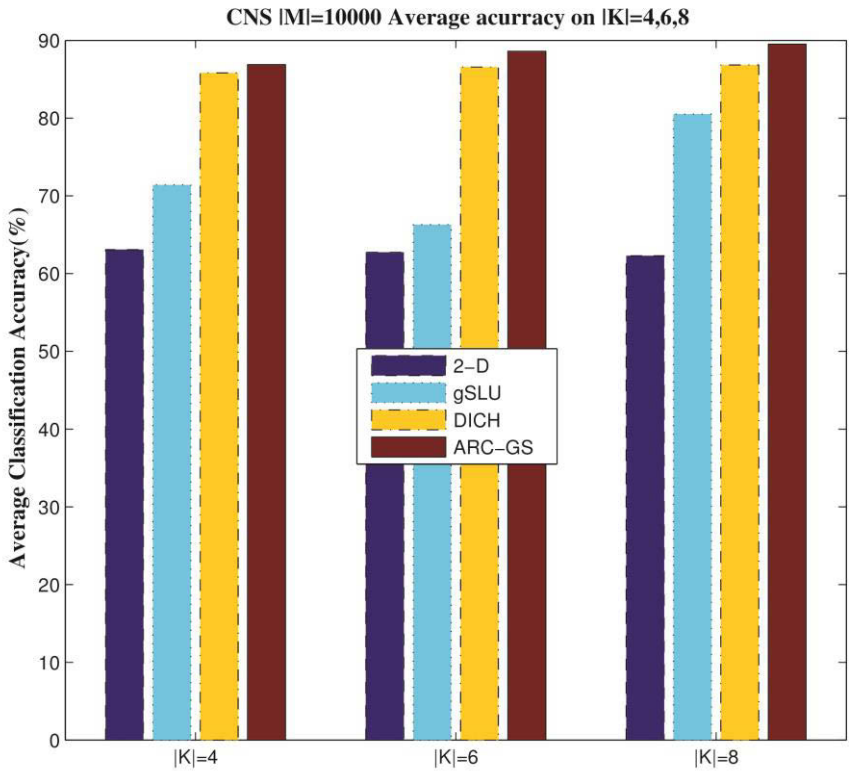
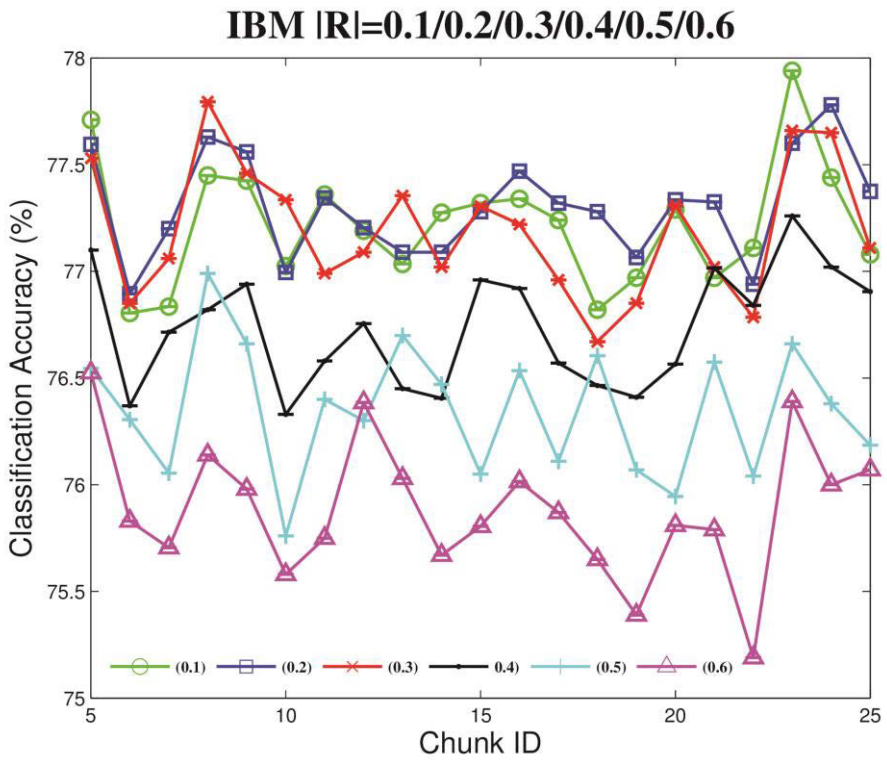
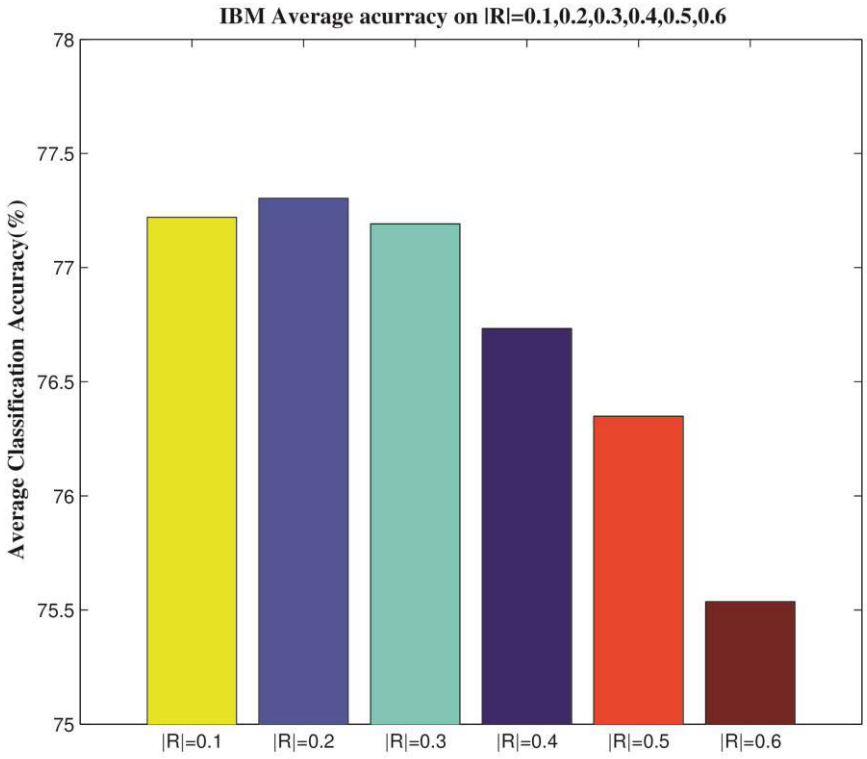


Figure 5.5: Average accuracy on the IBM (left, number of features M = 1000), and the CNS (right, number of features M = 10000) with different ensemble size K

Based on the overall effectiveness evaluation results, we can conclude that the proposed ARC-GS classifier can outperform the DICH and gSLU classifiers and significantly outperform the 2-D hash compressed stream classifier in classification accuracy.

Results w.r.t. the hash ratio R: In this experiment, for only ARC-GS classifier, we fix the number of features M ($M = 1000$ for the IBM, and $M = 10000$ for the CNS) and the ensemble size K ($K = 4$ for the IBM, $K = 6$ for the CNS), and adjust the hash ratio R for effectiveness evaluation. For both IBM and CNS, we investigate the hash Ratio R in $\{0.1,0.2,0.3,0.4,0.5,0.6\}$.


IBM |R|=0.1/0.2/0.3/0.4/0.5/0.6
Classification Accuracy (%)
Chunk ID
78
77.5
77
76.5
76
75.5
75
5
10
15
20
25
(0.1)
(0.2)
(0.3)
0.4)
(0.5)
(0.6)


IBM Average acurracy on |R|=0.1,0.2,0.3,0.4,0.5,0.6
Average Classification Accuracy(%)
78
77.5
77
76.5
76
75.5
75
|R|=0.1
|R|=0.2
|R|=0.3
|R|=0.4
|R|=0.5
|R|=0.6

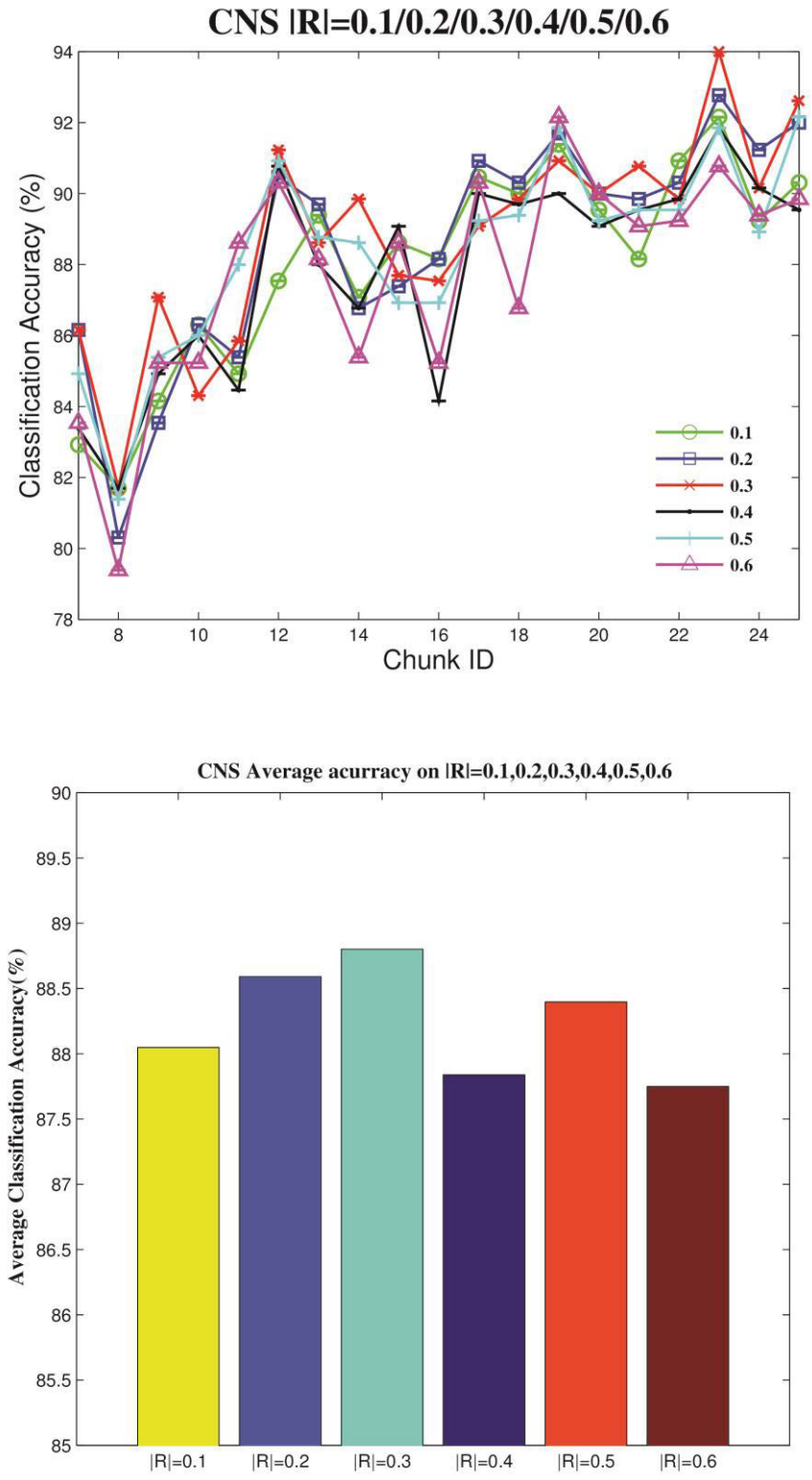


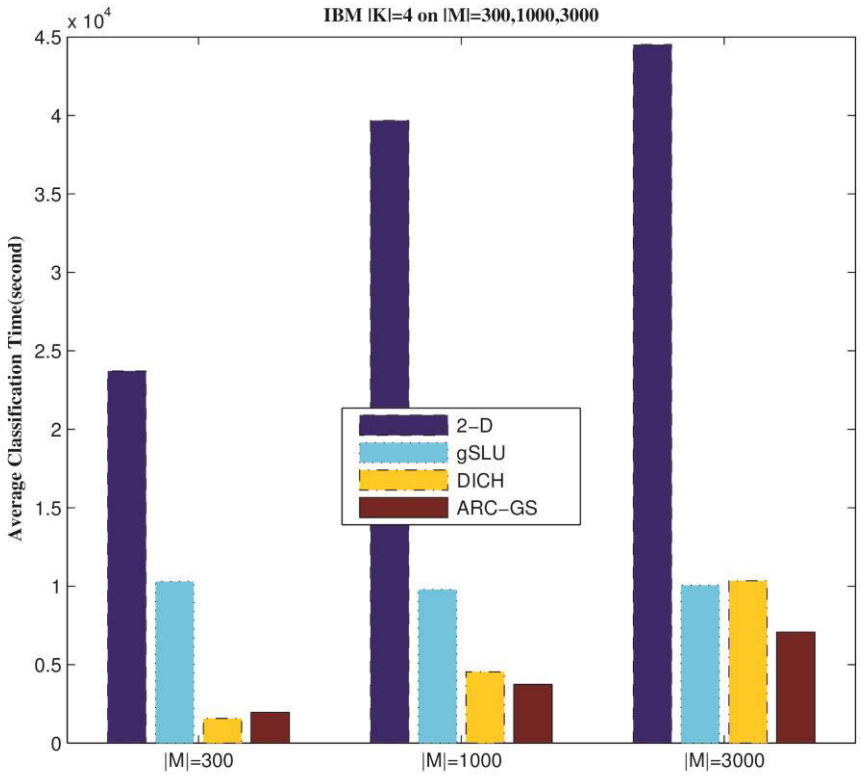
Figure 5.6: Classification accuracy and average classification accuracy on the IBM (upper row, number of features M = 1000 and ensemble size K = 4), and the CNS (bottom row, number of features M = 10000 and ensemble size K = 6) with different hash ratio R

Fig. 5.6 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. the Chunk ID ($\mathcal{X}$-axis) and average classification accuracy over the entire streams under different hash ratios. We can see that when the hash ratio R increases to 0.2 or 0.3, the overall classification performance of the ARC-GS classifier in the two streams is the best. Then, the classification accuracies will slightly decrease as R becomes larger especially in the IBM. Overall, the classification performance of the ARC-GS classifier is relatively stable for various hash ratios.

5.4.3 Efficiency Evaluation

In the following, we evaluate the efficiency of the four compared methods: the ARC-GS classifier, the DICH classifier, the gSLU classifier and the 2-D hash compressed stream classifier on the IBM and CNS streams.

Results w.r.t. the number of features M: In this experiment, we fix the ensemble size K ($K = 4$ for the IBM, $K = 6$ for the CNS) and adjust the number of features M for the efficiency evaluation. The experimental settings are the same as those in Fig. 5.7.



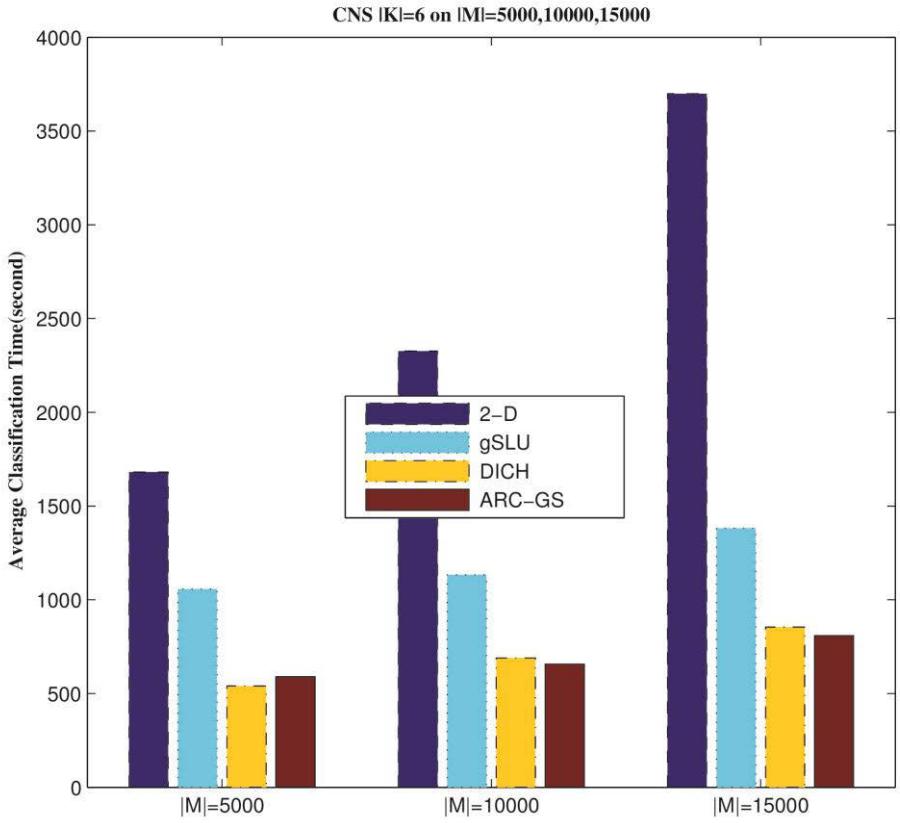


Figure 5.7: Average time on the IBM (up, ensemble size K = 4), and the CNS (down, ensemble size K = 6) with different numbers of features M

The average system runtime performance over the two streams is reported in Fig. 5.7. We see that the average time of both ARC-GS and DICH classifiers are less than the gSLU classifier and significantly less than the 2-D classifier. The reason is that an additional frequent pattern mining procedure in the 2-D is required to perform on the summary table which comprises massive transactions, and subgraph search process itself in the gSLU is slower than the clique search process in both ARC-GS and DICH classifiers. As the number of features increases, the average time of the four classifiers also increase accordingly. This is because that the learning process need take more time to manage more types of features. In the two streams, the overall average time of the ARC-GS classifier is close to the DICH classifier. As the number of features increases, the average time of the ARC-GS classifier increases more slowly than the DICH especially in the IBM stream. When the $M = 1000$ in the IBM and the $M = 10000$ in the CNS, the average time of the ARC-GS classifier is less than DICH classifier. We can

get that our ARC-GS classifier have better efficiency than the DICH classifier as the number of features increases. The reason is that the incremental stochastic learning strategy in the ARC-GS avoids the majority voting process that reduces the classification efficiency.

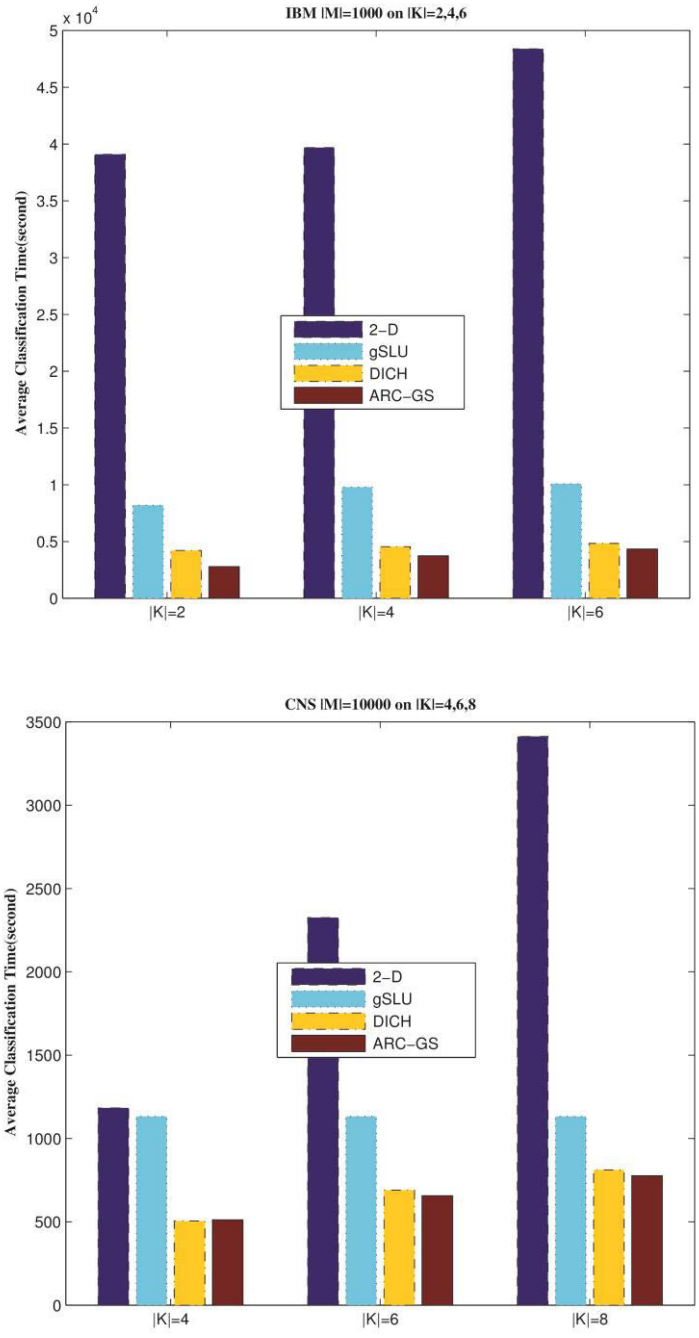


Figure 5.8: Average time on the IBM (up, number of features M = 1000), and the CNS (down, number of features M = 10000) with different ensemble size K

Results w.r.t. the ensemble size K: In this experiment, we fix the number of features M ($M = 1000$ for the IBM and $M = 10000$ for the IBM) and adjust the ensemble size K for the efficiency evaluation. The experimental settings are the same as those in Fig. 5.8.

Fig. 5.8 reports the average system runtime performance of the four classifiers over the two streams. Compared to the 2-D classifier, the ARC-GS, DICH and gSLU classifiers require significantly less time. We can see that the average time of the four classifiers increases as the ensemble size increases. This is because a larger ensemble size would result in more training graphs, which increase the training time accordingly. In the two streams, the overall average time of the ARC-GS classifier is close to the DICH classifier. As the ensemble size increases, the average time of the ARC-GS classifier increases more slowly than the DICH. Especially in the IBM stream, the overall average time of the ARC-GS classifier is always less than the DICH classifier. When the $K = 6$ in the CNS, the average time of the ARC-GS classifier is less than the DICH classifier. As the K increases, the average time of the ARC-GS classifier is much less than the DICH classifier. Therefore, our ARC-GS classifier has more stable and better efficiency than the DICH classifier as the ensemble size increases.

Overall, our ARC-GS classifier has the best efficiency among the four compared classifiers.

5.4.4 CONCEPT DRIFTS

In order to simulate the concept drifting, we consider adding the concept drifting chunks in our experimental graph stream. In this experiment, we use the IBM and GTGraph streams to compare the impact of the concept drifts on classification effectiveness of four classifiers. In the IBM stream, we change the class distribution in the concept drifting chunk to simulate the concept drifting. In the GTGraph stream, we simulate concept drifts through a parameter used to label the graphs in the stream.

- **The IBM Stream**: we select 50 different classes (1~50) of graphs from the whole IBM Sensor stream with 250 classes as our experimental data. In our IBM experimental stream, there are also total 25 chunks, and each of which comprises

20000 graphs. In the chunks 1~14 and chunks 18~25, the graphs are randomly selected from the IBM experimental data, and contains 50 classes. After analysing the overall class distribution in the chunks 1~14 and chunks 18~25, we insert the abrupt concept drift in the chunk 15 by designing a highly different class distribution. There are only two classes (5, 15) in the chunk 15, and the distribution ratio of these two classes is 1:1 (Class 5: 10000 graphs; Class 15: 10000 graphs). Then, we gradually changing the concept drift in the chunk 16 and chunk 17. In the chunk 16, there are two classes (1, 5), and the distribution ratio of these two classes is 1:1; in the chunk 17, there are three classes (1, 5, 15), and the distribution ratio of these three classes is 1:1:1.

- **The GTGraph Stream**: we create the synthetic GTGraph stream with drifting concepts through a parameter used to label the graphs in the stream. In the GTGraph network, we divide all nodes into d classes (d-dimensional space) and establish a hyperplane in d-dimensional space by equation:

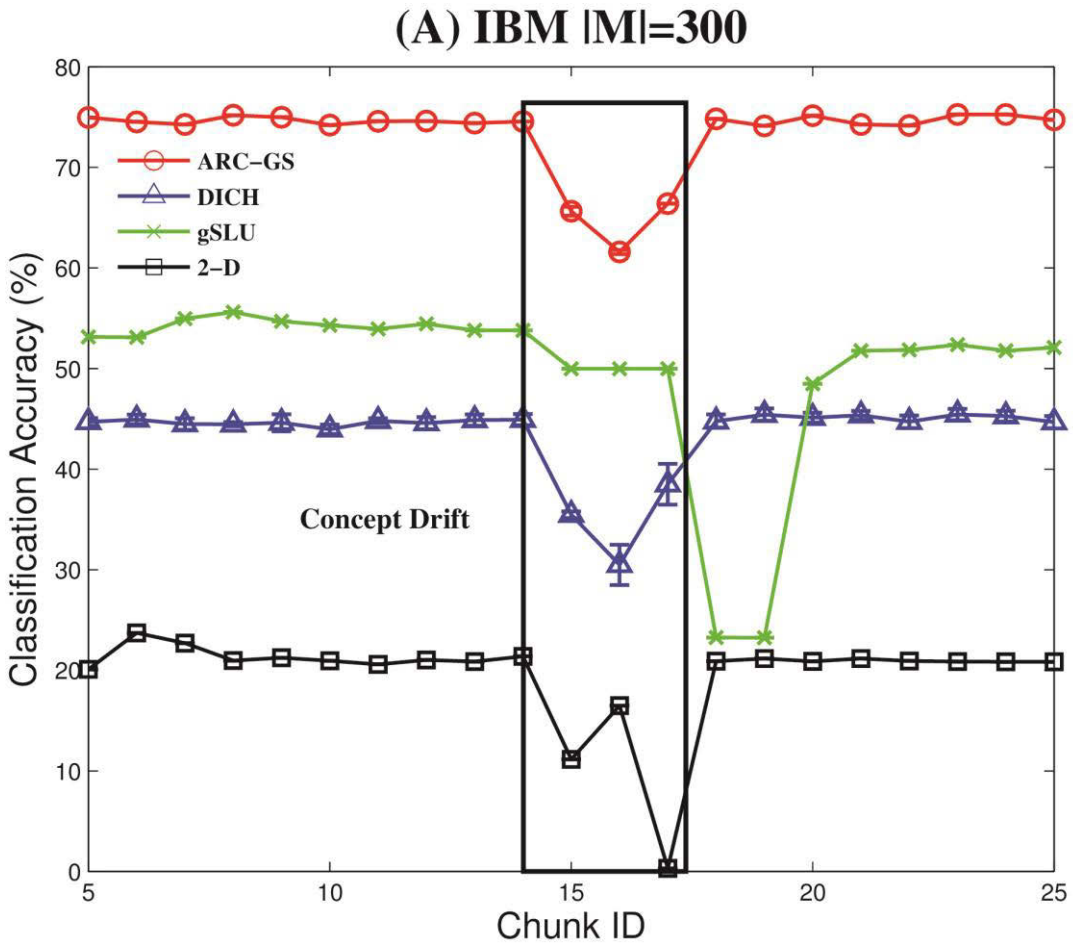
$$\sum_{i=1}^{d} f_i x_i = f_0 \tag{5.15}$$

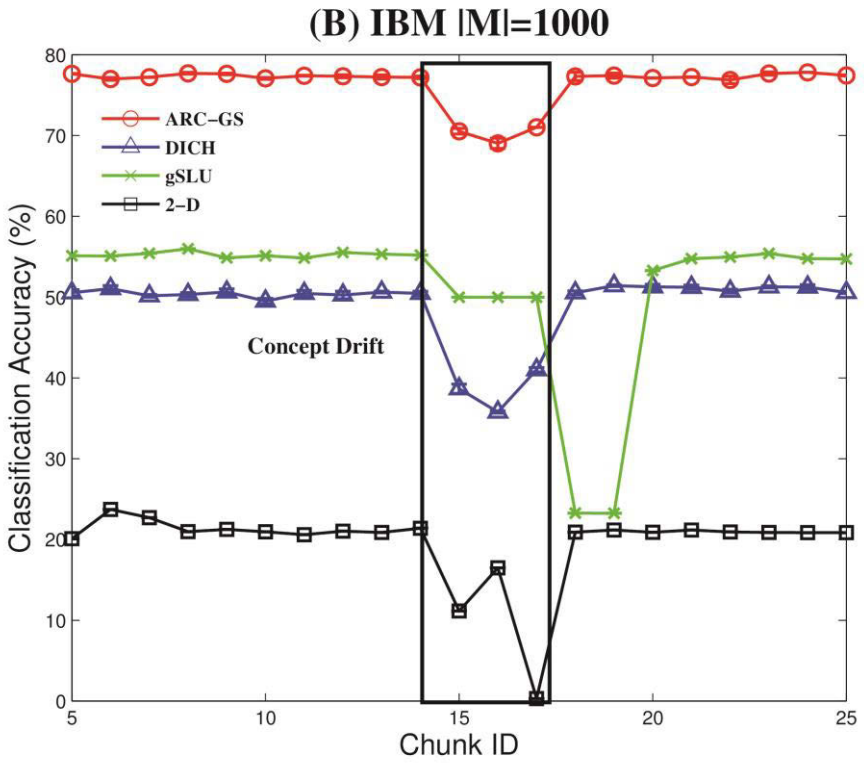
In this equation, the f_i denotes the i^{th} feature weight, and x_i denotes the number of nodes in the i^{th} feature in a graph. The feature weights $f_i(1 \leq i \leq d)$ are randomly initialized by the values in the range of [0, 1]. The f_0 is chose to cut the graphs into two parts, that is, $f_0 = \frac{1}{2}\sum_{i=1}^{d} f_i$. Thus, roughly half of graphs are labeled as positive, and the others are labeled as negative.

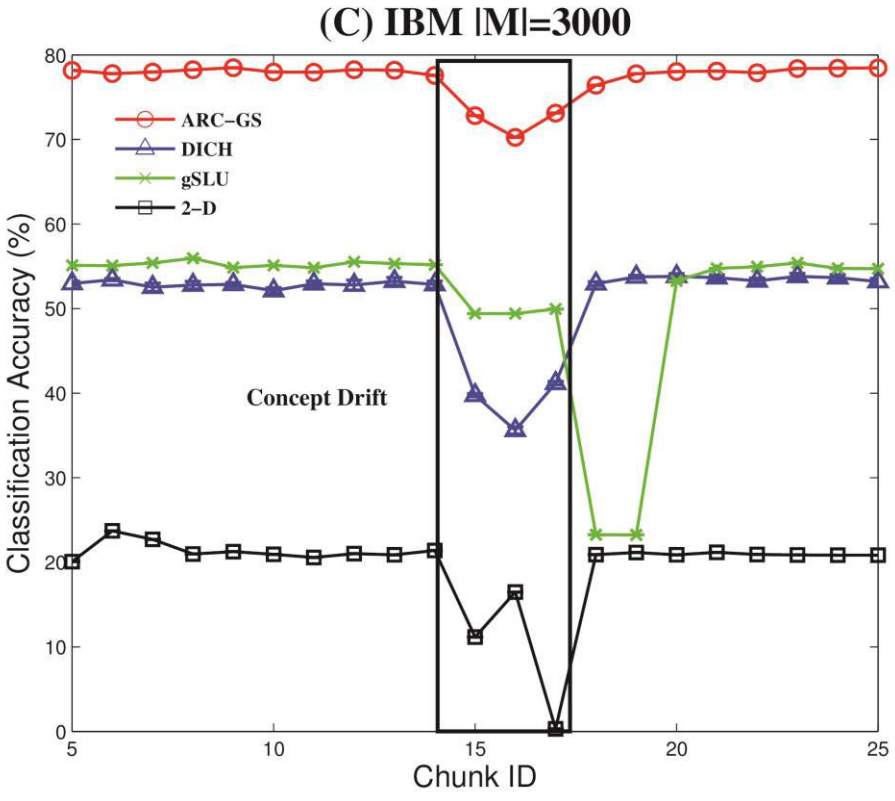
If the graph satisfies $\sum_{i=1}^{d} f_i x_i \geq f_0$, we label the graphs as positive; if the graph satisfy $\sum_{i=1}^{d} f_i x_i < f_0$, we label the graphs as negative. We simulate the concept drifts through the parameter f_0. In our examination, we set the d as 10, and insert the abrupt concept drift in the chunk 15 (the total number of chunks is 25) by adjusting the value of f_0 greatly, and then gradually adjust the value of f_0 to insert the gradual concept drifts in the chunk 16~18.

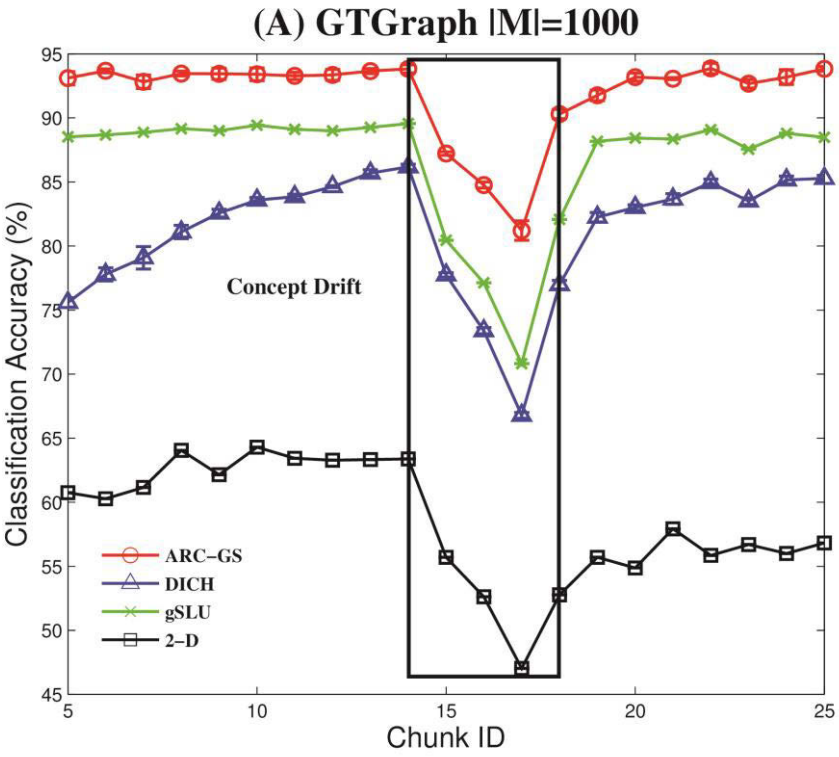
We also investigate the impact of the concept drifts on classification effectiveness of four classifiers in terms of 1) the number of features M and 2) the ensemble size K. In the ARC-GS classifier, we set the hash Ratio as 0.2.

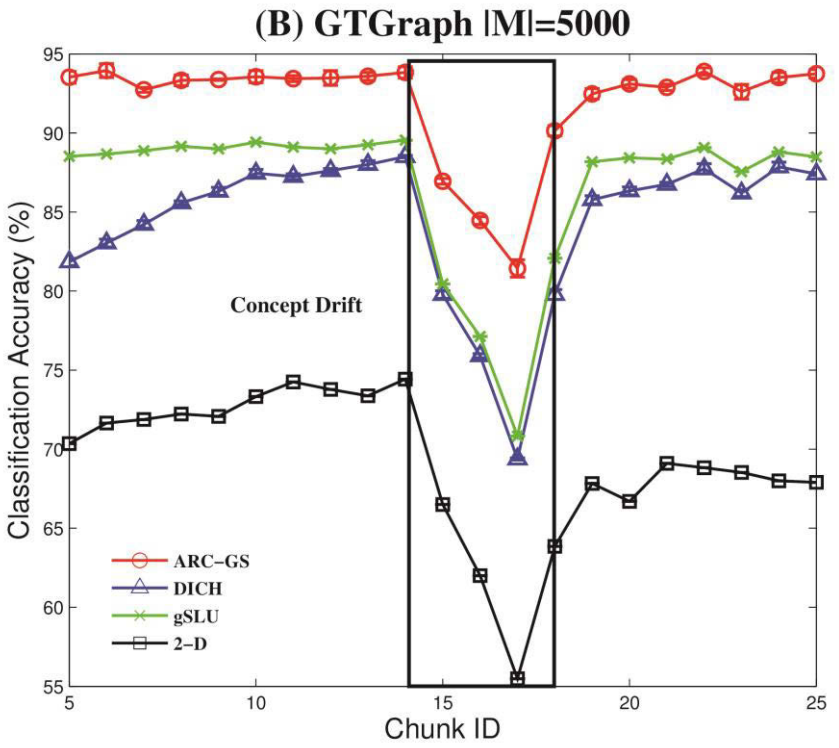
Results w.r.t. the number of features M: In this experiment, we fix the ensemble size K ($K = 4$ for both IBM and GTGraph) and adjust the number of features M for impact evaluation. For the IBM, we investigate the number of features M in $\{300,1000,3000\}$; for the GTGraph, we investigate the number of features M in $\{1000,5000,10000\}$.




(B) IBM |M|=1000
ARC−GS
DICH
gSLU
2−D
Concept Drift
Classification Accuracy (%)
Chunk ID
0
10
20
30
40
50
60
70
80
5
10
15
20
25


(C) IBM |M|=3000
ARC−GS
DICH
gSLU
2−D
Concept Drift
Classification Accuracy (%)
Chunk ID
0
10
20
30
40
50
60
70
80
5
10
15
20
25


(A) GTGraph |M|=1000
Classification Accuracy (%)
Chunk ID
Concept Drift
ARC−GS
DICH
gSLU
2−D
95
90
85
80
75
70
65
60
55
50
45
5
10
15
20
25


(B) GTGraph |M|=5000
Classification Accuracy (%)
Chunk ID
Concept Drift
ARC−GS
DICH
gSLU
2−D
95
90
85
80
75
70
65
60
55
5
10
15
20
25

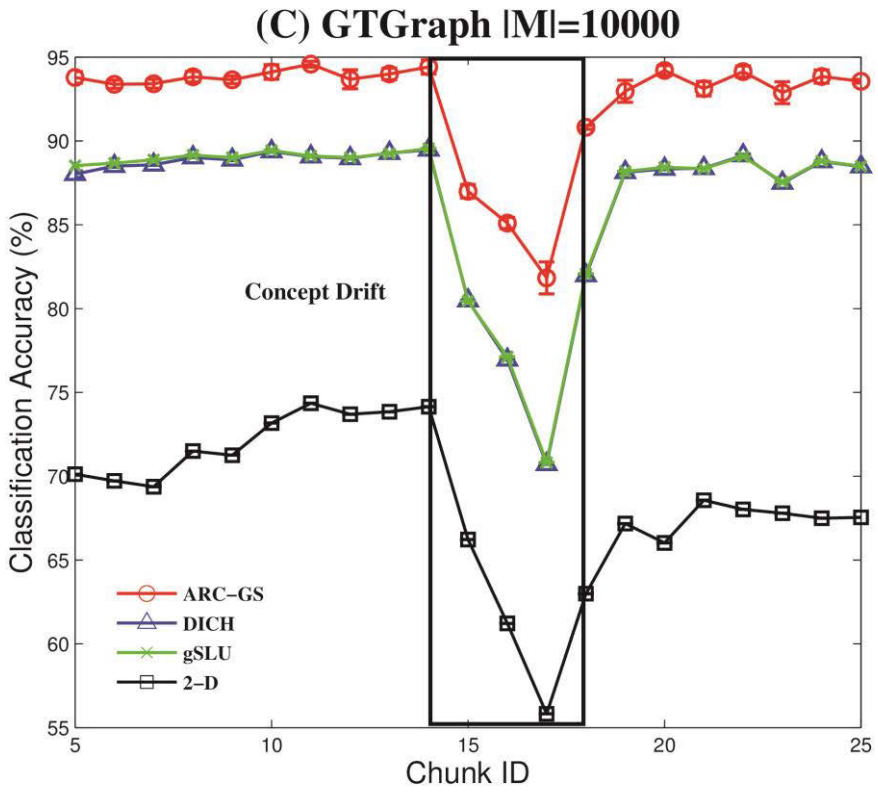


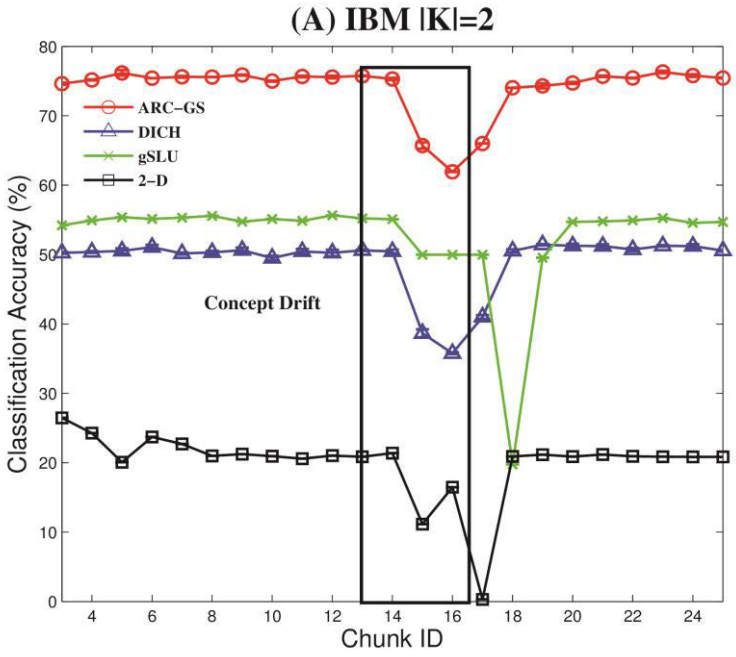
Figure 5.9: Classification accuracy on the IBM (ensemble size K = 4) and the GTGraph (ensemble size K = 4) with different numbers of features M

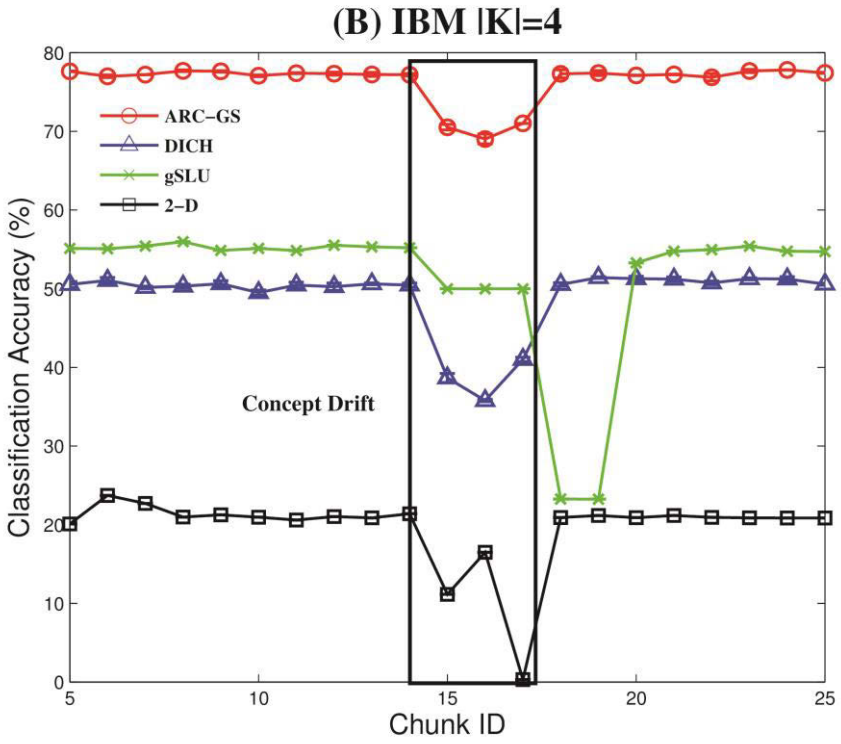
Fig. 5.9 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. the Chunk ID ($\mathcal{X}$-axis) in the IBM and GTGraph streams by using different numbers of features. We can see that there are noticeable concept drifting from chunks 14~19 in the GTGraph and chunks 14~18 in the IBM (marked by the rectangle boxes). As a result, all four classifiers experience performance loss. From chunks 14~15, in the GTGraph and IBM streams, all four classifiers experience large performance loss because there is an abrupt concept drift in the chunk 15. Then, except gSLU classifier, the performance loss becomes smaller because of the next gradual concept drifts. In the gSLU classifier, there exists larger performance loss after the concept-drifting chunks. The reason is that the instance weighting mechanism may be too sensitive to better adapt to the concept drifting, and the training results in the concept-drifting chunks mislead the classification of the following chunks. In all classifiers, our ARC-GS classifier receives less loss than the 2-D, gSLU and DICH classifiers especially in the IBM. We can see that the overall impact of the

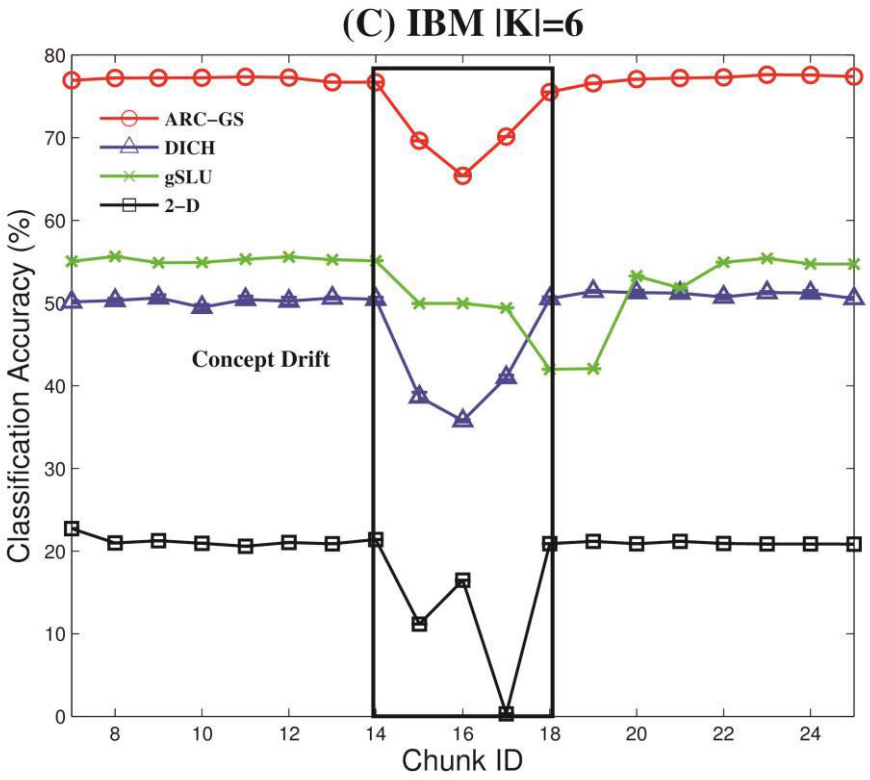
concept drifts on the classification performance of the ARC-GS classifier is minimal among the four compared classifiers on both two streams. As the number of features increases, the impact of the concept drifts on the classification performance of the ARC-GS classifier becomes less. However, for the DICH and 2-D classifiers, the impact almost remains the same. This experiment implies that our ARC-GS classifier can effectively handle the concept drifts.

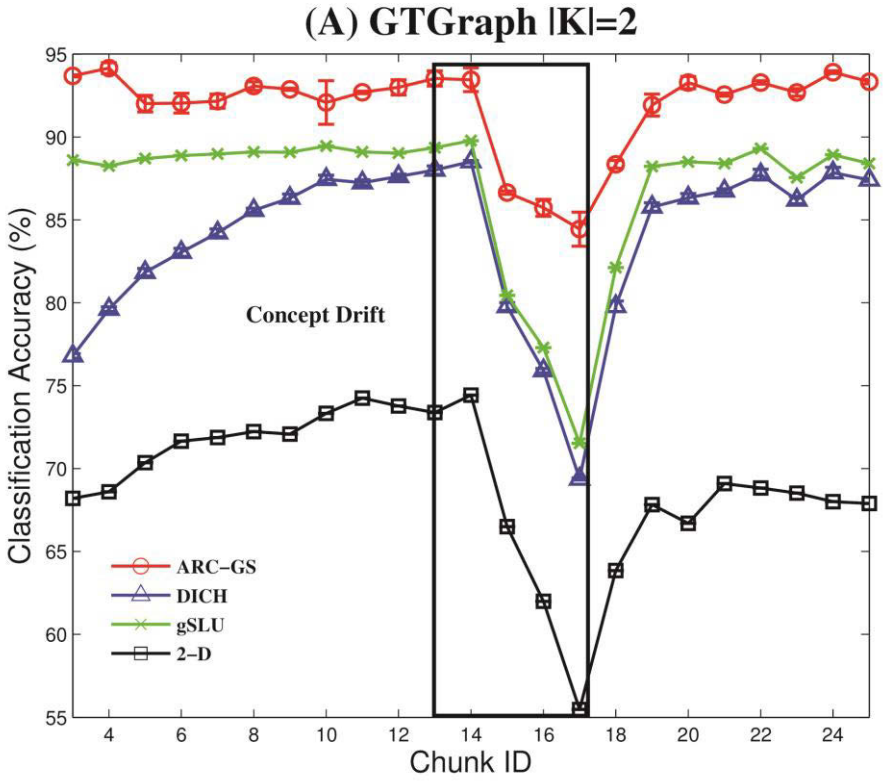
Results w.r.t. the ensemble size K: In this experiment, we fix the number of features M ($M = 1000$ for the IBM, $M = 5000$ for the GTGraph) and adjust the ensemble size K for impact evaluation of concept drifts. For the IBM and GTGraph, we investigate the ensemble size K in $\{2,4,6\}$.

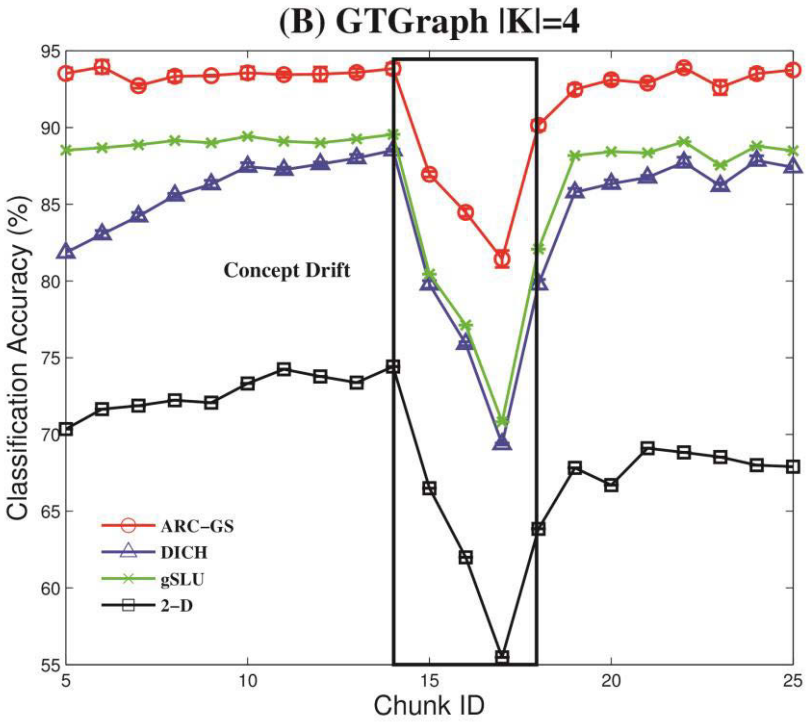
Fig. 5.10 plots the classification accuracy curves ($\mathcal{Y}$-axis) w.r.t. the Chunk ID ($\mathcal{X}$-axis) in the IBM and GTGraph streams by using different ensemble sizes. We also can observe that there are noticeable concept drifting from chunks 14~19 in the GTGraph and chunks 14~18 in the IBM (marked by the rectangle boxes). That is, all four classifiers experience performance loss. In all classifiers, we also can see that the overall impact of the concept drifts on the classification performance of the ARC-GS classifier is still minimal among the four compared classifiers on both two streams.




(B) IBM |K|=4
ARC−GS
DICH
gSLU
2−D
Concept Drift
Classification Accuracy (%)
Chunk ID
0
10
20
30
40
50
60
70
80
5
10
15
20
25


(C) IBM |K|=6
ARC−GS
DICH
gSLU
2−D
Concept Drift
Classification Accuracy (%)
Chunk ID
0
10
20
30
40
50
60
70
80
8
10
12
14
16
18
20
22
24


(A) GTGraph |K|=2
Classification Accuracy (%)
Chunk ID
Concept Drift
ARC−GS
DICH
gSLU
2−D


(B) GTGraph |K|=4
Classification Accuracy (%)
Chunk ID
Concept Drift
ARC−GS
DICH
gSLU
2−D

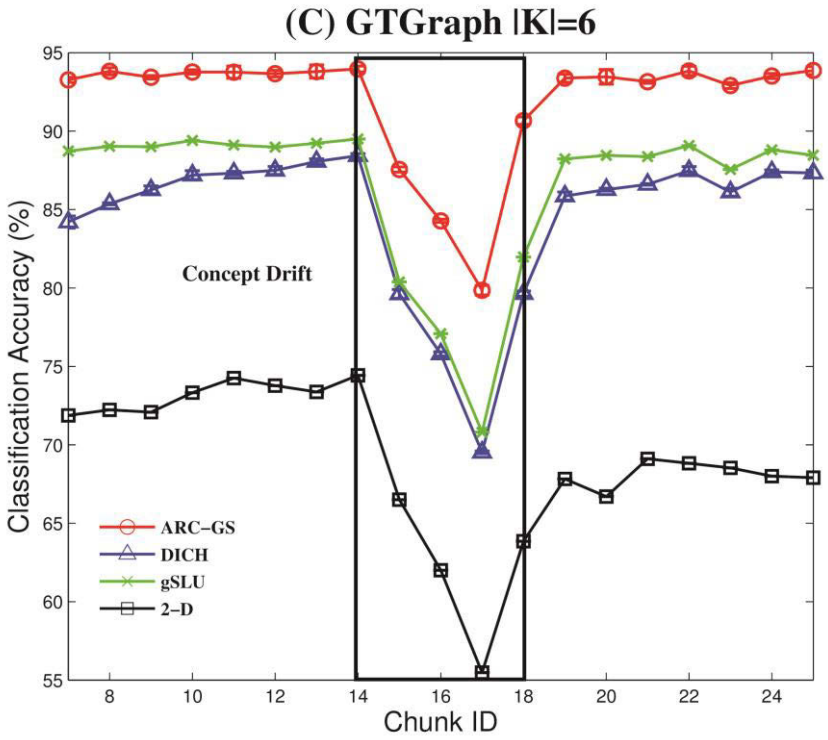


Figure 5.10: Classification accuracy on the IBM (number of features M = 1000) and the GTGraph (number of features M = 5000) with different ensemble size K

5.5 SUMMARY

This chapter proposes an adaptive real-time classification method for graph stream using two hashing schemes, incremental stochastic learning strategy and chunk level weighting mechanism to address the "real-time", "one-pass" and "concept drifting" challenges. In particular, we propose an approximate method for fast graph feature extraction by detecting cliques from the compressed graphs via hashing, which can significantly improve the efficiency of feature extraction to satisfy the "real-time" requirement. We also propose a graph feature reduction method by mapping unlimitedly expanding clique patterns onto corresponding fixed-size compatible feature spaces via differential hashing, which can avoid a pre-scan of graphs to address the "one-pass" and "concept drifting" challenges. Thanks to the clique hashing approach, the stream of graphs can be converted into feature vectors without additional parsing so that we can directly adopt a stochastic learning strategy to train a graph classifier online. Then, a

chunk level weighting mechanism is adopted to address "concept drifting" challenge. The experimental results on two real-world and one synthetic graph streams demonstrate that the proposed method can outperform the state-of-the-art method [9, 11, 12] in both classification accuracy and training efficiency; it also has an obvious advantage in handling concept drifting.

CHAPTER 6
CONTEXT-PRESERVING HASHING FOR FAST TEXT CLASSIFICATION

6.1 INTRODUCTION

In this chapter, we focus on the other popular and important structured data: text.

Mining of massive data [114] has become one of the most important research trends in the era of big data. The "3V" (volume, velocity and variety) nature of big data subverts the traditional learning paradigm because, in big data scenarios, the volume and the dimensionality of instances are usually unpredictable and increase rapidly. In such cases, even enumerating complete features to compute a similarity has become an intractable problem. For example, in document similarity search, the underlying feature space can easily exceed 108 dimensions if we consider 5-shingles (5 continuous characters) [114]; and the feature space can be much higher if we consider a vocabulary of words as features. Thus, it becomes urgent to develop fast approximate algorithms to address the storage and computation problems for big data.

There have been a number of approximate algorithms for big data similarity computation. Since many high-dimensional data can be represented as bags of words, min-wise hashing [10] has been naturally applied to them for fast approximating set similarities without scanning and comparing the complete sets. Recently, [114] further improves the efficiency of min-wise hashing by storing only the lowest b bits of each hashed value. Random projection [112, 107] was proposed to randomly project high-dimensional data onto low-dimensional spaces. For sketching streaming data, count-min

sketch [110] was developed to estimate feature occurrences. Recently, feature hashing [115, 116] was employed to estimate inner products of high-dimensional feature vectors. All these approximate algorithms have been found very effective in certain big data problems.

However, all the aforementioned approximate algorithms are based on the bag-of-words representation for its exchangeability that can facilitate random projection and hashing. A limitation of such *flat-set* representation is that context information and semantic hierarchy may be lost. For example, in Fig. 6.1, the second text is an abstract of a paper on transfer learning, which first introduces an application background (including Web media terms) and states the underlying learning problem next (including machine learning terms). If we represent the abstract as a flat set, it will resemble the first text, which is a technical blog also comprising Web media and some technical terms, but in different context. Thus, a more expressive bag-of-words representation needs to be explored to relieve this problem.

Texts from Google	Bag-of-Words	Nested Bag-of-Words
1. Technical blog about Google Maps Embed **video** and image in **Google** maps for multimedia **news** ... www.mulinblog.com/.../embed-**video**-and-images-in-**google**-maps-guide... Mar 6, 2013 - Embed **video** and images in **Google** maps: A beginner's guide for **multimedia** ... Using **Flickr** and **Youtube** to host your photos and **video**, you can add ... Here's how to embed the photo: when **viewing** your photo on **Flickr**, click	{embed video image Google map beginner guide multimedia using Flickr Youtube host photo video add view}	{{embed video image Google map beginner guide multimedia} {use Flickr Youtube host photo video add} {embed photo view Flickr click}}
2. Abstract of a SDM-13 paper on Transfer Learning **Multi**-Transfer: **Transfer Learning** with **Multiple** Views and **Multiple** ... knowledgecenter.siam.org/190SDM/ by B Tan - Related articles FM and **Google News**. **different** vocabularies of **google news**. ... on **Youtube**, actually proposed to place **transfer learning** under the **multi**- the piece of ... as **Transfer Learning** with **Multi**- **transfer learning** problem where **source** and ... FM and im- lem, **transfer learning** aims to borrow knowledge from ages from **Flickr** may have ...	{FM Google news different vocabulary available Youtube actually propose place transfer learning multi problem source video analysis describe complement}	{{FM Google news different vocabulary} {available Youtube actually propose place transfer learning multi} {problem transfer learning multi source} {video analysis describe source complement}}
3. Abstract of a SDM-13 paper on Transfer Learning On Handling Negative **Transfer** and Imbalanced **Distributions** in ... **knowledge**center.siam.org/50SDM/ by J Gao - Related articles As there are usually **multi**- and sentiment classification [12] demonstrate the power ple ... **knowledge** can be **transferred**, of **transfer** learning, **multiple source transfer** ... In this paper, we **propose** methods to many applications due to the existence of a **novel two-phase framework** to effectively **transfer** knowl- irrelevant **sources** ...	{usually multi sentiment classification demonstrate power knowledge transfer learning multiple source paper propose method application existence novel two-phase framework effectively irrelevant}	{{usually multi sentiment classification demonstrate power} {knowledge transfer learning multiple source} {paper propose method application existence novel two-phase framework effectively transfer irrelevant source}}
	Sim(1,2) > Sim(2,3) > Sim(1,3)	Sim(2,3) > Sim(1,2) > Sim(1,3)

Figure 6.1: Motivation examples. The standard min-wise hashing on bags-of-words (flat-sets) gives sim (1, 2) > sim (2, 3) while our RMH on nested bags-of-words (nested-sets) gives sim (2, 3) > sim (1, 2)

In this chapter, we aim to fast compute similarities between bag-of-words represented objects while also preserving context information inside the objects. We still follow the random algorithm approach to this end. We don't consider relational learning or structural patterns to capture context information which might be unrealistic in big data

scenarios. To take into account semantic hierarchy, we consider a notion of *multi-level exchangeability* which can be applied at word-level, sentence-level, paragraph-level, etc. We employ a nested-set to represent a multi-level exchangeable object, say "nested bag-of-words". For example, we can let $\{\{a, b, c, d\}, \{b, d, e\}, \{e, f, g\}\}$ represent a paragraph with three sentences, each of which further comprises several words. In this example, the top-level exchangeable elements are sentences while the bottom-level exchangeable elements are words. In such nested-set representations, context information and semantic hierarchy are preserved yet the resulting form is still simple for random algorithms.

To fast compute a similarity between nested-sets, we propose a Recursive Min-wise Hashing (RMH) algorithm for sketching nested-sets. The advantage of RMH is two-fold: 1) Account for multiple levels of exchangeabilities; 2) Enable a probabilistic comparison of sub-sets instead of hard matching. By virtue of RMH, we can compare two multi-level exchangeable objects with the same computational cost of the standard min-wise hashing algorithm while preserving context information as a plus. We also provide a theoretical bound to RMH to show it is a highly-concentrated estimator. We conduct empirical studies on three real-world text data sets (DBLP paper abstracts, IMDB movie reviews, and Amazon product reviews). The experimental results from three text classification tasks show that the proposed context-preserving hashing method can significantly outperform both min-wise hashing [10] and feature hashing [116] in accuracy at the same (even less) computational cost.

The remainder of the chapter is organised as follows. The preliminary knowledge and baselines are introduced in Section 6.2. The RMH algorithm (Recursive Min-wise Hashing) is presented in Section 6.3. A case study is then given in Section 6.4 to test our RMH algorithm. Finally, a summary of this chapter is given in Section 6.5.

6.2 PRELIMINARIES & BASELINES

6.2.1 PRELIMINARIES

- **Bag-of-Words**: A bag of words is the unordered collection of words in a text d. It is a simplified representation disregarding the structural information. Due to its simplicity, bag-of-words has been accepted as a standard model in information retrieval and text mining, especially in massive data scenarios. For text classification, there are two commonly used bag-of-words representations: (1) *Term Frequency (TF)*, which counts the occurrence of each word in d and let the counting be the value of the corresponding feature dimension. The resulting form is a feature vector $x \in R^V$ whose dimensions are spanned by V terms in a predefined vocabulary. It is common to use inverse document frequency (IDF) to weight TF for emphasizing uncommon terms [114]. (2) *(Multi-) Set*, which views all words in d as a set S; if a same word is allowed to appear multiple times, it is a multi-set. This representation is easier than TF since no predefined vocabulary (feature space) is required, hence it is more popular in high-dimensional data scenarios.

- **Min-wise Hashing**: The min-hash scheme [10] is an approximate method for measuring the similarity of two sets, say S_a and S_b. K hash functions (random permutations) $\{\pi_k\}_{k=1}^{K}$ are applied to the elements in S_* and we say $\min(\pi_k(S_*))$ is a min-hash of S_*. A nice property of min-hash is that the probability of S_a and S_b to generate the same min-hash value is exactly the Jaccard similarity of S_a and S_b:

$$Pr(\hbar(S_a) = \hbar(S_b)) = \frac{|S_a \cap S_b|}{|S_a \cup S_b|} = J(S_a, S_b) \tag{6.1}$$

where we write $\hbar(\cdot) = \min(\pi(\cdot))$ as a shorthand. In practice, multiple independent random permutations are used to generate min-hashes to approach the expected probability. The similarity between the two sets based on the K min-hashes is calculated by

$$\hat{J}(S_a, S_b) = \frac{\sum_{k=1}^{K} 1(\hbar(S_a) = \hbar(S_b))}{K} \tag{6.2}$$

where 1(state) = 1, if state is true; and 1(state) = 0, otherwise. As $\rightarrow \infty$, $\hat{\mathrm{J}}(\mathrm{S_a}, \mathrm{S_b}) \rightarrow \mathrm{J}(\mathrm{S_a}, \mathrm{S_b})$; that is

$$E[1(\hbar_k(S_a) = \hbar_k(S_b))] = J(S_a, S_b) \quad (6.3)$$

The proposed Recursive Min-wise Hashing algorithm in Section 4 can be viewed as a generalization of the min-hash scheme [10] for sketching nested sets.

- **Feature Hashing:** Feature hashing [116] provides an unbiased and highly-concentrated estimator of the inner product of high-dimensional feature vectors. It is closely related to the random projection [112, 107]. The difference is that the projection matrix $\boldsymbol{P}$ only comprises values in $\{1, -1\}$, i.e., $\boldsymbol{P} \in \{1, -1\}^{K*V}$, where V is the original dimensionality and K is the new, $K \ll V$. A constraint on $\boldsymbol{P}$ is that each column is allowed to have only one non-zero entry. The positions of non-zero entries and its signs are randomly generated. Given a feature vector $x \in R^V$ (e.g., TF), the hashed feature vector gives $\bar{x} = \boldsymbol{P}x \in R^K$. The intuition of this operation is to randomly partition the features into groups and sum up the signed features in the same group, where the sign is added to eliminate bias (a biased version without signs is [115]).

  In practice, it is not necessary to explicitly define the projection matrix. Two random hash functions can be directly applied to the terms $\{\mathrm{w}_1, \mathrm{w}_2, \dots\}$ in d to calculate the hashed TF feature vector

$$\bar{x} = [\bar{x}_1, \dots, \bar{x}_K]^T, and\ \bar{x}_K = \sum_{i:\xi(w_i)=k} x_i\, \delta(w_i) \quad (6.4)$$

  where ξ: str $\mapsto \{1, \dots, \mathrm{K}\}$ and δ: str $\mapsto \{1, -1\}$ are two random hash functions. Due to its implicitly projection property, feature hashing is extremely useful in big data scenarios where data may have infinite features. It has been adapted to many applications, such as multi-task learning [116], collaborative filtering [113], and graph stream classification [17].

6.2.2 BASELINES

Three baseline methods based on the above building blocks are listed below as the compared methods in the experiment section.

- **TF:** Each text is represented as a feature vector $x_n \in R^V$, whose components are term frequencies (TF). A linear classifier is applied to $\{x_n\}_{n=1}^N$.
- **FHTF:** This method uses the feature hashing technique [116] introduced above to implicitly map $x_n \in R^V$ to $x_n \in R^V, K \ll V$. Then a linear classifier is applied to $\{\bar{x}_n\}_{n=1}^N$.
- **MinHash:** This method represents each text as a set of terms (bag-of-words) S_n and uses the min-wise hashing scheme [10] introduced above to hash S_n into a fingerprint $h_n \in Z^K$. A classifier based on Hamming distance is applied to $\{h_n\}_{n=1}^N$.

6.3 RMH: RECURSIVE MIN-WISE HASHING

The main components of the RMH are described in detail as follows.

6.3.1 MULTI-LEVEL EXCHANGEABLE REPRESENTATIONS

As aforementioned, bag-of-words is widely accepted for representing a text due to its simplicity and conciseness. However, a flat enumeration of words might be inappropriate because: (1) Context information and semantic hierarchy are lost due to the shuffling of words from different parts of a text. (2) Some minor semantic parts of a text may be overlooked if the random samples are not sufficient to cover the entire text.

We propose an alternative way to represent a text using a nested set. A nested set is a set that contains non-trivial sets as elements, in contrast to a flat set that only contains atomic elements. For example, $\{\{a,b,c,d\},\{b,d,e\},\{e,f,g\}\}$ is a nested set where the top-level set comprises three non-trivial sets corresponding to three sentences; and each bottom-level set contains several words as atomic elements. If we represent the same text in a flat set, we get $\{a,b,c,d,e,f,g\}$ with seven words as atomic elements. Based on requirements, the nested-set representation for a text can have multiple levels of

exchangeabilities, where the bottom-level sets comprise words, the second-level sets comprise sentences, the third-level sets comprise paragraphs and so the fourth. In doing so, *we can roughly preserve the context information and avoid missing samples in minor semantic parts*.

To use a nested set to represent a text, we first need to customize the considered exchangeable levels, for example, paragraph-level, sentence-level, and word-level, where paragraph is the highest level and word is the lowest level. Then we recursively construct the nested set using lower-level set-elements until reaching the words as atomic elements.

Definition 6.1 *(Multi-Level Exchangeable Representations) The multi-level exchangeable representation of a text is a nested set* $S^{(R)} = \{S_1^{(r-1)}, \ldots, S_{|S_*^{(r)}|}^{(r-1)}\}$, *where*

$$S_*^{(r)} = \{S_1^{(r-1)}, \ldots, S_{|S_*^{(r)}|}^{(r-1)}\} \tag{6.5}$$

for $r = 2, \ldots, R-1$. $S_*^{(1)}$ *denotes a set of words as atomic elements and* $S^{(R)}$ *denotes a set of the highest-level exchangeable objects (e.g., paragraphs).*

We take the first example in Fig. 6.1 to illustrate the concept of multi-level exchangeable representation: The text is a short segment of a blog, which is partitioned into three (incomplete) sentences by "...". We consider two levels of exchangeabilities, word and sentence, to construct a nested set for the text. It is first represented as a set of sentences as $S^{(2)} = \{S_1^{(1)}, \ldots, S_3^{(1)}\}$, each set-element of which is further represented as a set of words. The steps of this nested-set construction procedure are listed in Algorithm 1 (Lines 1–5). The argument in $strhash(term, \infty)$ means that the hash keys are sufficient to avoid collisions. The obtained nested set $S^{(R)}$ is fed into the recursive min-wise hashing procedure introduced in the next section.

Algorithm 1 Context-Preserving Fingerprinting

Input: d: a text; $\left\{\pi_k^{(r)}\right\}_{r=1,k=1}^{R,K}$: K hash functions (random permutations) at the rth level, where R is the number of levels in the nested set.

Output: $\boldsymbol{h}^{(R)}$: the fingerprint of d.

1: **for** $term \in d$ **do**

2: $tokem \leftarrow strhash(term, \infty)$;

3: Replace $term$ with $token$ in d ;

4: **end for**

5: Replace d as a nested set $\boldsymbol{S}^{(R)}$based on Eq. (6.5) ;

6: $\boldsymbol{h}^{(R)} \leftarrow rmh\left(\boldsymbol{S}^{(R)}\right), \left\{\pi_k^{(r)}\right\}_{r=1,k=1}^{R,K}$)

6.3.2 RECURSIVE MIN-WISE HASHING

After obtaining the nested-set representation of a text, the next key step is to design an approximate algorithm to sketch the nested set for fast comparison. The nested-set sketching algorithm is expected to have the same properties as the min-wise hashing algorithm: 1) efficient in both time and space for massive data mining, 2) compact to represent an object with enormous features, and 3) locality-preserving such that the estimated similarity is consistent with the value computed by direct comparing nested-sets.

In the following, we propose a Recursive Min-wise Hashing (RMH) algorithm to estimate the similarity between nested sets. In contrast to the hard matching of elements in the standard min-hash scheme, a similarity metric that can measure a soft matching of set-elements (i.e., non-atomic elements in a nested set) is required. We require "soft matching" here since set-elements may have overlaps and we cannot simply view them unmatched if only a fraction of its elements (which may also be set-elements) are different. The similarity metric is expected to satisfy the following two requirements:

R.6.1 *Set-elements in nested sets can be compared in probability instead of hard matching.*

R.6.2 *Probabilistic similarities of lower-level sets can be propagated to higher-level sets.*

One possible way to measuring similarity between two nested sets, $S_a^{(r)}$ and $S_b^{(r)}$, is to first hash each lower-level set $S_*^{(r-1)}$as a fingerprint $\boldsymbol{h}_*^{(r-1)}$, which has L dimensions. We can rewrite Eq. (6.5) as

$$S_*^{(r)} = \{\boldsymbol{h}_1^{(r-1)}, \ldots, \boldsymbol{h}_{\left|S_*^{(r)}\right|}^{(r-1)}\} \tag{6.6}$$

Here we use $\doteq$ because the equation only holds as $L \to \infty$. Now we have represented a nested set as a set of L-dimensional fingerprints. Suppose these fingerprints are obtained through a min-hash-style algorithm on multiple random permutations, the similarity between any pair of fingerprints is the proportion of matched digits between $\boldsymbol{h}_i^{(r-1)}$ and $\boldsymbol{h}_j^{(r-1)}$. If we only consider one permutation sample, $\boldsymbol{h}_*^{(r-1)}$is a single digit and $S_*^{(r)}$ becomes a flat set. We can thus estimate the similarity between $S_a^{(r)}$ and $S_b^{(r)}$using the standard min-wise hashing algorithm. However, a good estimator requires a higher dimensionality (larger L) to approach the real Jaccard similarity as $L \to \infty$. For $L > 1$, we can consider each dimension of $\boldsymbol{h}_*^{(r-1)}$ separately, say $\{\boldsymbol{h}_{\left|S_*^{(r)}\right|}^{(r-1)}\}$, to approximately represent $S_*^{(r)}$as its L sampling-sets and the lth sampling-set is

$$S_{*,(l)}^{(r)} = \{\boldsymbol{h}_{1,(l)}^{(r-1)}, \ldots, \boldsymbol{h}_{\left|S_*^{(r)}\right|,(l)}^{(r-1)}\} \tag{6.7}$$

Now the fingerprint of $S_{*,(l)}^{(r)}$, say $\boldsymbol{h}_{\left|S_*^{(r)}\right|,(l)}^{(r-1)}$, can be obtained using min-wise hashing and the final fingerprint of $S_*^{(r)}$ is the concatenation of the L sampling-set fingerprints, that is, $\boldsymbol{h}_*^{(r)} = [\boldsymbol{h}_{*,(1)}^{(r-1)}, \ldots, \boldsymbol{h}_{*,(L)}^{(r-1)}]$. This direct concatenation is reasonable since min-hash fingerprints are unordered and digit-wise compared.

The detail of the RMH algorithm is summarized in Algorithm 2. $\mathbf{rmh}(S^{(r)}, \{\pi_k^{(r)}\})$ is a recursive procedure: If $S^{(r)}$ has reached the lowest level ($r = 1$), a min-hash procedure is applied to $S^{(r)}$, whose elements are tokens of words (Line 2). If $r > 1$, each element of $S^{(r)}$, say $S_*^{(r-1)}$, is input to a nested $\boldsymbol{rmh}$ procedure which returns the fingerprint of $S_*^{(r-1)}$ (Line 5). The L-dimensional fingerprints are reorganized into L separate sampling-

sets $\{S_{(l)}^{(r)}\}_{l=1}^{L}$ (Line 10), each of which is input to a ***minhash*** procedure to obtain its fingerprint (Line 11). Finally, the L sampling-set fingerprints are concatenated to form the fingerprint of $S^{(r)}$.

Algorithm 2 Recursive Min-wise Hashing (RMH)

Input: $S^{(r)}$: a nested set at the rth level; $\left\{\pi_k^{(r)}\right\}_{r=1,k=1}^{R,K}$: K hash functions (random permutations) at the rth level.

Output: $\boldsymbol{h}^{(r)}$: the fingerprint of $S^{(r)}$.

1: **if** $r = 1$ **then**

2: $\boldsymbol{h}^{(r)} \leftarrow minhash(S^{(r)}, \left\{\pi_k^{(r)}\right\}_{k=1}^{K})$;

3: **else**

4: **for** $S_*^{(r-1)} \in S^{(r)}$ **do**

5: $\boldsymbol{h}_*^{(r-1)} \leftarrow rmh\left(S_*^{(r-1)}, \left\{\pi_k^{(r)}\right\}_{r=1,k=1}^{R,K}\right)$

6: **end for**

7: $M \leftarrow \left|S^{(r)}\right|$

8: $L \leftarrow dim\left(\boldsymbol{h}_*^{(r-1)}\right)$

9: **for** $l = 1: L$ **do**

10: $S_{(l)}^{(r)} \leftarrow \left\{h_{m,(l)}^{(r-1)}\right\}_{m=1}^{M}$

11: $\boldsymbol{h}_{(l)}^{(r)} \leftarrow minhash\left(S_{(l)}^{(r)}, \left\{\pi_k^{(r)}\right\}_{k=1}^{K}\right)$

12: **end for**

13: $\boldsymbol{h}^{(r)} \leftarrow \left[\boldsymbol{h}_{(1)}^{(r)}; \ldots; \boldsymbol{h}_{(L)}^{(r)}\right]$;

14: **end if**

An example of the RMH algorithm is illustrated in Fig. 6.2: The min-wise hashing is performed on the three bottom-level sets (elements are terms) and we obtain three fingerprints, each of which has 4 min-hashes. We reorganize the three fingerprints into four separate sets by combining the lth min-hashes from the three fingerprints to form the l th sampling-set (shown in different colors in Fig 6.2). The min-wise hashing is performed on the second-level sets, and the obtained fingerprints are concatenated to form the final fingerprint of the input text.

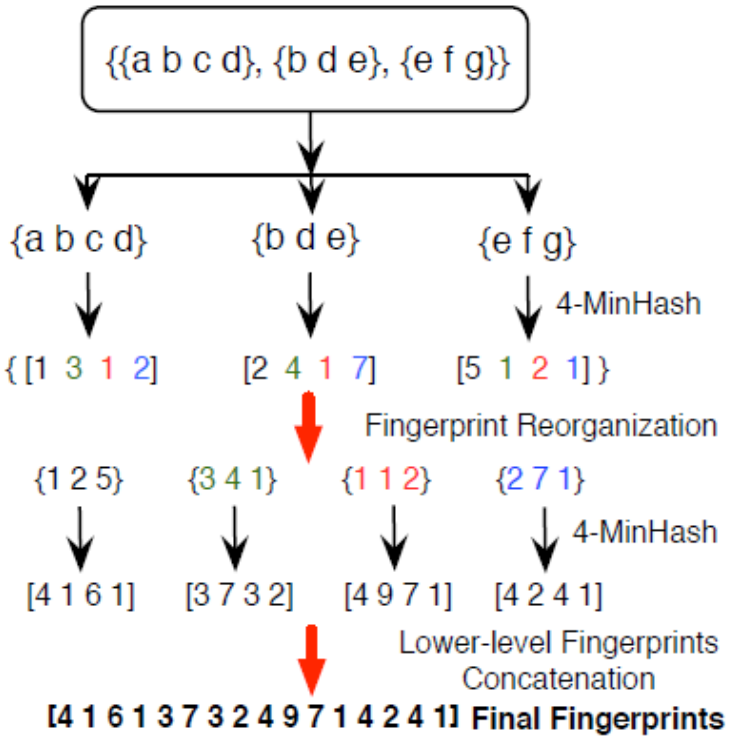


Figure 6.2: An illustration of the proposed Recursive Min-wise Hashing (RMH) algorithm on a nested set

6.3.3 Time Complexity Analysis

We finally analyse the computational complexity of the RMH algorithm. Let R be the number of levels in a nested set (i.e., the depth of recursive procedures), K be the number of min-hash functions at each level, and $|S^{(R)}|$ be the number of set-elements in the top-set. According to Algorithm 2, the time complexity of the top-level recursion is

$O(|S^{(R)}|K^R)$, in which $O(K^{R-1})$ is for the number of reorganized sets and $O|S^{(R)}|K$ for K min-wise hashing procedures on $S^{(R)}$. The time complexity of the bottom-level of recursion is $O(\prod_{r=1}^{R}|S^{(R)}|K)$, in which $O(\prod_{r=1}^{R}|S^{(R)}|)$ is for the number of the bottom-level sets and $O|S_*^{(1)}|K$ for K min-wise hashing procedures on $S_*^{(1)}$. Now it is easy to see that the time complexity for the tth level is

$$O(\prod_{r=R-t+1}^{R}|S_*^{(r)}|K^{R-t+1}) \tag{6.8}$$

It is worth noting that the orders of the two parts in Eq. (6.8), $\prod_{r=R-t+1}^{R}|S_*^{(r)}|$ and K^{R-t+1}, sum to a constant $R+1$.

THEOREM 6.1. *Suppose the sizes of all the nested sets in $S^{(R)}$ are smaller than a constant P, that is $|S_*^{(r)}| \leq P$, for any nested set at the rth level, Eq. (6.8) is upper bounded by $O(P^tK^{R-t+1})$. Thus, the time complexity of Algorithm 2 over all the recursions is at most*

$$O(\sum_{t=1}^{R}P^t\,K^{R-t+1}) \tag{6.9}$$

In practice, we normally consider two- or three-level nested sets, that is, $R = \{2,3\}$, P and K are in $O(10)$, therefore the overall practical time complexity is $O(10^{2\sim3})$ for computing a context-preserving fingerprint for a text.

6.4 EXPERIMENT

In this section, we empirically test the proposed context-preserving hashing method on three real-world text data sets. In particular, we investigate the effectiveness and efficiency of our method and the compared methods for text classification. We aim to show that the proposed method is able to 1) *significantly improve the classification accuracy* and 2) *remain the same (or less) computational cost*, compared to the baselines. The classification performance of different methods on the derived fingerprints or feature vectors is evaluated using LIBSVM [109]. All the results in our experiments are the average performance over five random data splits. The four compared methods are

implemented in Matlab and all the experiments are conducted on a node of Linux Cluster with 2.90GHz Intel Xeon CPU.

6.4.1 DATA SETS

We study three text classification tasks based on real-world text data sets:

- ***Paper Abstract Classification*** **(DBLP)**: There are 1,632,442 papers in the downloaded raw data package. We consider a binary classification task: Artificial Intelligence (AI) vs. Computer Vision (CV). We define the papers in {IJCAI, AAAI, NIPS, UAI, COLT, ACL, KR, ICML, ECML, IJCNN} as the AI category and {CVPR, ICCV, ECCV, ICIP, ICPR, ACM Multimedia, ICME} as the CV category. We extract the abstracts from the resulting 15,195 papers as the data set for our text classification task. *Review Polarity Classification (IMDB)*: This is a data set for binary sentiment classification with 25,000 highly polar movie reviews for training and 25,000 for testing. We samples 20,000 reviews with balanced positive and negative samples for our text classification task.

- ***Review Polarity Classification*** **(IMDB)**: This is a data set for binary sentiment classification with 25,000 highly polar movie reviews for training and 25,000 for testing. We samples 20,000 reviews with balanced positive and negative samples for our text classification task.

- ***Review Category Classification*** **(Amazon)**: This data set contains a large number of reviews in different product categories. We randomly sample 10,000 examples from Books and 10,000 from Music to form a binary review-category classification task.

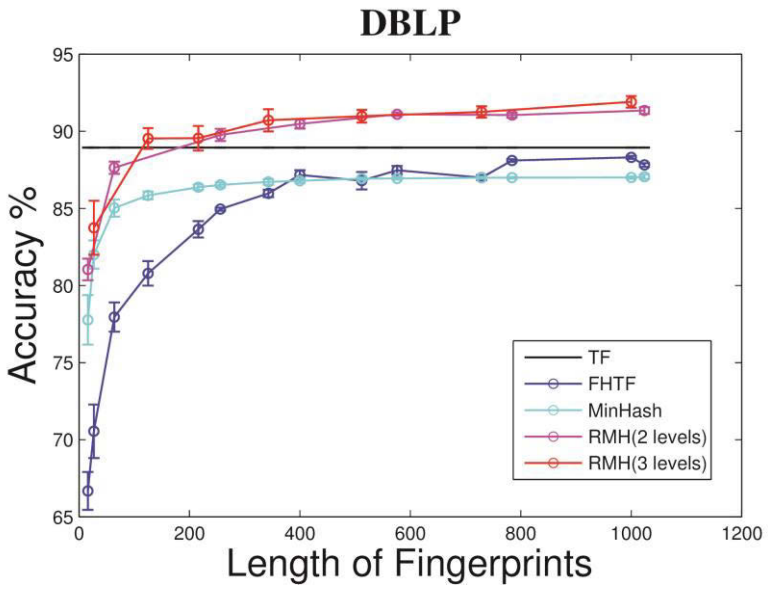
6.4.2 COMPARED METHODS

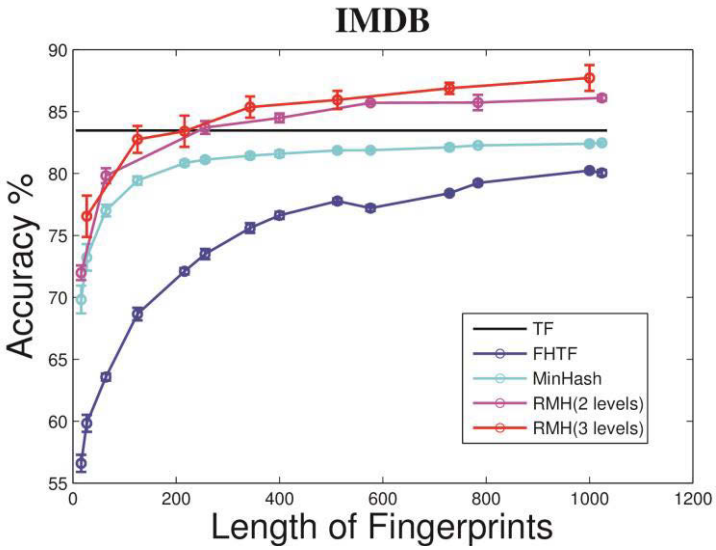
The three baseline methods are introduced in Section 2: (1) *Term Frequency* (TF): the most basic method; (2) *Feature Hashing on Term Frequency* (FHTF): applying feature hashing [116] to TF; (3) *Min-wise Hashing* (MinHash) [10]: the proposed RMH algorithm can be viewed as the generalization of MinHash. (4) *Recursive Min-wise*

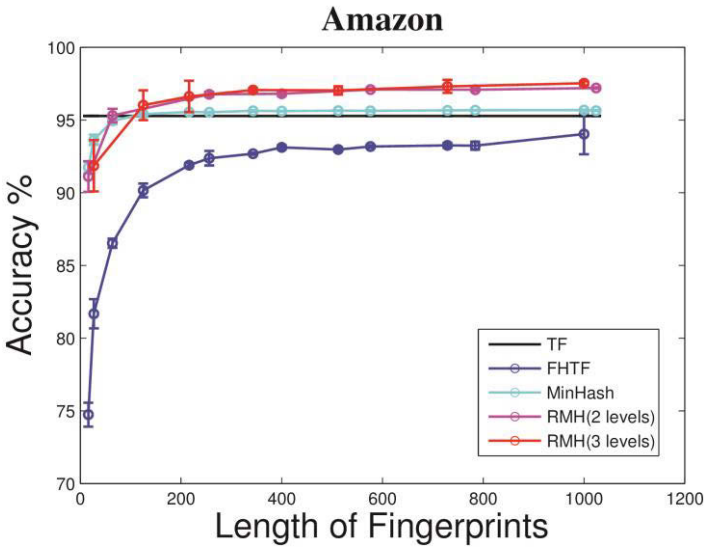
Hashing (RMH): the proposed method. We will investigate different configurations of RMH, such as RMH (2 levels) and RMH (3 levels), in the experiments.

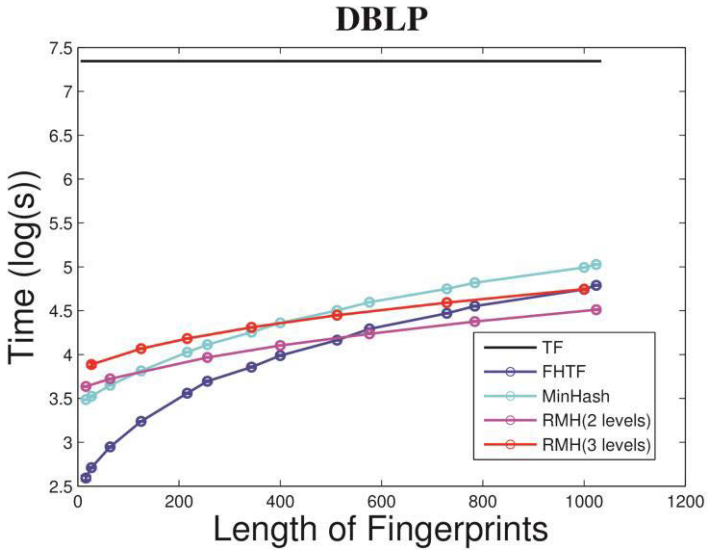
6.4.3 Performance Comparison

We first investigate the classification performance and CPU time of the compared methods (FHTF, MinHash, and two RMH configurations) in terms of the length of output fingerprints (L). TF is based on the real size of its vocabulary so it has only one result. We investigate $L \in \mathcal{B}_2 = \{4^2, 8^2, 16^2, 20^2, 24^2, 28^2, 32^2\}$ (i.e., $K \in \{4, 8, 16, 20, 24, 28, 32\}$) for RMH (2 levels), which considers words and sentences; and we investigate $L \in \mathcal{B}_3 = \{3^3, 5^3, 6^3, 7^3, 8^3, 9^3, 10^3\}$ (i.e., $K \in \{3, 5, 6, 7, 8, 9, 10\}$) for RMH (3 levels), which considers words, sub-sentences (commas separated), and sentences. The length range of the fingerprints for FHTF and the number of the hashed dimensions for MinHash is varied in $\mathcal{B}_2 \cup \mathcal{B}_3$.




IMDB
Accuracy %
Length of Fingerprints
TF
FHTF
MinHash
RMH(2 levels)
RMH(3 levels)


Amazon
Accuracy %
Length of Fingerprints
TF
FHTF
MinHash
RMH(2 levels)
RMH(3 levels)


DBLP
Time (log(s))
Length of Fingerprints
TF
FHTF
MinHash
RMH(2 levels)
RMH(3 levels)

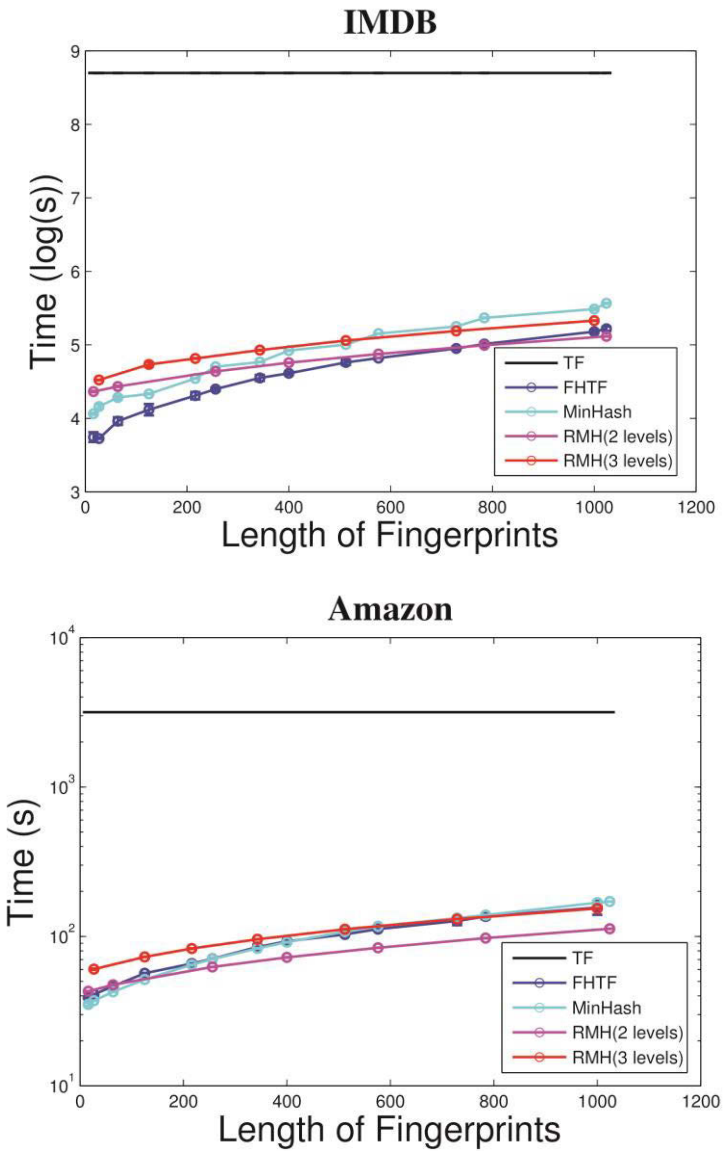


Figure 6.3: Classification accuracy and CPU time of the compared methods w.r.t. the length of output fingerprints

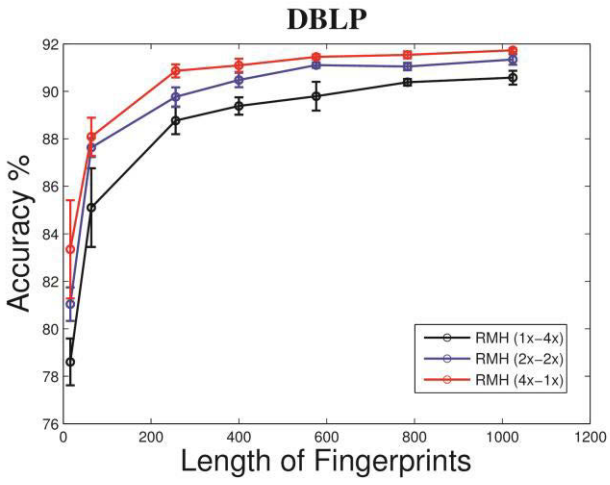
In Fig. 6.3, we can find that the two RMH methods achieve the best overall performance on all the data sets. As expected, RHM (3 levels) performs slightly better than RMH (2 levels) due to a more expressive representation for semantic hierarchy. The performance gain of RMH over MinHash becomes more significant as the length of the fingerprints increases; and all the three min-hash based methods can significantly outperform FHTF at all lengths. It is worth noting that, since $L > 100$, RMH starts to outperform TF which is based on the full vocabularies; and the performance gain keeps increasing afterwards. This experiment indicates that context-preserving hashing is able

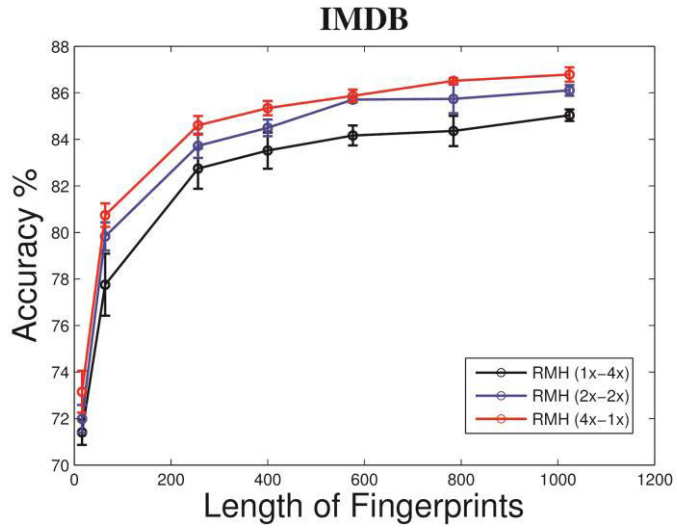
to acquire extra useful information from texts (because RMH can significantly outperform TF while MinHash can only approaches the performance of TF as its fingerprint length approaches the size of the vocabulary).

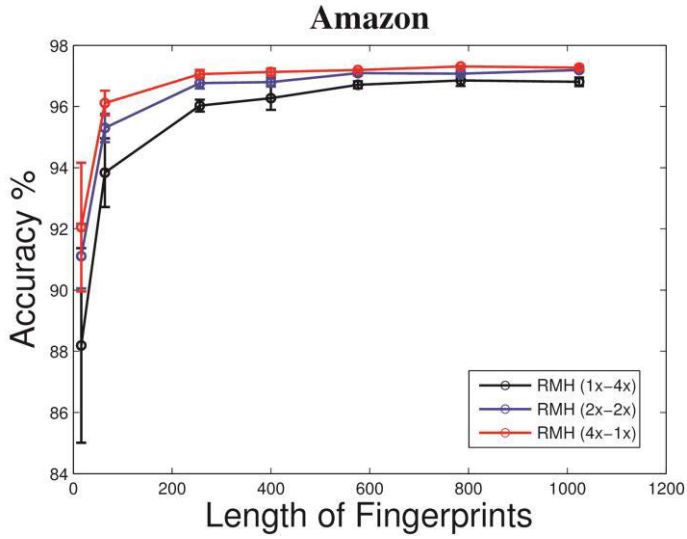
Fig. 6.3 also plots the CPU time comparison. TF has a computational cost two orders of magnitude larger than the other methods since it is based on the full vocabularies. FHTF is slightly faster than the three min-hash based methods because of a different hashing scheme. The two RMH methods consume slightly less CPU time than MinHash does; RMH (2 levels) is slightly faster than RMH (3 levels). This experiment implies that context-preserving hashing can achieve significantly improved accuracy without additional computational cost.

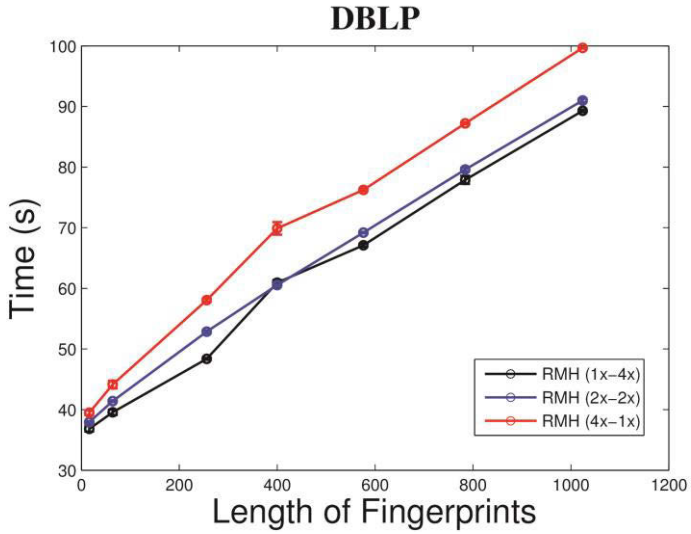
6.4.4 Investigation of Min-hash Size

In the previous experiment, we assume that the number of min-hashes (hash functions) is same for all the levels of a nested set. Indeed, we can vary the number of min-hashes for different levels of a nested set. In this experiment, we investigate the performance of RMH (2 levels) by varying min-hash sizes. In particular, we fix the length of output fingerprints $L = \{16, 64, 256, 400, 576, 784, 1024\}$ and consider three min-hash size configurations: "1x-4x" (the number of min-hashes at word-level is a quarter of that at sentence-level), "2x-2x" (equal size), and 4x-1x (the number of min-hashes at word-level is four times of that at sentence-level). For example, for $L = 400$, "1x-4x" means 10 min-hashes at the word-level and 40 min-hashes at the sentence-level.




IMDB
Accuracy %
Length of Fingerprints
RMH (1x–4x)
RMH (2x–2x)
RMH (4x–1x)


Amazon
Accuracy %
Length of Fingerprints
RMH (1x–4x)
RMH (2x–2x)
RMH (4x–1x)


DBLP
Time (s)
Length of Fingerprints
RMH (1x–4x)
RMH (2x–2x)
RMH (4x–1x)

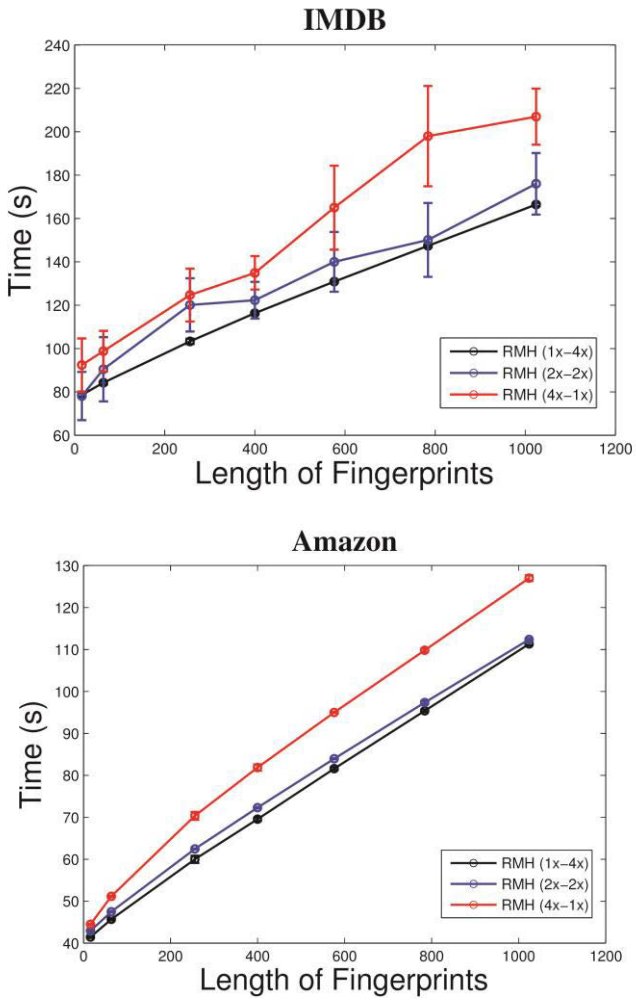


Figure 6.4: Classification accuracy and CPU time of the RMH algorithm with different min-hash sizes at different levels of the nested sets

In Fig. 6.4, we can find that RMH (4x-1x) slightly outperforms RMH (2x-2x), which further slightly outperforms RMH (1x-4x). This result is reasonable because the word-level sets are larger than sentence-level sets in size such that more min-hashes are required to produce a better estimation.

The CPU time comparison in Fig. 6.4 shows that RMH (4x-1x) consumes more time than the other two configurations. This is because it has more hash functions performed on word-level sets which have bigger sizes than sentence-level sets. RMH (2x-2x) is slightly slower than RMH (1x-4x) for the same reason.

6.5 SUMMARY

In this chapter, we focus on the text structured data and propose a context-preserving fingerprinting method for fast estimating similarity between texts to relieve the loss of context information. We first represent a text as a nested set based on the notion of multi-level exchangeability at words, sentences, paragraphs, etc., and then propose a Recursive Min-wise Hashing (RMH) algorithm to fingerprint the obtained nested set for fast comparison. The empirical studies on three real-world text classification tasks show that our context-preserving hashing method, as a generalization and improvement of the standard min-hash scheme, is able to not only significantly outperform min-wise hashing and feature hashing in accuracy but also maintain the same (even less) computational cost.

CHAPTER 7
CONCLUSIONS AND FURTHER STUDY

This chapter concludes the whole thesis and provides some further research directions of the topic.

7.1 CONCLUSIONS

In this thesis, we focus on exploring new hashing scheme in different data classification scenarios. We aim to explore new hashing methods from different views and utilize them to boost classification performance in different data tasks. We first do a literature review that surveys existing works on hashing from data classification perspective. Then, we focus on exploring the new hashing methods on the graph structured data classification and text structured data classification:

Firstly, we propose a DIscriminative Clique Hashing (DICH) for fast graph stream classification. The main idea is to employ a fast algorithm to decompose a compressed graph into a number of cliques to sequentially extract clique-patterns over the graph stream as features. Two random hashing schemes are employed to speed up the discriminative clique-pattern mining process and address the unlimitedly clique-pattern expanding problem. The hashed cliques are used to update an "in-memory" fixed-size pattern-class table, which is finally used to construct a rule-based classifier. We test DICH on two real-world graph stream data sets. Because DICH directly extracts cliques (connected subgraphs) from the graph stream as features for classifier training, rather than mining unconnected co-occurrence edge sets as that in the compared state-of-the-art

method, DICH can significantly outperform the baseline method in both classification accuracy and learning efficiency.

Secondly, we further improve the DICH and propose an adaptive hashing for real-time classification. In this method, we use two hashing schemes, incremental stochastic learning strategy and chunk level weighting mechanism to address the "real-time", "one-pass" and "concept drifting" challenges. In particular, we propose an approximate method for fast graph feature extraction by detecting cliques from the compressed graphs via hashing, which can significantly improve the efficiency of feature extraction to satisfy the "real-time" requirement. We also propose a graph feature reduction method by mapping unlimitedly expanding clique patterns onto corresponding fixed-size compatible feature spaces via differential hashing, which can avoid a pre-scan of graphs to address the "one-pass" and "concept drifting" challenges. Thanks to the clique hashing approach, the stream of graphs can be converted into feature vectors without additional parsing so that we can directly adopt a stochastic learning strategy to train a graph classifier online. Then, a chunk level weighting mechanism is adopted to address "concept drifting" challenge. The experimental results on two real-world and one synthetic graph streams demonstrate that the proposed method can outperform the state.

Finally, we focus on text structured data and propose a context-preserving fingerprinting method for fast estimating similarity to relieve the loss of context information. We first represent a text as a nested set based on the notion of multi-level exchangeability at words, sentences, paragraphs, etc., and propose a Recursive Min-wise Hashing (RMH) algorithm to fingerprint the obtained nested set for fast comparison. The empirical studies on three real-world text classification tasks show that our context-preserving hashing method, as a generalization and improvement of the standard min-hash scheme, is able to not only significantly outperform min-wise hashing and feature hashing in accuracy but also maintain the same (even less) computational cost.

7.2 FURTHER STUDY

In this thesis, we have studied hashing scheme from the view of large-scale structured data classification. This study is commonly applied to large-scale data sets. However, applying hashing algorithms to extremely large scale and streaming problems still poses challenges: (1) how to find an more effective representation for a high-dimensional large-scale data, so as to fit in memory, (2) how to better improve the efficiency of hashing method, including the classification accuracy and time and (3) how to flexibly apply the new hashing scheme into different structured data. In our future work, we focus on designing hashing algorithms and approaches that are faster, data efficient and less demanding in computational resources to achieve scalable algorithms for extremely large scale and streaming problems.

In addition, two extensions of the proposed context-preserving fingerprinting method can be considered in the future work: First, we can directly apply the b-bit min-hash algorithm [12] to our method to improve its efficiency. Second, our method can be naturally applied to visual features to capture contexts in images.

REFERENCES

1 Indyk, P., and Motwani, R.: 'Approximate nearest neighbors: towards removing the curse of dimensionality', in Editor (Ed.)^(Eds.): 'Book Approximate nearest neighbors: towards removing the curse of dimensionality' (ACM, 1998, edn.), pp. 604-613

2 Weber, R., Schek, H.-J., and Blott, S.: 'A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces', in Editor (Ed.)^(Eds.): 'Book A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces' (1998, edn.), pp. 194-205

3 Gionis, A., Indyk, P., and Motwani, R.: 'Similarity search in high dimensions via hashing', in Editor (Ed.)^(Eds.): 'Book Similarity search in high dimensions via hashing' (1999, edn.), pp. 518-529

4 Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V.S.: 'Locality-sensitive hashing scheme based on p-stable distributions', in Editor (Ed.)^(Eds.): 'Book Locality-sensitive hashing scheme based on p-stable distributions' (ACM, 2004, edn.), pp. 253-262

5 Joly, A., Frélicot, C., and Buisson, O.: 'Feature statistical retrieval applied to content based copy identification', in Editor (Ed.)^(Eds.): 'Book Feature statistical retrieval applied to content based copy identification' (IEEE, 2004, edn.), pp. 681-684

6 Shakhnarovich, G.: 'Learning Task-Specific Similarity', MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2005

7 Shakhnarovich, G., Indyk, P., and Darrell, T.: 'Nearest-neighbor methods in learning and vision: theory and practice' (2006. 2006)

8 Poullot, S., Buisson, O., and Crucianu, M.: 'Z-grid-based probabilistic retrieval for scaling up content-based copy detection', in Editor (Ed.)^(Eds.): 'Book Z-grid-based probabilistic retrieval for scaling up content-based copy detection' (ACM, 2007, edn.), pp. 348-355

9 Broder, A.Z.: 'On the resemblance and containment of documents', in Editor (Ed.)^(Eds.): 'Book On the resemblance and containment of documents' (IEEE, 1997, edn.), pp. 21-29

10 Broder, A.Z., Charikar, M., Frieze, A.M., and Mitzenmacher, M.: 'Min-wise independent permutations', Journal of Computer and System Sciences, 2000, 60, (3), pp. 630-659

11 Chum, O., Philbin, J., and Zisserman, A.: 'Near Duplicate Image Detection: min-Hash and tf-idf Weighting', in Editor (Ed.)^(Eds.): 'Book Near Duplicate Image Detection: min-Hash and tf-idf Weighting' (2008, edn.), pp. 812-815

12 Raginsky, M., and Lazebnik, S.: 'Locality-Sensitive Binary Codes from Shift-Invariant Kernels', 2009

13 Li, P., Konig, A., and Gui, W.: 'b-Bit minwise hashing for estimating three-way similarities', in Editor (Ed.)^(Eds.): 'Book b-Bit minwise hashing for estimating three-way similarities' (2010, edn.), pp. 1387-1395

14 Li, P., and König, C.: 'b-Bit minwise hashing', in Editor (Ed.)^(Eds.): 'Book b-Bit minwise hashing' (ACM, 2010, edn.), pp. 671-680

15 Li, P., and König, A.C.: 'Theory and applications of b-bit minwise hashing', Communications of the ACM, 2011, 54, (8), pp. 101-109

16 Li, P., Shrivastava, A., Moore, J.L., and König, A.C.: 'Hashing algorithms for large-scale learning', in Editor (Ed.)^(Eds.): 'Book Hashing algorithms for large-scale learning' (2011, edn.), pp. 2672-2680

17 Li, B., Zhu, X., Chi, L., and Zhang, C.: 'Nested Subtree Hash Kernels for Large-Scale Graph Classification over Streams', in Editor (Ed.)^(Eds.): 'Book Nested Subtree Hash Kernels for Large-Scale Graph Classification over Streams' (IEEE Computer Society, 2012, edn.), pp. 399-408

18 Chi, L., Li, B., and Zhu, X.: 'Fast graph stream classification using discriminative clique hashing': 'Advances in Knowledge Discovery and Data Mining' (Springer, 2013), pp. 225-236

19 Chi, L., Li, B., and Zhu, X.: 'Context-Preserving Hashing for Fast Text Classification', 2014

20 Moran, S., Lavrenko, V., and Osborne, M.: 'Neighbourhood preserving quantisation for lsh', in Editor (Ed.)^(Eds.): 'Book Neighbourhood preserving quantisation for lsh' (ACM, 2013, edn.), pp. 1009-1012

21 Moran, S., Lavrenko, V., and Osborne, M.: 'Variable Bit Quantisation for LSH', in Editor (Ed.)^(Eds.): 'Book Variable Bit Quantisation for LSH' (2013, edn.), pp. 753-758

22 Zhang, L., Zhang, Y., Zhang, D., and Tian, Q.: 'Distribution-Aware Locality Sensitive Hashing': 'Advances in Multimedia Modeling' (Springer, 2013), pp. 395-406

23 Charikar, M.S.: 'Similarity estimation techniques from rounding algorithms', in Editor (Ed.)^(Eds.): 'Book Similarity estimation techniques from rounding algorithms' (ACM, 2002, edn.), pp. 380-388

24 Kulis, B., and Grauman, K.: 'Kernelized locality-sensitive hashing for scalable image search', in Editor (Ed.)^(Eds.): 'Book Kernelized locality-sensitive hashing for scalable image search' (IEEE, 2009, edn.), pp. 2130-2137

25 Ji, J., Li, J., Yan, S., Zhang, B., and Tian, Q.: 'Super-bit locality-sensitive hashing', in Editor (Ed.)^(Eds.): 'Book Super-bit locality-sensitive hashing' (2012, edn.), pp. 108-116

26 D. E. Knuth, Sorting and Searching, 2nd ed., Art of Computer Programming 3, Addison-Wesley, Reading, MA, 1998.

27 J. L. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143-154.

28 M. N. Wegman and J. L. Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265-279.

29 A. Siegel, *On universal classes of extremely random constant-time hash functions*, SIAM J. Comput., 33 (2004), pp. 505 – 543.

30 G.R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(5):530–549, 2003.

31 J. Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. Israel Journal of Mathematics, 52:46–52, 1985.

32 C. Faloutsos and K.I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In ACM SIGMOD International Conference on Management of Data, pages 163–174, 1995.

33 X. Wang, J.T.L. Wang, K.I. Lin, D. Shasha, B.A. Shapiro, and K. Zhang. An index structure for data mining and clustering. Knowledge and Information Systems, 2(2):161–184, 2000.

34 R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297–336, 1999.

35 Jagadish, H.V. Analysis of the Hilbert curve for representing twodimensional space, Inf. Process. Lett., 1997, 62, (1), pp. 17-22

36 Weiss, Y., Torralba, A., and Fergus, R.: '17 Spectral hashing', in Editor (Ed.)^(Eds.): 'Book Spectral hashing' (2009, edn.), pp. 1753-1760

37 Liu, W., Wang, J., Kumar, S., and Chang, S.-F.: 'Hashing with graphs', in Editor (Ed.)^(Eds.): 'Book Hashing with graphs' (2011, edn.), pp. 1-8

38 Salakhutdinov, R., and Hinton, G.: 'Semantic hashing', International Journal of Approximate Reasoning, 2009, 50, (7), pp. 969-978

39 Brandt, J.: 'Transform coding for fast approximate nearest neighbor search in high dimensions', in Editor (Ed.)^(Eds.): 'Book Transform coding for fast approximate nearest neighbor search in high dimensions' (IEEE, 2010, edn.), pp. 1815-1822

40 Zhang, D., Wang, J., Cai, D., and Lu, J.: 'Laplacian co-hashing of terms and documents': 'Advances in Information Retrieval' (Springer, 2010), pp. 577-580

41 Zhang, D., Wang, J., Cai, D., and Lu, J.: 'Self-taught hashing for fast similarity search', in Editor (Ed.)^(Eds.): 'Book Self-taught hashing for fast similarity search' (ACM, 2010, edn.), pp. 18-25

42 Gong, Y., and Lazebnik, S.: 'Iterative quantization: A procrustean approach to learning binary codes', in Editor (Ed.)^(Eds.): 'Book 19 Iterative quantization: A procrustean approach to learning binary codes' (IEEE, 2011, edn.), pp. 817-824

43 Joly, A., and Buisson, O.: 'Random maximum margin hashing', in Editor (Ed.)^(Eds.): 'Book Random maximum margin hashing' (IEEE, 2011, edn.), pp. 873-880

44 Xu, H., Wang, J., Li, Z., Zeng, G., Li, S., and Yu, N.: 'Complementary hashing for approximate nearest neighbor search', in Editor (Ed.)^(Eds.): 'Book Complementary hashing for approximate nearest neighbor search' (IEEE, 2011, edn.), pp. 1631-1638

45 Wang, X.-J., Zhang, L., Jing, F., and Ma, W.-Y.: 'Annosearch: Image auto-annotation by search', in Editor (Ed.)^(Eds.): 'Book Annosearch: Image auto-annotation by search' (IEEE, 2006, edn.), pp. 1483-1490

46 He, J., Liu, W., and Chang, S.-F.: 'Scalable similarity search with optimized kernel hashing', in Editor (Ed.)^(Eds.): 'Book Scalable similarity search with optimized kernel hashing' (ACM, 2010, edn.), pp. 1129-1138

47 Liu, X., He, J., Liu, D., and Lang, B.: 'Compact kernel hashing with multiple features', in Editor (Ed.)^(Eds.): 'Book Compact kernel hashing with multiple features' (ACM, 2012, edn.), pp. 881-884

48 Wang, J., Kumar, S., and Chang, S.-F.: 'Sequential projection learning for hashing with compact codes', in Editor (Ed.)^(Eds.): 'Book Sequential projection learning for hashing with compact codes' (2010, edn.), pp. 1127-1134

49 Kang, Y., Kim, S., and Choi, S.: 'Deep Learning to Hash with Multiple Representations', in Editor (Ed.)^(Eds.): 'Book Deep Learning to Hash with Multiple Representations' (2012, edn.), pp. 930-935

50 Xu, Z., Kersting, K., and Bauckhage, C.: 'Efficient Learning for Hashing Proportional Data', in Editor (Ed.)^(Eds.): 'Book Efficient Learning for Hashing Proportional Data' (2012, edn.), pp. 735-744

51 Zhen, Y., and Yeung, D.-Y.: 'A probabilistic model for multimodal hash function learning', in Editor (Ed.)^(Eds.): 'Book A probabilistic model for multimodal hash function learning' (ACM, 2012, edn.), pp. 940-948

52 He, K., Wen, F., and Sun, J.: ‘K-means hashing: an affinity-preserving quantization method for learning binary compact codes’, in Editor (Ed.)^(Eds.): ‘Book K-means hashing: an affinity-preserving quantization method for learning binary compact codes’ (IEEE, 2013, edn.), pp. 2938-2945

53 Jegou, H., Douze, M., and Schmid, C.: ‘Product quantization for nearest neighbor search’, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2011, 33, (1), pp. 117-128

54 Gong, Y., Kumar, S., Verma, V., and Lazebnik, S.: ‘Angular quantization-based binary codes for fast similarity search’, in Editor (Ed.)^(Eds.): ‘Book Angular quantization-based binary codes for fast similarity search’ (2012, edn.), pp. 1196-1204

55 Heo, J.-P., Lee, Y., He, J., Chang, S.-F., and Yoon, S.-E.: ‘Spherical hashing’, in Editor (Ed.)^(Eds.): ‘Book 80 Spherical hashing’ (IEEE, 2012, edn.), pp. 2957-2964

56 Kong, W., and Li, W.-J.: ‘Isotropic hashing’, in Editor (Ed.)^(Eds.): ‘Book Isotropic hashing’ (2012, edn.), pp. 1646-1654

57 Kong, W., Li, W.-J., and Guo, M.: ‘Manhattan hashing for large-scale image retrieval’, in Editor (Ed.)^(Eds.): ‘Book Manhattan hashing for large-scale image retrieval’ (ACM, 2012, edn.), pp. 45-54

58 Lu, Y., Prabhakar, B., and Bonomi, F.: ‘Perfect hashing for network applications’, in Editor (Ed.)^(Eds.): ‘Book Perfect hashing for network applications’ (IEEE, 2006, edn.), pp. 2774-2778

59 Kontak, V., Srbljic, S., and Skvorc, D.: ‘Hashing scheme for space-efficient detection and localization of changes in large data sets’, in Editor (Ed.)^(Eds.): ‘Book Hashing scheme for space-efficient detection and localization of changes in large data sets’ (IEEE, 2012, edn.), pp. 1496-1501

60 Lin, Y., Jin, R., Cai, D., Yan, S., and Li, X.: 'Compressed hashing', in Editor (Ed.)^(Eds.): 'Book Compressed hashing' (IEEE, 2013, edn.), pp. 446-451

61 Rastegari, M., Choi, J., Fakhraei, S., Hal, D., and Davis, L.: 'Predictable Dual-View Hashing', in Editor (Ed.)^(Eds.): 'Book Predictable Dual-View Hashing' (2013, edn.), pp. 1328-1336

62 Shen, F., Shen, C., Shi, Q., Van Den Hengel, A., and Tang, Z.: 'Inductive hashing on manifolds', in Editor (Ed.)^(Eds.): 'Book Inductive hashing on manifolds' (IEEE, 2013, edn.), pp. 1562-1569

63 Irie, G., Li, Z., Wu, X.-M., and Chang, S.-F.: 'Locally Linear Hashing for Extracting Non-Linear Manifolds', 2014

64 Zhang, L., Zhang, Y., Gu, X., Tang, J., and Tian, Q.: 'Scalable Similarity Search with Topology Preserving Hashing', 2014

65 Wang, J., Kumar, S., and Chang, S.-F.: 'Semi-supervised hashing for scalable image retrieval', in Editor (Ed.)^(Eds.): 'Book Semi-supervised hashing for scalable image retrieval' (IEEE, 2010, edn.), pp. 3424-3431

66 Wang, J., Kumar, S., and Chang, S.-F.: 'Semi-supervised hashing for large-scale search', Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2012, 34, (12), pp. 2393-2406

67 Mu, Y., Shen, J., and Yan, S.: 'Weakly-supervised hashing in kernel space', in Editor (Ed.)^(Eds.): 'Book Weakly-supervised hashing in kernel space' (IEEE, 2010, edn.), pp. 3344-3351

68 Kim, S., and Choi, S.: 'Semi-supervised discriminant hashing', in Editor (Ed.)^(Eds.): 'Book Semi-supervised discriminant hashing' (IEEE, 2011, edn.), pp. 1122-1127

69 Wu, C., Zhu, J., Cai, D., Chen, C., and Bu, J.: 'Semi-supervised nonlinear hashing using bootstrap sequential projection learning', Knowledge and Data Engineering, IEEE Transactions on, 2013, 25, (6), pp. 1380-1393

70 Shakhnarovich, G., Viola, P., and Darrell, T.: 'Fast pose estimation with parameter-sensitive hashing', in Editor (Ed.)^(Eds.): 'Book Fast pose estimation with parameter-sensitive hashing' (IEEE, 2003, edn.), pp. 750-757

71 Salakhutdinov, R., and Hinton, G.E.: 'Learning a nonlinear embedding by preserving class neighbourhood structure', in Editor (Ed.)^(Eds.): 'Book Learning a nonlinear embedding by preserving class neighbourhood structure' (2007, edn.), pp. 412-419

72 Torralba, A., Fergus, R., and Weiss, Y.: 'Small codes and large image databases for recognition', in Editor (Ed.)^(Eds.): 'Book Small codes and large image databases for recognition' (IEEE, 2008, edn.), pp. 1-8

73 Kulis, B., and Darrell, T.: 'Learning to hash with binary reconstructive embeddings', in Editor (Ed.)^(Eds.): 'Book Learning to hash with binary reconstructive embeddings' (2009, edn.), pp. 1042-1050

74 Norouzi, M., and Blei, D.M.: 'Minimal loss hashing for compact binary codes', in Editor (Ed.)^(Eds.): 'Book Minimal loss hashing for compact binary codes' (2011, edn.), pp. 353-360

75 Liu, W., Wang, J., Ji, R., Jiang, Y.-G., and Chang, S.-F.: 'Supervised hashing with kernels', in Editor (Ed.)^(Eds.): 'Book Supervised hashing with kernels' (IEEE, 2012, edn.), pp. 2074-2081

76 Strecha, C., Bronstein, A.M., Bronstein, M.M., and Fua, P.: 'LDAHash: Improved matching with smaller descriptors', Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2012, 34, (1), pp. 66-78

77 Breitinger, F.: 'Similarity Preserving Hashing', in Editor (Ed.)^(Eds.): 'Book Similarity Preserving Hashing' (2013, edn.), pp. 17

78 Lin, G., Shen, C., Suter, D., and Hengel, A.v.d.: 'A general two-step approach to learning-based hashing', in Editor (Ed.)^(Eds.): 'Book A general two-step approach to learning-based hashing' (IEEE, 2013, edn.), pp. 2552-2559

79 Masci, J., Bronstein, M., Bronstein, A., and Schmidhuber, J.: 'Multimodal similarity-preserving hashing', 2013

80 Zhang, D., and Li, W.-J.: 'Large-Scale Supervised Multimodal Hashing with Semantic Correlation Maximization', in Editor (Ed.)^(Eds.): 'Book Large-Scale Supervised Multimodal Hashing with Semantic Correlation Maximization' (2014, edn.), pp.

81 Zhang, P., Zhang, W., Li, W.-J., and Guo, M.: 'Supervised Hashing with Latent Factor Models', in Editor (Ed.)^(Eds.): 'Book Supervised Hashing with Latent Factor Models' (SIGIR, 2014, edn.), pp.

82 Lin, G., Shen, C., Shi, Q., Hengel, A.v.d., and Suter, D.: 'Fast Supervised Hashing with Decision Trees for High-Dimensional Data', arXiv preprint arXiv:1404.1561, 2014

83 Zhu, X., Huang, Z., Cheng, H., Cui, J., and Shen, H.T.: 'Sparse hashing for fast multimedia search', ACM Transactions on Information Systems (TOIS), 2013, 31, (2), pp. 9

84 Zhu, X., Huang, Z., Shen, H.T., and Zhao, X.: 'Linear cross-modal hashing for efficient multimedia search', in Editor (Ed.)^(Eds.): 'Book Linear cross-modal hashing for efficient multimedia search' (ACM, 2013, edn.), pp. 143-152

85 Song, J.: 'Effective Hashing for Searching Large-scale Multimedia Databases', 2014

86 Xu, Y., Ma, L., Liu, Z., and Chao, H.J.: 'A multi-dimensional progressive perfect hashing for high-speed string matching', in Editor (Ed.)^(Eds.): 'Book A multi-dimensional progressive perfect hashing for high-speed string matching' (IEEE, 2011, edn.), pp. 167-177

87 Aggarwal, C.C., Wang, H.: Managing and Mining Graph Data. Springer, New York (2010)

88 Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: ICML. (2003) 321–328

89 Borgwardt, K.M., Kriegel, H.P.: Shortest-path kernels on graphs. In: ICDM. (2005) 74–81

90 Mah´e, P., Vert, J.P.: Graph kernels based on tree patterns for molecules. Machine Learning 75(1) (2009) 3–35

91 Shervashidze, N., Borgwardt, K.: Fast subtree kernels on graphs. In: NIPS. (2009) 1660–1668

92 Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the data-stream model. SIAM Journal on Computing 38(5) (2008) 1709–1727

93 Aggarwal, C.C., Zhao, Y., Yu, P.S.: On clustering graph streams. In: SDM. (2010) 478–489

94 Aggarwal, C.C., Li, Y., Yu, P.S., Jin, R.: On dense pattern mining in graph streams. In: PVLDB. (2010) 975–984

95 Aggarwal, C.C.: On classification of graph streams. In: SDM. (2011) 652–663

96 Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: KDD. (2003) 226–235

97 Vishwanathan, S., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research 11 (2010) 1201–1242

98 Hido, S., Kashima, H.: A linear-time graph kernel. In: ICDM. (2009) 179–188

99 Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: ICDM. (2002) 721–724

100 Soufiani, H.A., Airoldi, E.: Graphlet decomposition of a weighted network. Journal of Machine Learning Research – Proceedings Track 22 (2012) 54–63

101 Bron, C., Kerbosch, J.: Algorithm 457: Finding all cliques of an undirected graph. Communications of the ACM 16(9) (1973) 575–577

102 L. Chen and C. Wang, "Continuous subgraph pattern search over certain and uncertain graph streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 8, pp. 1093–1109, 2010.

103 S. Pan, X. Zhu, C. Zhang, and P. S. Yu, "Graph stream classification using labeled and unlabeled graphs," in *ICDE*, 2013, pp. 398–409.

104 U. Feige, S. Goldwasser, L. Lov´asz, S. Safra, and M. Szegedy, "Approximating clique is almost NP-complete (preliminary version)," in *Proc. 32th FOCS*, 1991, pp. 2–12.

105 A. Tsymbal, "The problem of concept drift: definitions and related work," The University of Dublin, Trinity College, Department of Computer Science, Tech. Rep., 2004.

106 D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *SDM*, 2004, pp. 442–446.

107 D. Achlioptas, Database-friendly random projections: Johnson-Lindenstrauss with binary coins, Journal of Computer and System Sciences, 66(1) (2003), pp. 671–687.

108 A. Andoni and P. Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, Communications of the ACM, 51 (2008), pp. 117–122.

109 C.-C. Chang and C.-J. Lin, LIBSVM: A library for support vector machines, ACM Trans. on Intelligent Systems and Technology, 2 (2011), pp. 27:1–27:27.

110 G. Cormode and S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, Journal of Algorithm, 55(1) (2005), pp. 5–75.

111 S. Gollapudi and R. Panigrahy, The power of two min-hashes for similarity search among heirarchical data objects, PODS, (2008), pp. 211–219.

112 P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation, Journal of the ACM, 53(3) (2006), pp. 307–323.

113 A. Karatzoglou, A. J. Smola, M. Weimer, Collaborative filtering on a budget, Journal of Machine Learning Research - Proceedings Track, 9 (2010), pp. 389–396.

114 A. Rajaraman and J. D, Ullman, Finding similar items, in Mining of Massive Datasets, Cambridge University Press, New York, 2012.

115 Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, Hash kernels for structured data, Journal of Machine Learning Research, 10 (2009), pp. 2615–2637.

116 K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, Feature hashing for large scale multitask learning, ICML, (2009), pp. 1113–1120.