3-2016

Sequential pattern mining with uncertain data

Jiaqi Ge
Purdue University

Recommended Citation

Ge, Jiaqi, "Sequential pattern mining with uncertain data" (2016). *Open Access Dissertations*. 650.
https://docs.lib.purdue.edu/open_access_dissertations/650

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Jiaqi Ge

Entitled
Sequential Pattern Mining with Uncertain Data

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Yuni Xia Chair	Christopher Clifton
Sunil Prabhakar Co-chair	
Snehasis Mukhopadhyay	
Jennifer Neville	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Yuni Xia

Approved by: Sunil Prabhakar 03/13/2016
Head of the Departmental Graduate Program Date

SEQUENTIAL PATTERN MINING WITH UNCERTAIN DATA

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jiaqi Ge

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2016

Purdue University

West Lafayette, Indiana

To my beloved wife J. Shi

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Ge, Jiaqi Ph.D., Purdue University, May 2016. Sequential Pattern Mining with Uncertain Data. Major Professor: Yuni Xia.

In recent years, a number of emerging applications, such as sensor monitoring systems, RFID networks and location based services, have led to the proliferation of uncertain data. However, traditional data mining algorithms are usually inapplicable in uncertain data because of its probabilistic nature. Uncertainty has to be carefully handled; otherwise, it might significantly downgrade the quality of underlying data mining applications.

Therefore, we extend traditional data mining algorithms into their uncertain versions so that they still can produce accurate results. In particular, we use a motivating example of sequential pattern mining to illustrate how to incorporate uncertain information in the process of data mining. We use possible world semantics to interpret two typical types of uncertainty: the tuple-level existential uncertainty and the attribute-level temporal uncertainty. In an uncertain database, it is probabilistic that a pattern is frequent or not; thus, we define the concept of probabilistic frequent sequential patterns. And various algorithms are designed to mine probabilistic frequent patterns efficiently in uncertain databases. We also implement our algorithms on distributed computing platforms, such as MapReduce and Spark, so that they can be applied in large scale databases.

Our work also includes uncertainty computation in supervised machine learning algorithms. We develop an artificial neural network to classify numeric uncertain data; and a Naïve Bayesian classifier is designed for classifying categorical uncertain data streams. We also propose a discretization algorithm to pre-process numerical uncertain data, since many classifiers work with categoric data only. And experimental results in both synthetic and real-world uncertain datasets demonstrate that our methods are effective and efficient.

1 INTRODUCTION

In recent years, many new applications such as sensor networks, RFID networks and location-based service, continouesly produce a large volume of data. Available data in these real-world applications are usually inaccurate or imprecise. Data uncertainty may be caused by reality limitations such as measurement precision limitation, sampling discrepancy, or other errors. And it might also be inherent in nature because data are vaguely specified in some applications. Examples of uncertain data sources are listed as follows:

- *Uncertainty may be from limitations of devices.* For example, sensor networks produce noisy data because of sensors' limited precision; the indistinctness between number '1' and letter 'l' may introduce uncertainty in OCR systems; an RFID antenna can only detect a tag with a certain probability within its working range. And uncertainty from reality limitation usually can be known from device specifications.

- *Statistic models output a large amount of uncertain data.* For example, sentiment analysis estimates users' uncertain attitudes towards various products [1]; in structured information extraction, uncertainty comes from rules of extracting patterns from unstructured data [2–4]. This type of uncertainty can be estimated from underlying algorithms.

- *Uncertainty might be a result of data aggregation.* In wireless sensor networks, instead of sending out every record, a sensor often aggregates all the data points within a time period to a probability distribution and then transfers the distribution via network, to reduce bandwidth consumption.

- *Data uncertainty arises because of granularity mismatch.* For instance, when an event is measured in one granularity and is recorded in a system with a finer granularity, we are not able to know exactly when a particular event happens. [5]

- *Uncertainty is generated to protect privacy and confidentiality.* In monitoring data, precise information is usually not released if there is a potential to identify individuals. In such cases, uncertainty is introduced to original data points in the form of probability distributions. Recent privacy models [6] are built to be friendly to uncertain data mining techniques.

The explosion of uncertain data creates a need of uncertain data mining applications. Directly applying traditional data mining techniques requires uncertain data to be summarized into atomic values; however, differences between summarized values and the actual values may affect the quality of underlying data mining applications. In order to obtain accurate results from uncertain data, we need to incorporate uncertain information in data mining algorithms so that the output of these methods can be closer to the results obtained as if actual data were used in the mining process [7]. And first of all, we need a model to represent uncertain data.

1.1 Uncertain Databases

Uncertain data modeling is well studied in literature [8,9]. Here we adopt a general model, which is called *uncertain database*, to represent different types of uncertainty. In uncertain databases, uncertain information is modeled by a probabilistic space whose possible outcomes are all the traditional certain instances. Given a probability space $(\Omega, \mathcal{F}, \mathbb{P}[])$, Ω is the set of outcomes and the σ-field of events $\mathcal{F}$ consists of all subsets of Ω. Its equivalent formulation is represented by pairs (Ω, p), where the outcome probability assignment $p : \Omega \to [0, 1]$ satisfies $\sum_{\omega \in \Omega} p(\omega) = 1$. [10] And then we have $\mathbb{P}[A] = \sum_{\omega \in A} p(\omega)$. We also adopt the definition of probabilistic databases in [10] to define uncertain databases as follows:

Definition 1.1 *An uncertain database is a probability space whose outcomes are possible certain database instances consistent with a given schema.*

To interpret an uncertain database, we apply the *possible world model.* An uncertain database generates *possible worlds*, each of which is a conventional database instance. However, directly expanding an uncertain database D is usually infeasible in data mining applications because the number of possible instances can be exponential to $|D|$. Therefore, the natural solution is to use a variety of simplified models which can be easily used for data mining purposes [11].

Assumptions of independence are often applied to simplify uncertain data models. There are two typical independence assumptions: (1) *Independent tuple existence.* The presence or absence of a tuple in the database is probabilistically independent with other tuples. (2) *Independent attribute value.* The value of a probabilistic attribute in a data instance is independent with the values of other attributes. With these two independence assumptions, we define the following types of data uncertainty:

- *Tuple-level uncertainty.* In an uncertain database, the presence of a tuple is probabilistic and can be modeled by an existential probability. Therefore, the tuple-level uncertainty is also called *existential uncertainty.* Considering the assumption of independent tuple existence, we assume that the presences of difference tuples are probabilistically independent.

- *Attribute-level uncertainty.* In this case, a number of tuples and their modeling are already determined. The value of an individual attribute in each tuple is probabilistic and can be modeled by probability distribution functions. Under attribute-level independence assumption, the selection of a possible value for one attribute has no influence with that of other attributes.

Table 1.1 shows an example of event databases with tuple-level uncertainty. The uncertain presence of each event e is modeled by an existential probability $P(e) \in (0, 1]$. For example, the probability that the event $e_2 = \{2, 3\}$ is present in the

Table 1.1.
An example of event database with tuple-level uncertainty

#	Event	Probability
1	$\{1\}$	1.0
2	$\{2,3\}$	0.8
3	$\{3,4\}$	0.6

Table 1.2.
An example of event database with attribute-level uncertainty

#	Event	Time
1	$\{1\}$	$2:0.4, 3:0.6$
2	$\{2,3\}$	$6:1.0$
3	$\{3,4\}$	$8:0.3, 9:0.7$

database is $P(e_2) = 0.8$. And the presence of different events in Table 1.1 are assumed to be mutually independent. In Table 1.2, it shows an example of event databases with attribute-level uncertainty. The uncertain occurring time of each event is represented by a probability mass function here. For example, the event $e_1 = \{1\}$ occurs at time $t = 2$ with probability 0.4, and the probability that e_1 happens at time $t = 3$ is 0.6.

1.2 Semantics

Most uncertain data mining applications require computation over a large number of probabilities. And the following two semantics are widely used in solving uncertain data mining problems:

- *Intensional semantics.* It enumerates all the possible worlds of an uncertain database with considering the dependencies between instances and attributes.

Data mining applications are then able to be applied to each of these possible worlds. This approach always yields correct results, though its evaluation time complexity is exponential.

- *Extensional semantics.* This is a heuristic approach that attempts to approximate the result of data mining applications in uncertain databases without enumerating the entire possible world table. This approach represent uncertainty as a generalized value of a probability distribution function (pdf) and try to evaluate the uncertainty of a given expression based on these pdfs.

Though intentional semantics can always yield the correct results, a lot of data mining applications have NP-complete complexity under intensional semantics [12]. Meanwhile, extensional semantics is mostly useful for evaluating simple expressions. Consider the uncertain sequence in Table 1.1. The aim is to compute the probability that both e_2 and e_3 are present in the database. In intensional semantics, it requires to explicitly compute the joint probability $P(e_2, e_3)$, which depends on the correlations among the events. However, with the assumption of tuple independence, an efficient extensional approach would directly compute $P(e_2, e_3)$ as $P(e_2) * P(e_3) = 0.48$.

While the extensional semantics provides an efficient heuristic, it need to be carefully designed to approximate the correct results. In sum, an uncertain data model should have the advantages in representing the probabilistic intrinsic of uncertain data and meanwhile reducing the complexity of data mining algorithms.

1.3 Challenges

The emerging uncertain data mining applications can abstract reasonable and valuable knowledge by mining large volumes of uncertain data to find the "diamonds in the dirt". [13] Since data values are no longer atomically represented in uncertain databases, in order to directly apply traditional data mining algorithms, uncertain data have to be summarized into atomic values in pre-processing steps. For example, in moving-object tracking application, the location of an object is usually summarized by its last recorded location. Unfortunately, discrepancy in the summarized values and the actual values could seriously affect the performances of data mining algorithms. [14] In order obtain an accurate understanding of uncertain data, we incorporate the two types of data uncertainty (tuple-level and attribute-level uncertainty) in data mining applications so that we can reduce the influence of noises. However, this process is not straight forward. And efficiency and scalability challenges arise in solving uncertain data mining problems.

The computational complexity of mining uncertain data is much higher than that in traditional datasets. Uncertain data are oftern represented by pdfs, which contain much more information than atomic values. Even basic operations become complex in computation. For example, the aggregation operator *sum* in traditional database can be done in linear time; however, in an uncertain database D, the computational complexity of *sum* is exponential to $|D|$, since we need to enumerate a large number of possible worlds to compute the probability distribution of the aggretation of many uncertain attribute values. Therefore, efficiency turns out to be the most critical issue in designing uncertain data mining algorithms.

Meanwhile, in order to reduce the influence of noise and obtain more useful information, uncertain data mining methods are usually applied to large scale databases so that they can discover precise knowledge from noisy data. And many recent real world applications, such as sensor networks and online shopping websites, generate a large amount of data (also known as *big data*), and these create an emerging need of mining large scale uncertain databases. Thus, scalability becomes another challenge in developing uncertain data mining algorithms.

In this dissertation, we aim to mine precise results from uncertain databases by conquering these challenges. We use motivating examples of sequential pattern mining, classification and discretization to illustrate how uncertainty can be incorporated in various types of data mining applications. The dissertation is organized as follows. In Chapter 2, we propose a sequential pattern mining algorithm in sequence databases with attribute-level temporal uncertainty. In Chapter 3, we develop an iterative MapReduce algorithm to mine gap-constraint frequent sequential patterns in large scale sequences databases with tuple-level existential uncertainty. In order to further improve the efficiency and scalability, a dynamic programming approach of mining sequential patterns is designed and extended to distributed computing platform Spark in chapter 4. An artificial neural network classifier for uncertain data (UNN) is proposed in Chapter 5; a Naïve Bayesian classifier is designed in Chapter 6 for classifying categorical uncertain streams. We design a discretization algorithm to pre-process numerical uncertain data in chapter 7. The summary is in Chapter 8.

2 SEQUENTIAL PATTERN MINING IN DATABASES WITH TEMPORAL UNCERTAINTY

2.1 Introduction

Sequential pattern mining (SPM) is an important data mining application which provides inter-transactional analysis for timestamped data. SPM is often applied to discover patterns in sequence databases which are widely used to model shopping sequences, medical syndromes and treatments, natural disasters, stock markets, and so on. For example, supermarkets collect customer purchase histories and use SPM methods to reveal customer purchasing patterns, which can be informally represented as: if a customer buys an item A, he/she will buy another item B within a certain time period. Most of the existing work fundamentally relies on an assumption that event occurrences are either in a total order [15–20] or in a strict partial order [21–25] based on precise event occurring time. However, this assumption fails in many real world applications because event time can be inaccurate or even unknown for a variety of reasons:

- *The exact time of an event is often unknown.* For instance, *Yahoo! finance* collects the highest and lowest prices of stocks every day, from which we can detect an event such as "price grows more than 8% in one day". However, the exact time of this event is unknown because the system does not record when the stocks are traded at the highest or lowest prices. And this type of event is usually assumed to occur equally likely at any time during the day.

- *Temporal uncertainty arises because of granularity mismatch.*

  Example 2.1 *In a GPS monitoring system, a handheld GPS device* $\mathcal{G}_a$ *may update its position every 10 minutes; while a GPS* $\mathcal{G}_b$ *mounted on a fast-moving* *vehicle might report the position every 5 seconds. Then, an event* e_a *reported by* $\mathcal{G}_a$ *can occur anywhere in a 10-minute period (e.g. 09:00:00* $\sim$ *09:10:00), and an event* e_b *reported by* $\mathcal{G}_b$ *occurs randomly within a 5-second period (e.g. 09:01:00* $\sim$ *09:01:05).*

It is difficult to determine whether e_a occurs before or after e_b, because the temporal relationship between two events defined in different granularities becomes uncertain and cannot be modeled by either a total or partial order.

- *Temporal uncertainty is added to protect privacy and confidentiality.* Precise time information in monitoring data is often not released if there is a potential to identify individuals. Obfuscation techniques deliberately degrade temporal information, using uncertainty to protect privacy.

In traditional temporal databases, valid-time indeterminacy is modeled by a sequence of consecutive *chronons* as $t_0, t_1, \ldots, t_N$, where a chronon is the smallest time unit in the system [26]. However, this model becomes inefficient when data are collected under different time scales. In Example 2.1, if the chronon is set to be a *second*, the uncertain time of an event reported by the on-vehicle device $\mathcal{G}_B$ may be represented by five consecutive timestamps (e.g. $t_1, t_2, \ldots, t_5$) with equal probabilities; while the timestamp of an event reported by the handheld device $\mathcal{G}_A$ is represented by a sequence of hundreds of timestamps (e.g. $t_1, \ldots, t_{600}$), which is neither efficient nor convenient.

Instead of relying on chronons, we propose our *temporally uncertain model* to efficiently represent uncertain event times by *random variables.* If the timestamp of an event equally likely occurs at any point in a time period, it is reasonable to model it by a uniform probability density function (pdf). Furthermore, for any arbitrary shaped pdf, we approximate it by discrete probability mass functions (pmf) using sampling and histogramming techniques.

In temporally uncertain sequence databases, it is much more difficult to identify frequent sequential patterns because orders of events are uncertain. We adopt *possible* *world semantics* from probabilistic databases [27–29] to interpret our uncertain model by a set of certain databases. However, the number of certain databases derived from a temporally uncertain database is infinite because we use continues pdf to represent data uncertainty in our model. Besides, aggregating probabilities associated with possible worlds requires the computation of multivariable integration, which brings efficiency and scalability challenges to the uncertain SPM problem.

Another challenge comes from integrating gap constraints into SPM process. A gap constraint requires the pattern appears frequently in the database such that the time difference between every two adjacent events must be longer or shorter than a given gap. [30] Incorporating gap constraints (e.g. minimum-gap and maximum-gap) in SPM can help to mine user-interested patterns; however, it makes the management of temporal uncertainty more complicated. For example, suppose event A occurs equally likely in the range of $[1, 5]$ and event B occurs within $[6, 10]$, then it is certain that B occurs after A; however, after applying minimal-gap ($g_l = 2$) and maximal-gap ($g_h = 5$), it becomes complicated to check whether B occurs after A with satisfying gap-constraints or not. And we will propose a solution of this problem in section 2.5.

In this chapter, we address the SPM problem in temporally uncertain databases. And the major contributions are summarized as follows:

(1) Besides representing temporal uncertainty by uniformly distributed random variables, we also use a discrete probability mass function (pmf) to approximate the distribution of any arbitrary shaped uncertainty and propose a general solution based on this model. This approximation is proved to be effective and efficient, according to the experimental results.

(2) We develop a novel approach to compute temporal uncertainty during the SPM process which efficiently calculate multivariable pdfs/pmfs for uncertain timestamps.

(3) We incorporate gap constraints to mine user-interested sequential patterns with user-specified thresholds.

(4) We develop new pruning techniques to speed up the computation.

(5) We conduct extensive experiments on both synthetic and real datasets to test and

prove the efficiency and scalability of the proposed algorithms. We also apply our method on a real world stock market dataset and analyze the results via visualization techniques.

2.2 Related Works

2.2.1 Traditional Sequential Pattern Mining

Sequential pattern mining problem in traditional deterministic databases has attracted a lot of attention. There have been many algorithms on efficient sequential pattern mining and its applications [31–38]. In general, sequential pattern mining algorithms can be categorized into three classes: (1) Apriori-like algorithm with horizontal data format, e.g. GSP [32]; (2) Apriori-like algorithm with vertical data format, like SPADE [33]; (3) Projection-based pattern growth algorithm, like PrefixSpan [35].

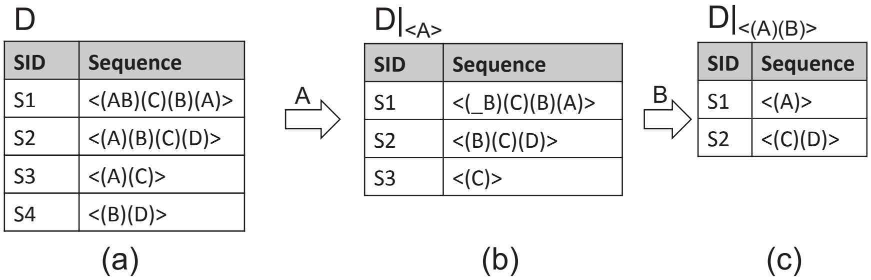
PrefixSpan is proved to be more efficient than other Apriori-like algorithms such as GSP due to its prefix-projection technique. Here we briefly review the framework of PrefixSpan, which is related to our proposed algorithms. We first present the following definitions.

Definition 2.2.1 *A sequential pattern* $s = \langle s_1, s_2, \ldots, s_n \rangle$ *is a temporally ordered sequence of itemsets, where* $s_i \in s$ *is called an* element *of* s*.*

Definition 2.2.2 *A sequential pattern* $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$ *is a* sub-sequence *of a sequence* $\beta = \langle b_1, \ldots, b_m \rangle$*, denoted by* $\alpha \sqsubseteq \beta$*, if there exist* n *integers* $1 \leq k_1 < \cdots < k_n \leq m$ *such that* $a_i \subseteq b_{k_i}$*.*

For example, $\langle (a)(b) \rangle$ is a sub-sequence of $\langle (ab)(bc) \rangle$. Without loss of generality, items in an element are assumed to be ordered alphabetically.

Definition 2.2.3 *A sequential pattern* $\alpha = \langle a_1, \ldots, a_m \rangle$ *is a prefix of a sequence* $\beta = \langle b_1, \ldots, b_n \rangle$ *(*$m \leq n$*) if (1)* $a_i = b_i$*,* $\forall i \in [1, m-1]$*; (2)* $a_m \subseteq b_m$ *and all items in* $b_m - a_m$ *are alphabetically larger than those in* a_m*.*



D

SID	Sequence
S1	<(AB)(C)(B)(A)>
S2	<(A)(B)(C)(D)>
S3	<(A)(C)>
S4	<(B)(D)>

(a)

D|<A>

SID	Sequence
S1	<(_B)(C)(B)(A)>
S2	<(B)(C)(D)>
S3	<(C)>

(b)

D|<(A)(B)>

SID	Sequence
S1	<(A)>
S2	<(C)(D)>

(c)

Figure 2.1. Example of database projection in PrefixSpan

For example, both $\langle(a)\rangle$ and $\langle(ab)(b)\rangle$ are prefixes of $\langle(ab)(bc)\rangle$. And for ease of presentation, we denote $\alpha\beta$ to be a sequence resulted from appending sequence β to α.

Definition 2.2.4 *Given a pattern α and a sequence s, the α-*projected sequence *$s|_\alpha$ is defined to be the suffix γ of s such that $s = \beta\gamma$ with β being the minimal prefix of s satisfying $\alpha \sqsubseteq \beta$. And the α-*projected database *$D|_\alpha$ is defined to be a collection of projected sequences $\{s_\alpha | s \in D \wedge s_\alpha \neq \phi\}$.*

Consider the sequence database D shown in Figure 2.1(a). The $\langle A\rangle$-projected database $D|_{\langle A\rangle}$, shown in Figure 2.1(b), is built by projecting item A from each sequence. For example, $\langle(\_B)(C)(B)(A)\rangle$ is the suffix after removing prefix $\langle A\rangle$ from sequence s_1. Notice that $(\_B)$ means that the last element in the prefix, which is A here, together with item $\_B$, form one element (AB). And we add the notation $\_$ ahead to distinguish it from a regular item B. Suppose B is a frequent item in $D|_{\langle A\rangle}$, then PrefixSpan recursively constructs $\langle(A)(B)\rangle$-projected database, as shown in Figure 2.1(c). For example, the projected sequence $s_1|_{\langle(A)(B)\rangle} = \langle A\rangle$ is generated by projecting prefix $(A)(B)$ from s_1. PrefixSpan adopts a depth-first strategy to find frequent sequential patterns with growing lengths and it stops until all frequent patterns are found.

2.2.2 Uncertain Sequential Pattern Mining

Interval-based SPM. An event that do not occur at a time point but last for a period of time can be modeled by a time interval which represents the duration of the event. In [21], Allen et al. introduce thirteen temporal relationships between two time intervals, and many algorithms have been designed to mine Allen's relations from data with interval-based timestamps [22–24]. However, events in these algorithms still have precise timestamps and are usually imposed to have a strict partial order. In contrast, our work deals with events that occur at a time point but with uncertain timestamps. When a partial order is pre-defined, some possible orders of events are not allowed, which will cause the loss of information.

SPM with existential uncertainty. Muzammal and Raman proposed an SPM algorithm in probabilistic database using the expected support as the measurement of pattern frequentness [17]. Zhao et al. measure pattern frequentness in possible world semantics and propose a pattern-growth uncertain SPM algorithm [15,16]. Sun et al use approximation with probabilistic guarantee to improve the efficiency of mining uncertain frequent itemsets [19]. Dynamic programming is used to mine frequent serial episodes in an uncertain sequence [39] and probabilistic spatial-temporal frequent sequential patterns [40]. All these methods are designed for dealing with existential uncertainty in databases with accurate timestamps.

Indeterminate temporal database. Dyreson and Snodgrass introduced indeterminate semantics which models valid-time indeterminacy by a set of consecutive timestamps with equal probabilities [26]. Zhang et al. proposed a pattern recognition algorithm in temporal uncertain streams [41], and pattern queries in temporal uncertain sequences is studied in [25]. Our work distinguishes from the above in that we use random variables to represent uncertain timestamps. A random variable is more flexible and efficient in modeling data collected from different scales. Meanwhile, the above work focused on matching patterns in one sequence, while our work addresses mining patterns from a large number of sequences. In [20], Sun et al. introduce one type of uncertain event into Apriori-like sequential pattern discovery with only considering the total order of events; however, every event in our model is uncertain and we do not assume any total or partial orders of events. In addition to our preliminary work [42], we extend the model of temporal uncertainty from uniform distributions to any arbitrary shaped distributions and propose a general solution based on this model; we develop new pruning techniques to help improve efficiency; we also apply our algorithm to a real stock dataset and analyze the results via visualization techniques.

2.3 Problem Statement

2.3.1 Temporally Uncertain Model

The temporal uncertain model applied in this chapter is based on temporally uncertain events.

Definition 2.3.1 *A* temporally uncertain event *is an event whose occurance time is uncertain and can be represented by a random variable.*

Definition 2.3.2 *An* uncertain sequence *is a list of temporally uncertain events. An* uncertain sequence database *is a collection of uncertain sequences.*

We represent a temporally uncertain event by $e = \langle sid, eid, T, I \rangle$, where sid is the sequence-id, eid is the event-id and I is an itemset that describes the content of event e. T is a random variable representing the uncertain event time of e. And in this paper, we consider the following two models of temporal uncertainty :
(1) T is modeled by a uniform probability density function(pdf) over a range, denoted by $T \sim U[t^-, t^+]$; (2) T is modeled by a discrete probability mass function (pmf), denoted by $\{T | t_1 : p_1, \cdots, t_n : p_n\}$, where p_i is the probability that $T = t_i$.

We denote an event with $sid = i$ and $eid = j$ by e_{ij} and denote its uncertain event time by T_{ij}. Figure 2.2 shows two examples of temporally uncertain databases in a logging system which monitors a large number of distributed computers. In

Sid	eid	T	I
1	1	[50, 60]	{Client A: Connection Lost}
1	2	[55,70]	{Client B: Connection established}
2	1	[160, 170]	{Client C: Sleep}
2	2	[165, 175]	{Client D: Connection Lost}
2	3	[180, 190]	{Client E: Wake up}

(a) uncertain time modeled by uniform pdf

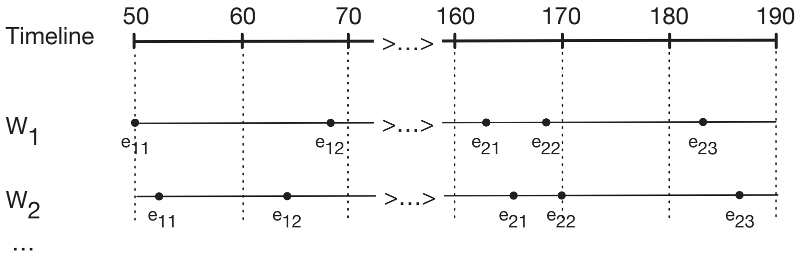
Sid	eid	T	I
1	1	{100: 0.2, 101: 0.5, 102: 0.3}	{ssh, http-web}
1	2	{103: 0.2, 104: 0.6, 105: 0.2}	{ftp}
1	3	{107: 0.2, 108: 0.6, 109: 0.2}	{ssh}
2	1	{33: 0.2, 34: 0.5, 35: 0.3}	{buffer-overflow}
2	2	{35: 0.2, 36: 0.6, 37: 0.2}	{ftp, ssh}
2	3	{38: 0.2, 39: 0.6, 40: 0.2}	{smtp-mail, ftp}

(b) uncertain time modeled by discrete pmf

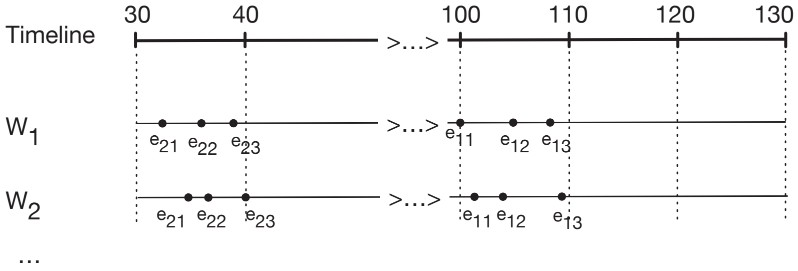
Figure 2.2. Examples of uncertain sequence databases in logging systems

Figure 2.2(a), each sequence is a list of events that are detected by a server. Suppose a server pings its clients periodically to test connections. For example, the event $e_{11} = \{$*Client A: Connection lost*$\}$ is detected because the server can connect the client at time 50 but fails to reach it at time 60. The time of occurrence of e_{11} is equally likely to be at anywhere between 50 and 60 and is modeled by the uniform distribution $T_{11} \sim U(50, 60)$. For simplicity's sake, we write $U[t^-, t^+]$ as $[t^-, t^+]$ in Figure 2.2(a).

Figure 2.2(b) records sequences of actions in a cluster of distributed computers. The exact time of these actions is unknown because of the network latency. For example, suppose the network latency δ_t is a Gaussian noise $\delta_t \sim N(-4, 1)$, then the occurrence time of event e_{11}, which is recorded at time 105, is $T_{11} = 105 + \delta_t$. However, in real applications, it is very expensive and sometimes infeasible to obtain the exact distribution of an arbitrary shaped noise. It is more practical to approximate the underlying continues distribution by a discrete pmf based on sampling and/or



(a) possible worlds of uncertain database in Figure 2.2(a)



(b) possible worlds of uncertain database in Fig 2.2(b)

Figure 2.3. Examples of possible worlds of the temporal uncertain database

histogramming methods. For example, the pmf of T_{11} is approximated by $\{100 : 0.2, 101 : 0.5, 102 : 0.3\}$ in Figure 2.2(b).

A sequential pattern $s = \langle s_1, \ldots, s_n \rangle$ is a sequence of itemsets, where s_i is called an *element* of s. For example, a sequence of actions $\langle$(*buffer-overflow*)(*ssh*)(*ftp*)$\rangle$ in Figure 2.2(b) is a sequential pattern which corresponds to a web-based attack followed by copying data from the host computer to remote destination via *ftp*.

2.3.2 Temporal Possible World Semantics

An uncertain database D is interpreted by a set of possible worlds under possible world semantics. A temporal possible world is a certain sequence database with point-value timestamps drawn from the pdfs/pmfs of uncertain timestamps.

Pdf-Modeled Uncertainty. A sequence database with pdf-modeled uncertain timestamps derives an infinite number of possible worlds. Figure 2.3(a) shows two example possible worlds that are instantiated from the uncertain database in Figure 2.2(a). In Figure 2.3(a), each event time is certain and is drawn from the corresponding uniform distribution. E.g., the event time $t_{11} = 50$ in w_1 is instantiated from $T_{11} \sim U(50, 60)$ in Figure 2.2(a).

The pdf of a possible word w is $f_D(w) = f(\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_n)$, where $\hat{d}_i \in w$ is a certain sequence instantiated from the uncertain sequence d_i. It is widely assumed that uncertain sequences are mutually independent, which is known as the *tuple-level independence* [28, 43] in probabilistic databases. Event times are also assumed to be independent [15, 29, 44, 45], which can be justified by the fact that events are usually observed independently in real applications. With these assumptions, the computation of $f_D(w)$ can be simplified, as shown in Equation (2.1).

$$f_D(w) = \prod_{i=1}^{|D|} f(d_i = \hat{d}_i) = \prod_{i=1}^{|D|} \prod_{j=1}^{|d_i|} f_{T_{ij}}(t) \tag{2.1}$$

Here $|D|$ is the number of sequences in D, $|d_i|$ is the number of events in sequence d_i. $f_{T_{ij}}(t)$ is the pdf of $T_{ij} \sim U(t^-, t^+)$, as shown in Equation (2.2).

$$f_{T_{ij}}(t) = \begin{cases} \frac{1}{t^+ - t^-} & , t \in [t^-, t^+] \\ 0 & , \text{otherwise} \end{cases} \tag{2.2}$$

Pmf-Modeled Uncertainty. A sequence database with discrete temporal uncertainty is interpreted by a finite set of possible words. Figure 2.3(b) shows two possible world examples derived from the example in Figure 2.2(b). In the possible world w_1, the event time $t_{11} = 101$ is instantiated from the discrete pdf $\{T_{11} | 101 : 0.2, 102 : 0.5, 103 : 0.3\}$ in Figure 2.2(b). With the independence assumptions, we can compute the probability mass function (pmf) of w in Equation (2.3).

$$f_D(w) = \prod_{i=1}^{|w|} P(d_i = \hat{d}_i) = \prod_{i=1}^{|w|} \prod_{j=1}^{|d_i|} f_{T_{ij}}(t) \tag{2.3}$$

where $f_{T_{ij}}(t) = P(T_{ij} = t)$ is the probability that T_{ij} equals to t.

2.3.3 Uncertain SPM Problem

In traditional certain database, a sequential pattern s is *supported* by a sequence d, denoted by $s \preceq d$, if and only if it satisfies: (1) there is an occurrence of s in d; (2) this occurrence satisfies gap constraints.

A sequential pattern $s = \langle s_1, \ldots, s_n \rangle$ occurs in a sequence $d = \langle e_1, \ldots, e_m \rangle$ if and only if s is a subsequence of d. Therefore, there exist n integers $1 \le k_1 < \cdots < k_n \le m$ which have $s_i \subseteq e_{k_i}$, and the ordered set of events $o = \langle e_{k_1}, \cdots, e_{k_n} \rangle$ is an occurrence of s in d. For example, let $d = \langle (a)(b)(cd)(ef) \rangle$ and $s = \langle (a)(b)(c) \rangle$, then $s \sqsubseteq d$ and $o = \{(a), (b), (cd)\}$ is an occurrence of s in d.

A pattern occurrence satisfies gap constraints if the occurrence times of its adjacent events are longer or shorter than a given gap [30]. Given the minimum gap g_l and the maximum gap g_h, an occurrence o satisfies the constraints if and only if $g_l \le T_{k_{i+1}} - T_{k_i} \le g_h$ for $\forall i \in [1, n)$, where T_{k_i} is the timestamp of e_{k_i}.

The support of a pattern s, denoted by $sup(s)$, is the number of sequences that support it. And s is *frequent* in a deterministic database if and only if $sup(s) \ge \tau_s$, where τ_s is the user-defined minimum threshold. However, the frequentness of s in an uncertain database D is probabilistic. And we define *probabilistic frequent sequential pattern* as follows:

Definition 2.3.3 *A sequential pattern s is a* probabilistic frequent pattern *(p-FSP) if and only if its probability of being frequent is at least τ_p, denoted by $P\big(sup(s) \ge \tau_s\big) \ge \tau_p$.*

Here τ_p is the user-defined minimum confidence in the frequentness of a sequential pattern. And $P(sup(s) \ge \tau_s)$ is the accumulation of existential probabilities of possible worlds in which s is frequent. Depending on the model of temporal uncertainty, it can be computed by Equation (2.4a) if temporal uncertainty is pdf-modeled or Equation (2.4b) if discrete pmf is adopted.

$$P(sup(s) \geq \tau_s) = \int\limits_{sup(s|w) \geq \tau_s} f_D(w)\mathrm{d}w \tag{2.4a}$$

$$P(sup(s) \geq \tau_s) = \sum_{sup(s|w) \geq \tau_s} f_D(w) \tag{2.4b}$$

Here $sup(s|w) \geq \tau_s$ indicates that s is a frequent pattern in a possible world w. And the SPM problem in temporal uncertain databases is defined as follows:

Given thresholds τ_s, τ_p and gap constraints g_l, g_h, return all p-FSPs in temporally uncertain database D.

However, directly expanding all possible worlds of D is usually infeasible in practice, because the number of possible worlds can be infinite. And our goal is to efficiently discover p-FSPs without enumerating all possible worlds.

2.4 Uncertain Sequential Pattern Mining Algorithm

In this section, we propose our uncertain sequential pattern mining (USPM) algorithms in temporally uncertain databases.

2.4.1 SPM Framework

Here we first define two types of pattern extension as follows:

Definition 2.4.1 *An* item-extended pattern *s is a sequential pattern which is generated by appending an item i to the last element of another pattern s', denoted by $s = s' \cup \{i\}$.*

Definition 2.4.2 *A* sequence-extended pattern *s is a sequential pattern generated by appending an itemset $\{i\}$ to another pattern s' as its last element, denoted by $s = s' + \{i\}$.*

E.g., given $s' = \langle(a)(b)(c)\rangle$ and an item d, $s_1 = \langle(a)(b)(cd)\rangle$ is an item-extended pattern of s; while $s_2 = \langle(a)(b)(c)(d)\rangle$ is a sequence-extended pattern.

The anti-monotonicity property of p-FSPs in Lemma 4.1 allows us to prune a sequential pattern if it is extended from a pattern that is not a p-FSP.

Lemma 2.1 *If s is extended from s' and s is a p-FSP, then s' is also a p-FSP.*

Proof In a possible world w, if s is frequent in w, s' is also frequent because $s' \sqsubseteq s$. Thus, $P(sup(s') \geq \tau_s) \geq P(sup(s) \geq \tau_s)$. Since s is a p-FSP, we have $P(sup(s') \geq \tau_s) \geq P(sup(s) \geq \tau_s) \geq \tau_p$. Therefore, s' is a p-FSP. ■

In Algorithm 1, we extend the PrefixSpan framework to temporally uncertain databases and adopt a depth-first strategy to search p-FSPs. Suppose s is a p-FSP and $D|_s$ is the s-projected database. For each item i in $D|_s$, we generate a sequence-extended pattern $s' = s + \{i\}$; while for each item $\_i$, we item-extend s to generate $s' = s \cup \{i\}$. Thereafter, we use a *project* function to build the s'-projected database $D|_{s'}$. Then, a *freqProb* function is applied to $D|_{s'}$ to compute the frequentness probability of s'. If s' is a p-FSP, we add it to L and call a recursive function to continue search frequent patterns by $D|_{s'}$. Here L is a set of all p-FSPs. The detailed design of *project* and *freqProb* functions are described in the following sections.

2.4.2 Approximate Frequentness Probability

In an uncertain database D, the probabilistic support $sup(s)$ can be represented by a random variable. Since sequences are assumed to be mutually independent, $sup(s)$ is the sum of n independent random variables, as shown in Equation (2.5),

$$sup(s) = \sum_{i=1}^{n} sup(s|d_i) \tag{2.5}$$

where $d_i \in D$ is an uncertain sequence. $sup(s|d_i) \sim \mathcal{B}(1, p_i)$ is a Bernoulli random representing the probabilistic support of s in d_i, where $p_i = P(s \preceq d_i)$ is so called the *support probability* of s in d_i. We will discuss its computation in section 2.4.3.

Algorithm 2.1: USPM

```
Input: $s$: a p-FSP, $D|_s$: $s$-projected uncertain database
       $\tau_s$: minimal support, $\tau_p$: minimal frequentness probability
Output: $L$: a set of p-FSPs
$L \leftarrow \phi$
foreach item $i \in D|_s$ do
    $s' \leftarrow$ extend $s$ by item $i$
    $D|_{s'} \leftarrow$ project($i$, $D|_s$)                  // build $s'$-projected database
    $P(sup(s') \geq \tau_s) \leftarrow$ freqProb($i$, $D|_{s'}$)       // compute frequentness
    probability of $s'$
    if $P(sup(s') \geq \tau_s) \geq \tau_p$ then
        $L \leftarrow L \cup \{s'\}$
        $L \leftarrow L \cup$ USPM($s', D|_{s'}, \tau_s, \tau_p$)
    end
end
return $L$
```

As the sum of n independent but non-identical Bernoulli trials, $sup(s)$ follows a Poisson-Binomial distribution. The Fast Fourier Transform (FFT) technique is adopted to compute the pmf of $sup(s)$ in $O(nlogn)$ time. [15] In order to further improve the efficiency, we use Gaussian distribution to approximate the underlying Poisson-Binomial distribution. This type of approximation has also been used in frequent patterns mining problems [16, 19].

Lemma 2.2 *Let $p_i = P(s \preceq d_i)$, then the overall support $sup(s)$ in a large uncertain database converges in distribution to a Gaussian random variable, shown as follows:*

$$sup(s) = \sum_{i=1}^{n} sup(s|d_i) \xrightarrow{n \to \infty} N(\sum_{i=1}^{n} p_i, \sum_{i=1}^{n} p_i(1-p_i))$$

Proof The variance of $sup(s)$ is $\sigma^2 = \sum_{i=1}^{n} \sigma_i^2$, where $\sigma_i^2 = (1-p_i)p_i$ is the variance of Bernoulli random variable $sup(s|d_i)$. Therefore, we have $\sigma^2 \to \infty$ when $n \to \infty$, which satisfies the Lindeberg's condition of the *central limit theorem.* ■

Let $\mu = \sum_{i=1}^{n} p_i$ and $\sigma^2 = \sum_{i=1}^{n} p_i(1 - p_i)$, then we can compute the *approximate frequentness probability* of s by Equation (2.6), in linear time.

$$P(sup(s) \geq \tau_s) = 1 - P(sup(s) \leq (\tau_s - 1)) = 1 - \Phi(\frac{\tau_s - 1 - \mu}{\sigma}) \tag{2.6}$$

And s is a p-FSP if it has $P(sup(s) \geq \tau_s) \geq \tau_p$.

2.4.3 Support Probabilities in Uncertain Sequences

In this section, we discuss the computation of support probability $P(s \preceq d)$ in a temporally uncertain sequence. As previously mentioned, directly expanding all possible sequences is infeasible in practice. Therefore, here we propose a new approach to compute the support probability without enumerating all possible worlds. We first define the *minimum possible occurrence* (*mpo*) as follows.

Definition 2.4.3 *Given an uncertain sequence* $d = \{e_1, \ldots, e_m\}$ *and a sequential pattern* $s = \langle s_1, \ldots, s_n \rangle$ $(n \leq m)$*, a minimum possible occurrence (*mpo*) of* s *in* d *is an ordered size-n subset* $\langle e_{k_1}, \ldots, e_{k_n} \rangle$ *of* d *which have* $s_i \subseteq e_{k_i}$.

Notice that we use $s_i \subseteq e_{k_i}$ to represent $s_i \subseteq e_{k_i}.I$, for simplicity's sake. In the following example, we use itemsets to represent uncertain events. Suppose $d = \{(a)(ab)(abc)(cd)\}$ and $s = \langle (a)(b) \rangle$, then $\langle (\mathbf{a})(a\mathbf{b}) \rangle$, $\langle (\mathbf{a})(a\mathbf{b}c) \rangle$, $\langle (a\mathbf{b})(a\mathbf{b}c) \rangle$ and $\langle (\mathbf{a}bc)(a\mathbf{b}) \rangle$ are four *mpo*s of s in d; while $\langle (a)(ab)(abc) \rangle$ is not a *mpo*. Two properties of *mpo* are presented as follows.

Lemma 2.3 *If there are no* mpo *of* s *in* d, $P(s \preceq d) = 0$

Proof If no *mpo*s of s are found in d, there is no possible world $\hat{d}$ of d having $s \sqsubseteq \hat{d}$. Therefore, there does not exist any possible world which support s, and this prove that $P(s \preceq d) = 0$. ■

Lemma 2.4 *If* s *is not potentially supported by any* mpo *of* s *in* d, $P(s \preceq d) = 0$.

Proof Let $O = \{o_1, \ldots, o_n\}$ be a set of all *mpo*s of s in d. Suppose $P(s \preceq d | s \npreceq o_1, \ldots, s \npreceq o_n) > 0$, then there must exist a possible world $\hat{d}$ that contains an occurrence o_i of s. Thus, $P(s \preceq o_i) = P(d = \hat{d}) > 0$, which contrasts with $s \npreceq o_i$, $\forall o_i \in O$. This proves the correctness of the Lemma. ■

Let $O = \{o_1, \ldots, o_n\}$ be n *mpo*s of s in d. If $O = \phi$, $P(s \preceq d) = 0$, according to Lemma 2.3; otherwise, we compute the support probability $P(s \preceq d)$ in Equation (2.7), referring to *the law of total probability.*

$$\begin{aligned} P(s \preceq d) =& P(s \preceq d | s \preceq o_1) P(s \preceq o_1) + P(s \preceq d | s \npreceq o_1) P(s \npreceq o_1) \\ =& P(s \preceq o_1) + P(s \preceq d | s \npreceq o_1)(1 - P(s \preceq o_1)) \end{aligned} \tag{2.7}$$

where $P(s \preceq d | s \preceq o_i) = 1$ by definition. We continue to decompose the probability $P(s \preceq d | s \npreceq o_k, \ldots, s \npreceq o_1)$ until obtaining Equation (2.8):

$$\begin{aligned} P(s \preceq d) = P(s \preceq o_1) + \sum_{i=2}^{n} (P(s \preceq o_i) \prod_{j=1}^{i-1} P(s \npreceq o_j)) \\ + P(s \preceq d | s \npreceq o_1, \ldots, s \npreceq o_n) \prod_{i=1}^{n} P(s \npreceq o_i) \end{aligned} \tag{2.8}$$

According to Lemma 4.3, $P(s \preceq d | s \npreceq o_1, \ldots, s \npreceq o_n) = 0$. Then Equation (2.8) can be shorten as:

$$\begin{aligned} P(s \preceq d) = P(s \preceq o_1) + \sum_{i=2}^{n} \left(P(s \preceq o_i) * \prod_{j=1}^{i-1} P(s \npreceq o_j)\right) \\ = \sum_{i=1}^{n} P(s \preceq o_i) * \mathcal{A}_i \end{aligned} \tag{2.9}$$

where we use an auxiliary variable $\mathcal{A}_i$, as shown in Equation (2.10), to track the product of $P(s \npreceq o_1), \ldots, P(s \npreceq o_{(i-1)})$. Then the amortized time complexity of Equation (2.9) is $O(n)$.

$$\mathcal{A}_i = \begin{cases} 1 & \text{, if } i = 1 \\ \mathcal{A}_{i-1} * P(s \npreceq o_{(i-1)}) & \text{, if } i \geq 2 \end{cases} \tag{2.10}$$

event	T	I
e1	1	{a}
e2	{3:0.4, 8:0.6}	{b}
e3	{4:0.5, 9:0.5}	{b}

(a) uncertain sequence d

world	T1	T2	T3	Prob	Satisfy?
w1	1	3	9	0.2	✓
w2	1	3	4	0.2	✓
w3	1	8	9	0.3	✗
w4	1	8	4	0.3	✓

(b) possible worlds $\hat{d}$

Figure 2.4. An example of computing support probability

Figure 2.4 is an example to demonstrate the process of computing $P(s \preceq d)$ by *mpo*s. Let $g_l = 1$ and $g_h = 3$. Given $s = \langle (a)(b)(c) \rangle$, we first compute $P(s \preceq d) = 0.7$ by enumerating all possible worlds of d as shown in Figure 2.4(b).

Next we compute $P(s \preceq d)$ without enumerating the possible worlds table. First, we find two *mpo*s of s, which are $o_1 = \langle e_1, e_2 \rangle$ and $o_2 = \langle e_1, e_3 \rangle$. Then, we compute $P(s \preceq o_1) = 0.4$ and $P(s \preceq o_2) = 0.5$. Finally, $P(s \preceq d)$ is computed as $P(s \preceq d) = P(s \preceq o_1) + P(s \preceq o_2) * P(s \npreceq o_1) = 0.4 + 0.5 * 0.6 = 0.7$, which is consistent with the result of expanding all possible worlds.

Function *freqProb* is used to compute frequentness probabilities from projected databases. Let $d|_s$ be a projected sequence in $D|_s$. $\mathcal{A}$ is the auxiliary variable defined in Equation (2.10). Suppose o_s is a *mpo* of s in d and $d|_{o_s}$ is the o_s-projected sequence, then $P(s \preceq o_s)$ is the probability that s is supported by o_s and we will discuss its computation in the next sections. Referring to Equation (2.9), we iterate project sequences w.r.t. all *mpo*s of s to compute the support probability $P(s \preceq d)$.

The probabilistic support of s in an uncertain sequence d_i is modeled by a Bernoulli random variable $X_i \sim \mathcal{B}(1, p_i)$, where $p_i = P(s \preceq d_i)$. As the sum of a large number of random variables, the overall support $sup(s)$ is approximated by the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. Here μ and σ^2 are the sum of the means and variances of these Bernoulli random variables. And then, Equation (2.6) is applied to compute the approximate frequentness probability $P(sup(s) \geq \tau_s)$.

Function freqProb($D|_s$)

```
Input: $D|_s$: an uncertain projected database
Output: $P(sup(s) \geq \tau_s)$
$\mu \leftarrow 0$, $\sigma^2 \leftarrow 0$
foreach $d|_s \in D|_s$ do
    $\mathcal{A} \leftarrow 1$
    $p \leftarrow 0$                    // $p$ is the support probability $P(s \preceq d)$
    foreach $d|_{o_s} \in d|_s$ do
        compute $P(s \preceq o_s)$
        $p \leftarrow p + P(s \preceq o_s) * \mathcal{A}$
        $\mathcal{A} \leftarrow \mathcal{A} * P(s \npreceq o_s)$
    end
    $\mu \leftarrow \mu + p$
    $\sigma^2 \leftarrow \sigma^2 + p * (1 - p)$
end
$P(sup(s) \geq \tau_s) \leftarrow 1 - \Phi(\frac{\tau_s - 1 - \mu}{\sigma})$
return $P(sup(s) \geq \tau_s)$
```

2.4.4 Uncertain Database Projection

Different from the traditional *PrefixSpan* method, we project temporally uncertain databases using *mpo*s. First of all, we have the following definitions.

Definition 2.4.4 *Suppose all the items in an element are listed alphabetically. Given a* mpo $o_s = \langle e_{k_1}, \ldots, e_{k_n} \rangle$ *of a pattern* s *in sequence* d*, a sequence* $\alpha = \left\langle e'_{k_1}, \ldots, e'_{k_n} \right\rangle$ *is defined to be an* uncertain prefix *of* d *w.r.t. pattern* s*, if and only if: (1)* $e'_{k_i} = e_{k_i}$ *for* $i \leq (n-1)$*; (2)* $e'_{k_n} \subseteq e_{k_n}$*; (3) all the items in* $(e_{k_n} - e'_{k_n})$ *are alphabetically after those in* e'_{k_n}*.*

Definition 2.4.5 *The* o_s-projected sequence $d|_{o_s}$ *is defined to be the suffix* β *of* d *such that* $d = \alpha\beta$ *with* α *being the prefix of* d *w.r.t.* s *derived from* o_s*.*

Consider the following example where we use only items to represent uncertain events. Let $d = \{(ab), (cd), (e)\}$ and $s = \langle(a)(c)\rangle$, then $o = \langle(ab)(cd)\rangle$ is a *mpo* of s in d, and $\langle(ab)(c)\rangle$ is an uncertain prefix derived from o. $d|_o = \{(\_d)(e)\}$ is the o-projected sequence in d, where $\_d$ indicates that the last element of the prefix, which is c here, together with d, form one element (cd).

Definition 2.4.6 *Suppose $O = \{o_1, \ldots, o_n\}$ is a set of all possible* mpos *of s in d, then a s-*projected uncertain sequence *$d|_s$ is defined to be a set of all o_i-projected sequences, denoted by $d|_s = \{d|_{o_1}, \ldots, d|_{o_n}\}$. And the s-*projected uncertain database *is a collection of all s-projected sequences, denoted by $D|_s = \{d_1|_s, \ldots, d_n|_s\}$.*

Considering gap-constraints (e.g. $g_l = l$ and $g_h = h$), if an event in $d|_{o_s}$ is not able to support any extension of s together with o_s, we can prune it according to Lemma 2.5.

Lemma 2.5 *Given a pattern s and an uncertain sequence d. Given a* mpo *$o = \langle e_{k_1}, \ldots, e_{k_n}\rangle$, an uncertain event e_i in $d|_{o_s}$ can be pruned if $P(T_i \geq T_{k_n} + g_l) = 0$.*

Proof If $P(T_i \geq T_{k_n} + g_l) = 0$, we have $P(T_{k_1}, \ldots, T_{k_n}, T_i) = 0$, which indicates that e_i does not contribute to support any extension of s. Therefore, it can be pruned. ■

Here $P(T_i \geq T_{k_n} + g_l) = 0$ is equivalent to $\max(T_i) < \min(T_{k_n}) + g_l$, where $\max(T_i)$ is the maximum possible value of T_i and $\min(T_{k_n})$ is the minimum value of T_{k_n}.

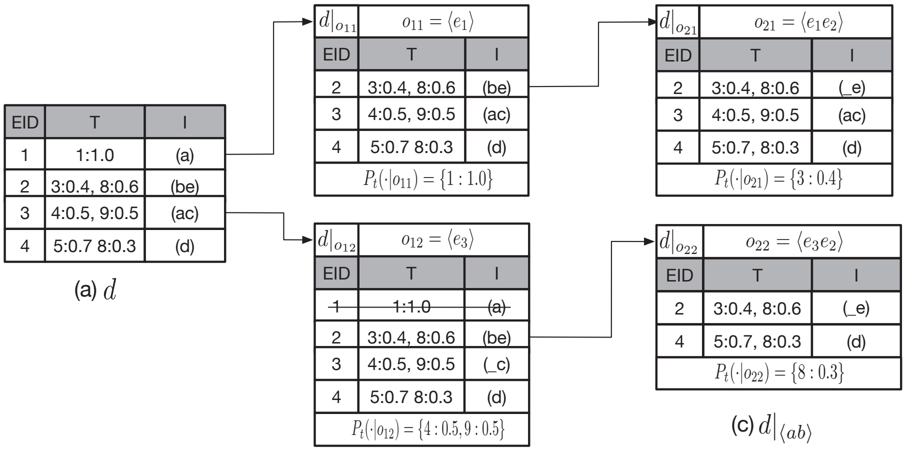
Function *project* shows the details of our uncertain project function. Here $D|_s$ is the s-projected database and i is an item in $D|_s$. s' is a pattern extended from s with item i, and we build its projected database $D|_{s'}$ from $D|_s$. We first search the occurrences of i in each o_s-projected sequence $d|_{o_s}$. Suppose an event e contains one of the occurrences, then it corresponds to a *mpo* $o_{s'}$ of s' in d. And we can build an $o_{s'}$-projected sequence by these two operations: (1) remove any items which is alphabetically smaller than or equal to i in e; (2) remove any event e_k with $\max(T_k) < \min(T) + g_l$, where T_k is the event time of e_k and T is the time of event e.

```
Function project($i$, $D|_s$)
  Input: $D|_s$: an uncertain projected database;
         $i$: an item, $g_l$: minimal gap-constraint
  Output: $D|_{s'}$
  foreach $d|_s \in D|_s$ do
      $d|_{s'} \leftarrow \phi$
      foreach $d|_{o_s} \in d|_s$ do
          foreach $e \in d|_{o_s} \wedge i \in e$        // search the occurrences of item $i$
          do
              build $d|_{o_{s'}}$
              compute $P_t(\cdot|o_{s'})$ from $P_t(\cdot|o_s)$ and $e.T$ and save it in $d|_{o_{s'}}$
              add $d|_{o_{s'}}$ to $d|_{s'}$
          end
      end
  end
  return $d|_{s'}$
```

Figure 2.5 shows an example of projecting an uncertain sequence. Here we set $g_l = 1$ and $g_h = 5$. The uncertain sequence d is shown in Figure 2.5(a). In Figure 2.5(b), it illustrates the $\langle a \rangle$-projected uncertain sequence of d. $o_{11} = \langle e_1 \rangle$ and $o_{12} = \langle e_3 \rangle$ are two *mpo*s of pattern $\langle a \rangle$; $d|_{o_{11}}$ and $d|_{o_{12}}$ are the two projected sequences w.r.t. o_{11} and o_{12}. Notice that the event e_1 in $d|_{o_2}$ can be eliminated because $P(T_3 \geq T_1 + 1) = 0$, referring to Lemma 2.5.

2.4.5 USPM Algorithm

We can avoid generating candidate patterns that are not likely to be p-FSPs, according to the Lemma below.



(a) d

(b) $d|_{\langle a \rangle}$

(c) $d|_{\langle ab \rangle}$

Figure 2.5. An example of projecting an uncertain sequence

Lemma 2.6 *Let $D|_s$ be the projected database of a pattern s. If an item i is not probabilistic frequent in $D|_s$, either $s' = s \cup \{i\}$ or $s' = s + \{i\}$ is not a p-FSP.*

Proof Suppose w is a possible of $D|_s$, then for each sequence $\hat{d} \in w$, $s' \preceq \hat{d}$ implies that $\langle i \rangle \preceq d$. Thus, s' is frequent in w implies that i is also frequent in w. Since $P(sup(s') \geq \tau_s) \leq P(sup(i) \geq \tau_s) < \tau_p$, this proves that s' is not a p-FSP. ■

We set $s = \phi$ and $D|_s = D$ initially. Recall from the framework shown in Algorithm 1, we can mine a set L of all p-FSPs in the following steps:

STEP 1: Find a set of all probabilistic frequent items $I = \{i_1, i_2, ..., i_n\}$ in $D|_s$. This process is fast because it does not involve temporal uncertainty management.

STEP 2: For a frequent item $i \in I$, generate a candidate pattern s' from s and i. Note that s' is item-extend such as $s' = s \cup \{i\}$, if item i is represented by $\_i$ in $D|_s$; otherwise, we sequence-extend s to generate $s' = s + \{i\}$.

STEP 3: Call function project(i, $D|_s$) to build the s'-projected database $D|_{s'}$.

STEP 4: Call function freqProb($D|_{s'}$) to compute $P(sup(s') \geq \tau_s)$.

STEP 5: If s' is not a p-FSP, remove i from I and go to step (2); otherwise, save s to L and call USPM(s', $D|_{s'}$) recursively to search all frequent patterns in $D|_{s'}$.

2.5 Management of Temporal Uncertainty

In this section, we discuss the computation of the probability $P(s \preceq o_s)$. Let $o_s = \langle e_{k_1}, \ldots, e_{k_n} \rangle$ be a *mpo* of a pattern s in an uncertain sequence d, where $e_{k_i} \in d$ is an uncertain event. Here we represent the probability of o_s satisfying gap-constrains by $P_t(o_s)$. Therefore, we have $P(s \preceq o_s) = P_t(o_s)$ by definition.

We set gap-constraints as $g_l = l$ and $g_h = h$. Let T_{k_i} be the uncertain time of event e_{k_i}. Then, a brute force approach of computing $P_t(o_s)$ is to directly decompose it by the *chain rule*, as shown in Equation (2.11) and (2.12).

$$\begin{aligned} P_t(o_s) &= \underset{l \le t_i - t_{i-1} \le h}{\int \cdots \int} f(T_{k_1} = t_1, \ldots, T_{k_n} = t_n) \mathrm{d}t_1 \cdots \mathrm{d}t_n \\ &= \underset{l \le t_i - t_{i-1} \le h}{\int \cdots \int} f(t_n | t_1 \cdots t_{n-1}) * \cdots * f(t_2|t_1) f(t_1) \mathrm{d}t_1 \cdots \mathrm{d}t_n \end{aligned} \tag{2.11}$$

$$\begin{aligned} P_t(o_s) &= \sum_{l \le t_{i+1} - t_i \le h} P(T_{k_1} = t_1, \ldots, T_{k_n} = t_n) \\ &= \underset{l \le t_{i+1} - t_i \le h}{\sum \cdots \sum} P(t_n | t_1, t_2, \ldots, t_{n-1}) * \cdots * P(t_2|t_1) * P(t_1) \end{aligned} \tag{2.12}$$

However, the complexity of this approach is exponential to the number of uncertain timestamps, so it is usually too complex to be used in practice. In this section, we propose a recursive approach to compute $P_t(o_s)$ efficiently.

2.5.1 Analysis of Base Cases

In this section, we discuss the computation of the probability that two uniformly distributed uncertain timestamps satisfy gap-constraints such as $g_l = l$ and $g_h = h$. Given $X \sim U(x^-, x^+)$ and $Y \sim U(y^-, y^+)$, the probability that X and Y satisfy gap constraints (e.g. $l \leq Y - X \leq h$) is denoted by $P(\langle XY \rangle)$. And $P(\langle XY \rangle)$ can be computed by:

$$
\begin{aligned}
P(\langle XY \rangle) &= \int \int_{l \leq Y-X \leq h} \frac{1}{(x^+ - x^-)(y^+ - y^-)} \mathrm{d}x\mathrm{d}y \\
&= \int_{\max(y^-, x^- + l)}^{\min(y^+, x^+ + h)} \int_{\max(x^-, y-h)}^{\min(x^+, y-l)} \frac{1}{(x^+ - x^-)(y^+ - y^-)} \mathrm{d}x\mathrm{d}y \\
&= \frac{1}{S} * \int_{\max(y^-, x^- + l)}^{\min(y^+, x^+ + h)} \min(x^+, y-l) - \max(x^-, y-h) \mathrm{d}y
\end{aligned}
\tag{2.13}
$$

where $S = (x^+ - x^-)(y^+ - y^-)$ is a constant. Let $f(y) = min(x^+, y-l) - max(x^-, y-h)$, then the value of $f(y)$ is interpreted into the following four deterministic cases:

$$
f(y) = \begin{cases}
x^+ + h - y & , \ \max(a_2, a_3) \leq y \leq a_4 \\
x^+ - x^- & , \ a_3 \leq y \leq a_2 \\
h - l & , \ a_2 \leq y \leq a_3 \\
y - l - x^- & , \ a_1 \leq y \leq \min(a_2, a_3)
\end{cases}
\tag{2.14}
$$

where

$$
\begin{aligned}
a_1 &= \min(y^+, \max(y^-, x^- + l)), \qquad & a_2 &= \min(y^+, \max(y^-, x^- + h)) \\
a_3 &= \min(y^+, \max(y^-, x^+ + l)), \qquad & a_4 &= \min(y^+, \max(y^-, x^+ + h))
\end{aligned}
$$

Let $b_1 = a_1$, $b_2 = \min(a_2, a_3)$, $b_3 = \max(a_2, a_3)$ and $b_4 = a_4$, then we have $b_1 \leq b_2 \leq b_3 \leq b_4$. Equation (2.13) can be written as the sum of integrations in three disjoint sub-partitions $[b_1, b_2]$, $[b_2, b_3]$ and $[b_3, b_4]$, as shown in Equation (2.15).

$$
\begin{aligned}
P(\langle XY \rangle) &= \sum_{k=1}^{3} P(\langle XY \rangle | y \in [b_k, b_{k+1}]) * P(y \in [b_k, b_{k+1}]) \\
&= P(\langle XY_k \rangle) * P(Y_k)
\end{aligned}
\tag{2.15}
$$

where we denote $Y_k = \{Y | y \in [b_k, b_{k+1}]\}$, and $P(Y_k)$ is computed:

$$
P(Y_k) = \int_{b_k}^{b_{k+1}} \frac{1}{y^+ - y^-} \mathrm{d}y = \frac{b_{k+1} - b_k}{y^+ - y^-} \tag{2.16}
$$

Since $P(\langle XY_k \rangle)$ corresponds to a deterministic case, it can be computed by Equation (2.17).

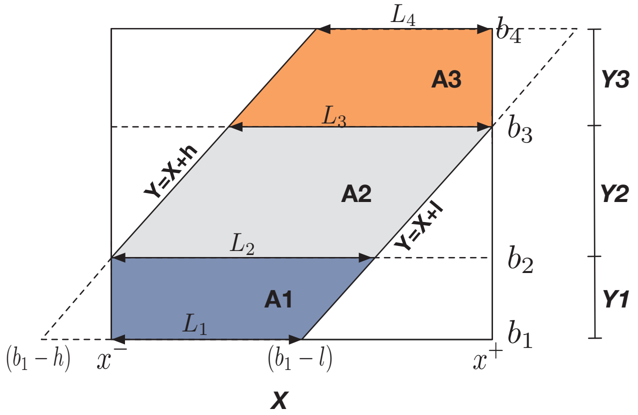


Figure 2.6. An example of computing the probability of satisfying gap constraints

$$
\begin{aligned}
P_t(\langle XY_1\rangle) &= \frac{A_1}{S_1} = \frac{\int_{a_1}^{\min(a_2,a_3)}(y-l-x^-)\mathrm{d}y}{S_1} \\
P_t(\langle XY_2\rangle) &= \frac{A_2}{S_2} = \begin{cases} \frac{1}{S_2}\int_{a_3}^{a_2}(x^+ - x^-)\mathrm{d}y & \text{, if } a_2 \geq a_3 \\ \frac{1}{S_2}\int_{a_2}^{a_3}(h-l)\mathrm{d}y & \text{, if } a_2 < a_3 \end{cases} \\
P_t(\langle XY_3\rangle) &= \frac{A_3}{S_3} = \frac{\int_{\max(a_2,a_3)}^{a_4}(x^+ + h - y)\mathrm{d}y}{S_3}
\end{aligned}
\tag{2.17}
$$

In order to derive a general formula, we use a geographic approach to compute $P(\langle XY_k\rangle)$. Figure 2.6 shows an example of the geographic representations. Here, the gap-constraints $g_l = l$ and $g_h = h$ corresponds to two straight lines $Y = X + l$ and $Y = X + h$. S_k is the area of the rectangle within $X \in [x^-, x^+]$ and $Y \in [b_k, b_{k+1}]$; A_k is the area of a trapezoid between two boundary lines within S_k. Thereafter, we can compute the probability $P(\langle XY_k\rangle)$ by Equation (2.18).

$$
\begin{aligned}
P(\langle XY_k\rangle) &= \frac{A_k}{S_k} = \frac{\int_{b_k}^{b_{k+1}} f(y)\mathrm{d}y}{S_k} = \frac{(1/2)*(L_{k+1}+L_k)*(b_{k+1}-b_k)}{(b_{k+1}-b_k)*(x^+ - x^-)} \\
&= \frac{L_{k+1}+L_k}{2*(x^+ - x^-)}
\end{aligned}
\tag{2.18}
$$

Here L_k are base lines lengths of the trapezoids. As L_k is the length of the line which is between two endpoints $(b_k - l, b_k)$ $(b_k - h, b_k)$ and satisfies the constraints, it can be computed by:

$$L_k = \min(b_k - l, x^+) - \max(b_k - h, x^-).$$

For example, to compute A_1 in Figure 2.6, we first compute $L_1 = (b_1 - l) - x^-$ and $L_2 = (b_2 - l) - (b_2 - h) = (l - h)$. Then A_1 is equal to $(L_1 + L_2) * (b_2 - b_1)/2$. And we can compute $P(\langle XY_1 \rangle = A_1/S_1$, where $S_1 = (x^+ - x^-) * (b_2 - b_1)$. After we use the same approach to compute $P(\langle XY_2 \rangle$ and $P(\langle XY_3 \rangle$, the probability of $P(\langle XY \rangle)$ can be computed by Equation (2.15).

2.5.2 Recursion for Pdf-Modeled Uncertainty Management

Given n pdf-modeled uncertain timestamps $T_i \sim U[t_i^-, t_i^+]$, We propose a recursive approach to compute the probability that $T_1, \ldots, T_n$ satisfy gap-constraints, denoted by $P(\langle T_1, \ldots, T_n \rangle)$.

Definition 2.5.1 *Suppose* $[a, b] \subseteq [t_n^-, t_n^+]$ *and* $T \sim U[a, b]$, *then we define* $P_n(T)$ *to be the probability that* $T_n = T$ *and* $T_1, \ldots, T_{n-1}, T_n$ *satisfies gap-constraints, denoted as* $P_n(T) = P(\langle T_1, \ldots, T_{n-1}, T_n \rangle \wedge T_n = T)$.

Suppose the range of T_n is divided into p disjoint sub-partitions as $[t_n^-, t_n^+] = \cup_{i=1}^{p} [t_n^i, t_n^{i+1}]$ and $T_n^i \sim U[t_n^i, t_n^{i+1}]$, then $P(\langle T_1 \cdots T_n \rangle)$ can be computed by the sum of $P_n(T_n^i)$, as shown in Equation(2.19) , according to the *the law of total probability.*

$$P(\langle T_1, \ldots, T_n \rangle) = \sum_{i=1}^{p} P(\langle T_1, \ldots, T_n^i \rangle) * P(T_n = T_n^i) = \sum_{i=1}^{p} P_n(T_n^i) \tag{2.19}$$

where we denote $P(\langle T_1, \ldots, T_n \rangle \,|T_n = T_n^i)$ by $P(\langle T_1, \ldots, T_n^i \rangle)$, for the purpose of simpilicity.

In order to derive all deterministic cases between T_{n-1} and T_n, we use sub-partitions of T_{n-1} to divide the range of T_n. Suppose the range of T_{n-1} consists

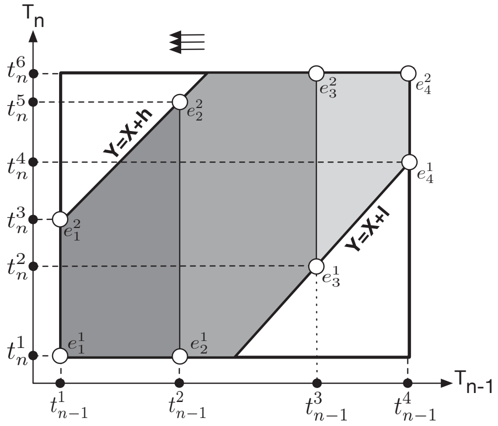


Figure 2.7. An example of dividing the range of T_n

of q disjoint sub-partitions such as $[t_{n-1}^-, t_{n-1}^+] = \cup_{j=1}^q [t_{n-1}^j, t_{n-1}^{j+1}]$, then each endpoint $x = t_{n-1}^j$ can generate two boundary points:

$$e_j^1 = (x, \min(t_n^+, \max(x+l, t_n^-))), e_j^2 = (x, \min(t_n^+, \max(x+h, t_n^-)))$$

Figure 2.7 shows an example of dividing the range of T_n based on the three disjoint sub-partitions of T_{n-1}. Here $\{e_1^1, e_1^2, \cdots, e_4^1, e_4^2\}$ are eight boundary points derived from t_{n-1}^j (for $j \in [1,4]$). The T_n-axis projections of these boundary points are $t_n^1 < \cdots < t_n^6$, which divide the range of T_n into five sub-partitions.

In general, we assume that $\{t_n^i | i \in [1, p+1]\}$ are an ordered set of endpoints in T_n, then the range of T_n is divided into p sub-partitions such as $[t_n^-, t_n^+] = \cup_{i=1}^p [t_n^i, t_n^{i+1}]$. Therefore, we can compute $P(\langle T_{n-1}^j T_n^i \rangle)$ by Equation (2.18), referring to Lemma 2.7.

Lemma 2.7 *Given $T_{n-1}^j \sim U[t_{n-1}^j, t_{n-1}^{j+1}]$ and $T_n^i \sim U[t_n^i, t_n^{i+1}]$, $\langle T_{n-1}^j T_n^i \rangle$ corresponds to a deterministic case.*

Proof Suppose the range of T_n is split into three sub-partitions such as $[t_n^-, t_n^+] = \cup_{i=1}^3 [b_i, b_{i+1}]$. As t_n^i and t_n^{i+1} are two consecutive points that further split $[b_k, b_{k+1}]$, we have $b_k \leq t_n^i < t_n^{i+1} \leq b_{k+1}$. Thus, $\langle T_{n-1}^j T_n^i \rangle$ is a deterministic case because $[t_n^i, t_n^{i+1}] \subseteq [b_2, b_3]$. ■

Here we design a divide-and-conquer algorithm to compute $P_n(T_n^i)$. The computation of $P_n(T_n^i)$ is divided into sub-problems of computing the probabilities that $T_n = T_n^i$, $T_{n-1} = T_{n-1}^j$ and $T_1, \cdots, T_n$ satisfy gap-constraints, for $j = 1, \ldots, q$. These probabilities are then combined together by the law of total probability, as shown in Equation (2.20).

$$P_n(T_n^i) = \sum_j P(\langle T_1, \ldots, T_{n-1}^j, T_n^i \rangle) * P(T_{n-1} = T_{n-1}^j) * P(T_n = T_n^i) \tag{2.20}$$

Since the gap-constraints only regulate relationships between adjacent timestamps, we have:

$$P(\langle T_1, \cdots, T_{n-1}^j, T_n^i \rangle) = P(\langle T_1, \ldots, T_{n-1}^j \rangle) * P(\langle T_{n-1}^j, T_n^i \rangle)$$

And then, Equation (2.20) can be written as a recursive function:

$$\begin{aligned} P_n(T_n^i) &= \sum_j P(\langle T_1, \ldots, T_{n-1}^j \rangle) * P(\langle T_{n-1}^j, T_n^i \rangle) * P(T_{n-1}^j) * P(T_n^i) \\ &= \sum_j P_{n-1}(T_{n-1}^j) * P(\langle T_{n-1}^j, T_n^i \rangle) * P(T_n^i) \end{aligned} \tag{2.21}$$

Algorithm 2.2 summarizes the computation of $P_n(T_n^i)$ for n uniformly distributed uncertain timestamps $T_1, \ldots, T_n$. In the base case of $n = 2$, we compute $P_2(T_2^j)$ from T_1 and T_2 by Equation (2.16) and (2.18). If $n > 2$, it first computes $P_{n-1}(T_{n-1}^j)$ recursively, for $j \in [1, p]$. Suppose the range of T_n is divided into q sub-partitions by the splits of T_{n-1}. Let T_n^i be a random variable associated with i^{th} sub-partition of T_n, then $P(\langle T_{n-1}^j T_n^i \rangle)$ can be computed in constant time. Thereafter, Equation (2.21) is used to compute $P_n(T_n^i)$. Algorithm 2.2 returns a set of values of $P_n(T_n^i)$ (for $i = 1, \ldots, q$), from which we can compute $P(\langle T_1, \ldots, T_n \rangle) = \sum_i P_n(T_n^i)$.

2.5.3 Recursion for Pmf-Modeled Uncertainty Management

We denote the *pmf* of a discrete uncertain timestamp T_i by $f_{T_i} = P(T = t_i^k)$, where t_i^k is one of the possible values of T_i. Here we first define $P_n(t)$ for pmf-modeled uncertain timestamps.

Algorithm 2.2: UTimeProb

```
Input: $\{T_i | T_i \sim U[t_i^-, t_i^+], i \in [1, n]\}$
Output: $\{P_n(T_n^i) | i \in [1, p]\}$
if $n = 2$ then
    compute $P_2(T_2^j)$ by Equation (2.16) and (2.18)
    return $\{P_2(T_2^j) | j \in [1, 3]\}$
end
$\{P_{n-1}(T_{n-1}^j) | j \in [1, q]\} = \text{UTimeProb}(T_1, \ldots, T_{n-1})$
divide $[t_n^-, t_n^+]$ into $p$ disjoint sub-partitions, obtain $T_n^i \sim U[t_n^i, t_n^{i+1}] \subseteq [t_n^-, t_n^+]$, $\forall i \in [1, p]$
for $i \leftarrow 1 : p$ do
    $P(T_n = T_n^i) \leftarrow (t_n^{i+1} - t_n^i)/(t_n^+ - t_n^-)$
    $P_n(T_n^i) \leftarrow 0$
    for $j \leftarrow 1 : q$ do
        $P_n(T_n^i) \leftarrow P_n(T_n^i) + P_{n-1}(T_{n-1}^j) * P(\langle T_{n-1}^j T_n^i \rangle) * P(T_n = T_n^i)$
    end
end
return $\{P_n(T_n^i) | i \in [1, p]\}$
```

Definition 2.5.2 *Let t be a possible value of T_n, then $P_n(t)$ is defined to be the probability that $T_n = t$ and $T_1, \ldots, T_{n-1}, T_n$ satisfies gap-constraints, such as $P_n(t) = P(\langle T_1, \ldots, T_{n-1}, T_n \rangle \wedge T_n = t)$.*

Suppose T_n has p possible values such as $t_n^1, \cdots, t_n^p$, then $P(\langle T_1, \ldots, T_n \rangle)$ is the sum of $P_n(t_n^i)$ ($\forall i \in [1, p]$), as shown in Equation (2.22), according to the law of total probability.

$$\begin{aligned} P(\langle T_1, \ldots, T_n \rangle) &= \sum_i P(\langle T_1, \ldots, T_n \rangle | T_n = t_n^i) * P(T_n = t_n^i) \\ &= \sum_i P(\langle T_1, \ldots, T_n \rangle \wedge T_n = t_n^i) = \sum_i P_n(t_n^i) \end{aligned} \tag{2.22}$$

We first compute the probability $P(\langle T_1 T_2 \rangle)$ that two uncertain timestamps T_1 and T_2 satisfy gap-constraints. Suppose $f_{T_1}(t_1^j) = P(T_1 = t_1^j)$ and $f_{T_2}(t_2^i) = P(T_2 = t_2^i)$, for $j \in [1, p]$ and $i \in [1, q]$. Then, $P(\langle T_1 T_2 \rangle$ can be computed from the joint distribution of (X, Y), as shown in Equation (2.23).

$$P(\langle T_1 T_2 \rangle) = \sum_i P_2(t_2^i) = \sum_i \sum_j \delta(t_1^j, t_2^i) * P(T_1 = t_1^j) * P(T_2 = t_2^i) \tag{2.23}$$

where the value of $\delta(x_j, y_i)$ depends on if x_j and y_i satisfy gap-constraints or not. Let $g_l = l$ and $g_h = h$, then $\delta(x_j, y_i)$ is computed by Equation (2.24).

$$\delta(x_j, y_i) = P(\langle x_j, y_i \rangle) = \begin{cases} 1 & \text{, if } l \leq y_i - x_j \leq h \\ 0 & \text{, otherwise} \end{cases} \tag{2.24}$$

The key idea of recursive approach is to first split the computation of $P_n(t_n^i)$ into sub-problems of computing $P_{n-1}(t_n^j)$, and then combine them by the law of total probability, as shown in Equation (2.25).

$$P_n(t_n^i) = \sum_{j=1}^{q} P(\langle T_1, \ldots, T_{n-1}^j, T_n^i \rangle) * P(T_{n-1} = t_{n-1}^j) * P(T_n = t_n^i) \tag{2.25}$$

Since gap-constraints only apply to adjacent timestamps, $T_1, \ldots, T_n$ satisfy gap-constrains, if both $\langle T_1, \ldots, T_{n-1} \rangle$ and $\langle T_{n-1} T_n \rangle$ satisfy constraints. Therefore, Equation (2.25) can be written as the following recursive function:

$$\begin{aligned} P_n(t_n^i) &= \sum_{j=1}^{q} P(\langle t_{n-1}^j, t_n^i \rangle) * P(\langle T_1, \ldots, T_{n-1} \rangle) * P(t_{n-1}^j) * P(t_n^i) \\ &= \sum_{j=1}^{q} \delta(t_{n-1}^j, t_n^i) * P_{n-1}(t_{n-1}^j) * P(t_n^i) \end{aligned} \tag{2.26}$$

Algorithm 2.3 summarizes the computation of the probability that n pmf-modeled uncertain times $T_1, \ldots, T_n$ satisfy gap-constraints. In the base case of $n = 2$, it is straightforward to compute the values of $P_2(t_n^i)$ from the pmf of T_1 and T_n by Equation (2.23). If $n > 2$, we first call a recursive function to compute the values of $P_{n-1}(t_{n_1}^j)$ $(j = 1, \ldots, q)$, then we combine the output of these sub-problems using Equation (2.26). Algorithm 2.3 returns a set of values of $P_n(t_n^i)$, from which we can

Algorithm 2.3: DTimeProb

```
Input: ${T_i | i \in [1, n]}$, $f_{T_i} = P(T_i = t_i^k)$
Output: ${P_n(t_n^i) | i \in [1, p]}$
if $n = 2$ then
    compute $P_2(t_n^i)$ by Equation (2.23)
end
${P_{n-1}(t_{n-1}^j) | j \in [1, q]}$ = DTimeProb($T_1, \ldots, T_{n-1}$)
for $i \leftarrow 1 : p$ do
    $P_n(t_n^i) \leftarrow 0$
    for $j \leftarrow 1 : q$ do
        $P_n(t_n^i)$ += $P_{n-1}(t_{n-1}^j) * \delta(t_{n-1}^j, t_n^i) * P(T_n = t_n^i)$
    end
end
return ${P_n(t_n^i) | i \in [1, p]}$
```

compute $P(\langle T_1, \ldots, T_n \rangle = \sum_i P_n(t_n^i)$. The complexity of Algorithm 2.3 is $O(n * m)$, where m is the average number of possible values for an uncertain timestamp.

2.5.4 Integration with SPM Framework

Now we integrates our temporal uncertainty management into sequence project process, which is shown in Function project. The computational complexity of Algorithm 2.2 and 2.3 can be reduced if we save and reuse the values of $P_{n-1}(T_{n-1}^i)$ or $P_{n-1}(t_{n-1}^i)$. For simplicity of notation, we define $P_t(\cdot|o_s)$ to be a general representation of the values of either $P_{n-1}(T_{n-1}^i)$ or $P_{n-1}(t_{n-1}^i)$ as follows:

Definition 2.5.3 *If temporal uncertainty is modeled by uniform probability density function, we have* $P_t(\cdot|o_s) = \{P_n(T_n^1), \ldots, P_n(T_n^p)\}$*; while if temporal uncertainty is pmf-modeled, we have* $P_t(\cdot|o_s) = \{P_n(t_n^1), \ldots, P_n(t_n^p)\}$ *instead.*

The value of $P_t(\cdot|o_s)$ is saved and associated with o_s-projected sequence $d|_{o_s}$. Here we reuse it to avoid redundant computation in mining sequential patterns. Let i be a item in $d|_{o_s}$ and s' is a pattern extend from s with i. If $s' = s \cup \{i\}$, we have $P_t(\cdot|o_{s'}) = P_t(\cdot|o_s)$ because $o_{s'} = o_s$; if $s' = s + \{i\}$, we can compute the value of $P_t(\cdot|o_{s'})$ directly from $P_t(\cdot|o_s)$ and T without calling the recursive functions.

In the running example of Figure 2.5, o_{12} is a *mpo* of $s = \langle a \rangle$ in d and o_{22} is a *mpo* of $s' = \langle ab \rangle$. $d|_{o_{22}}$ is projected from $d|_{o_{12}}$, since the event e_2 in $d|_{o_{12}}$ contains an occurrence of item b. Given $g_l = 1$ and $g_h = 5$, we we can compute $P_t(\cdot|o_{22}) = \{8 : 0.3\}$ from the saved value $P_t(\cdot|o_{12}) = \{4 : 0.5, 9 : 0.5\}$ and $T_2 = \{3 : 0.4, 8 : 0.6\}$ in $d|_{o_{12}}$ by Equation (2.26).

2.5.5 Pruning

We discuss pruning techniques used in computing frequentness probabilities in this section. Suppose $X_i \sim \mathcal{B}(1, p_i)$ are n Bernoulli random variables, where $p_i = P(s \preceq d_i) > 0$ is the non-zero support probability of pattern s in an uncertain sequence d_i. Let $S = sup(s)$ here, then the overall support S is the sum of X_i such as $S = X_1 + \cdots + X_n$. And the expected value of S is denoted by $\mathrm{E}(S)$. Lemma 2.8 and 2.9 below describe how to prune s without computing its frequentness probability.

Lemma 2.8 (Count Pruning) *Suppose there are* n *uncertain sequences which have* $P(s \preceq d) > 0$ *then* s *can be pruned, if* $n < \tau_s$.

Proof Because $S = X_1 + \cdots + X_n$ and $X \in \{0, 1\}$, we have $S \leq n$. If $n < \tau_s$, then $S < \tau_s$ and s is not likely to be a p-FSP. ■

The count pruning technique has also been used in mining uncertain frequent itemsets in [19].

Lemma 2.9 (Hoeffding's inequality pruning) *Let* $t = \sqrt{\frac{n * \ln(1/\tau_p)}{2}}$, *then* s *can be pruned if* $\mathrm{E}(S) + t < \tau_s$.

Proof Since $X_1, \ldots, X_n$ are independent random variables bounded by the interval $[0, 1]$, $S = X_1 + \cdots + X_n$ satisfies *Hoeffding's inequality*: $P(S - \mathrm{E}[S] \geq t) \leq \exp(-\frac{2t^2}{n})$. Here let $t = \sqrt{\frac{n*\ln(1/\tau_p)}{2}}$, then we have $P(S \geq \mathrm{E}(S) + t) \leq \tau_p$. if $\mathrm{E}(S) + t < \tau_s$, $P(sup(s) \geq \tau_s) < P(\mathrm{E}(S) + t) \leq \tau_p$. ■

Beside pruning in computing frequentness probabilities, Lemma 2.10 describes a pruning technique based on gap-constraints (e.g. $g_l = l$ and $g_h = h$) to speed up uncertainty management.

Lemma 2.10 (Gap pruning) *Suppose* $a_1 \leq T_1 \leq b_1$ *and* $a_2 \leq T_2 \leq b_2$, *then* $P(\langle T_1, T_2 \rangle) = 0$ *if* $b_2 < a_1 + l$ *or* $a_2 > b_1 + h$.

Proof The possible range of T, which satisfies $g_l \leq T - T_1 \leq g_h$, is $[a_1 + l, b_1 + h]$. Given $a_2 \leq T_2 \leq b_2$, we have $P(\langle T_1 T_2 \rangle) = 0$, if $b_2 < a_1 + l$ or $a_2 > b_1 + h$. ■

We apply gap-pruning in two places: (1) it helps to compute the base case of two two uncertain timestamps satisfying constraints, as in Equation either (2.15) or (2.23); (2) it speeds up the computation of $P_t(\cdot|o_{s'})$ in the *project* function. In an uncertain sequence d, suppose the minimal and maximal timestamps of $P_t(\cdot|o_s)$ are a_1 and b_1. Let $a_2 \leq T \leq b_2$ be the time of uncertain event e that contains an occurrence of item i. Therefore, if $b_2 < a_1 + l$ or $a_2 > b_1 + h$, we do not need to compute the values of $P_t(\cdot|o_{s'})$ because we have $P(\cdot|o_{s'}) = 0$. Here $s' = s + \{i\}$ is a sequence-extension of s with item i.

2.6 Evaluation

We employ the IBM market-basket data generator [46] to generate synthetic datasets using the following parameters:

C: number of sequences;

T: average number of transactions/itemsets per data-sequence;

L: average number of items per transaction/itemset per data-sequence;

I: number of different items.

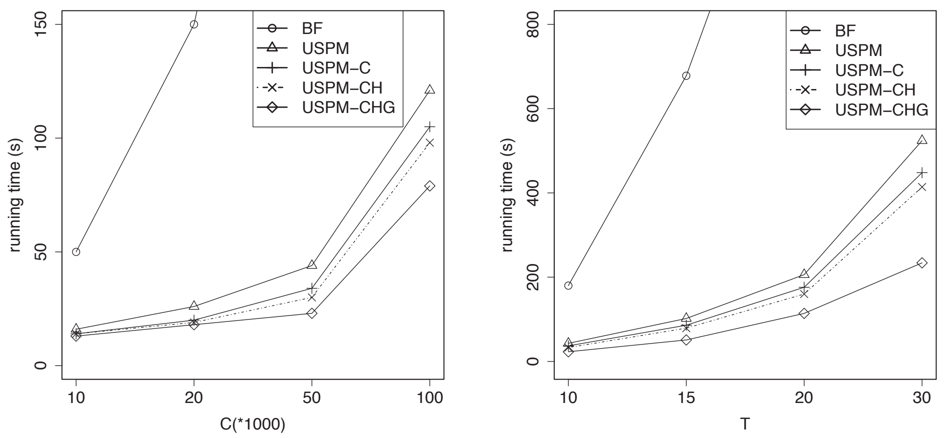
To add uniform pdf modeled uncertainty, we replace each point-value timestamp t by a uniformly distributed random variable $T \sim [(1-r)*t, (1+r)*t]$, where r is randomly drawn from $(0, 1)$. We name the generated synthetic dataset by parameters. E.g., the dataset named $T4L10I1C10$ indicates that $T = 4$, $L = 10$, $I = 1 * 1000$ and $C = 10 * 1000$.

Recall from Section 2.5 that the brute force method to compute the probability of an occurrence satisfying gap constraints is to compute Equation (2.11) or Equation (2.12) using chain rule. This method is implemented and abbreviated as *bf*. And our USPM algorithm is abbreviated as *uspm*. In order to evaluate the effect of our pruning techniques, the approximate algorithm with count pruning techniques is named *uspm-c*; we apply both count pruning and Hoeffding's inequality pruning in our algorithm and name it by *usm-ch*; and the algorithm with all three pruning techniques is so called *uspm-chg*. All the experiments were done in a desktop with Intel(R) Core (TM) Duo CPU @ 2.33GHz and 4GB memory.

2.6.1 Scalability Evaluation

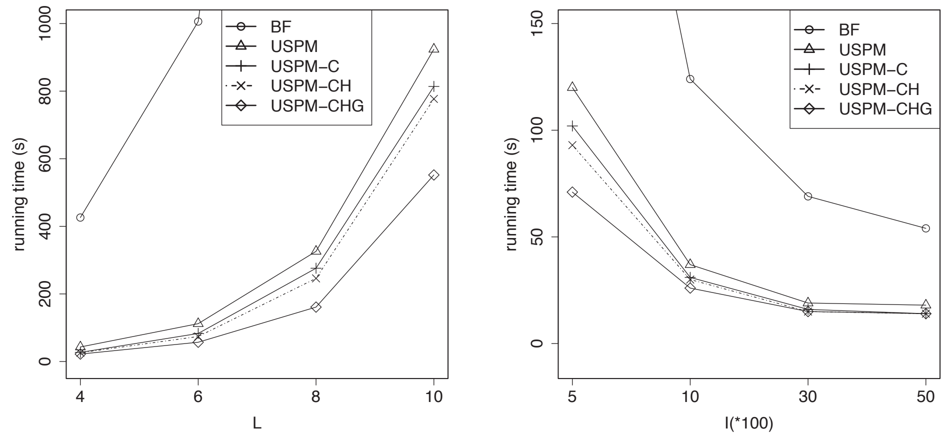
In Figure 2.8, we compare the running time of *BF*, *USPM*, *USPM-C*, *USPM-CH* and *USPM-CHG* on synthetic datasets with uniformly distributed temporal uncertainty. We set $\tau_s = 0.5\%$, $\tau_p = 0.7$, $g_l = 1$, and $g_h = 10$. And $C = 10\,000$, $T = 4$, $I = 10\,000$ and $L = 2$ are the default parameters used to generate datasets. Thereafter, we vary one of the parameters to test the performance in different scales. For example, in Figure 2.8(a), C varies from 10 000 to 100 000; in Figure 2.8(b), T varies from 10 to 30; in Figure 2.8(c), L varies from 4 to 10; and in Figure 2.8(c) I varies from 500 to 50 000. Figure 2.8 shows the following phenomena:

(1) The running time of both *BF* and *USPM* increases with the increment of C, T, L, as the increment of these parameters generates larger scale datasets.



(a) scale of C

(b) scale of T



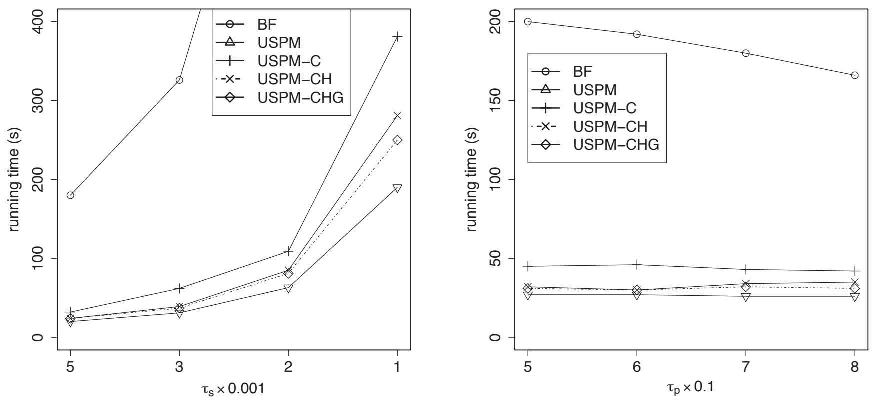
(c) scale of L

(d) scale of I

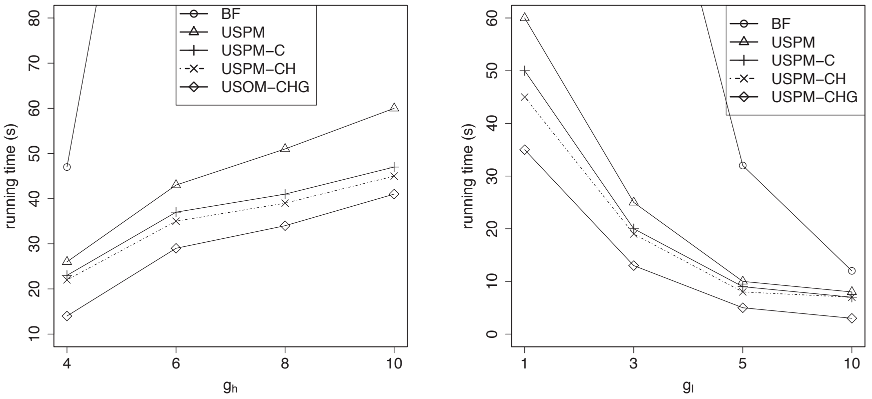
Figure 2.8. Scalability comparison in synthetic uncertain datasets

(2) The running time decreases slightly with the increment of I. When I is set to a larger value, the number of *mpo*s of a pattern in a sequence is fewer because there are fewer repeated items in a sequence.

(3) Our algorithm *USPM* is significantly faster than *BF* under every setting of parameters, which proves the effectiveness of our uncertainty computation approach.



(a) effect of τ_s (b) effect of τ_p



(c) effect of g_h (d) effect of g_l

Figure 2.9. Effect of parameters in mining synthetic uncertain datasets

(4) The *USPM-CHG* algorithm with all three pruning methods is about two times faster than the original algorithm *USPM*, which proves the overall effectiveness of our pruning techniques.

(5) Comparing the performance of *USPM-C*, *USPM-CH* and *USPM-CHG*, we can see that the effect of gap-pruning is more significant than that of other two pruning techniques. And gap-pruning becomes even more effective when T and L are larger.

Figure 2.10 compares the running time of *BF*, *USPM*, *USPM-C*, *USPM-CH* and *USPM-CHG* with different parameters in the uncertain dataset T10L2I10C10. We set the default values as $\tau_s = 0.2\%$, $\tau_p = 0.7$, $g_l = 1$, and $g_h = 10$. In Figure 2.9(a), τ_s decreases from 0.5% to 0.1%; in Figure 2.9(b), τ_p varies from 0.5 to 0.8; g_h varies from 4 to 10 in Figure 2.9(c); and g_l varies from 1 to 10 in Figure 2.9(d). And we observe the following phenomena in Figure 2.10:

(1) The running time of all five algorithms increases with the decrement of τ_s. The smaller τ_s is, the more p-FSPare found. This explains why all the algorithms take more time when τ_s is set to a smaller value.

(2) The performance of all the algorithms is relatively stable despite the variation of τ_p. According to the *Hoeffding's inequality*, the support of a sequential pattern is bounded to its expected value as $P(|S - E(S)| > t) < \delta$, where $S = sup(s)$. Therefore, the frequentness of a pattern is deterministic if its expected support is much smaller or larger than τ_s. As the supports of patterns are evenly distributed in the datasets, only a small number of patterns change from being not-frequent to being frequent when we lower the value of t_p.That is the reason why the running time of uncertain SPM algorithms does not fluctuate significantly in Figure 2.9(b).

(3) The running time of all the algorithms decreases when we decrease g_h or increase g_l. Meanwhile, gap pruning becomes more effective when g_h becomes smaller or g_l becomes larger . This is because a smaller gap $(g_h - g_l)$ indicates a more strict constraint to sequential patterns.

2.6.2 Approximation Evaluation

In our model, we use a discrete pmf to approximate any arbitrary shaped pdf of temporal uncertainty. In this section, we evaluate the effectiveness of this approximation. Let $T \sim U(t^-, t^+)$ be an uncertain timestamp modeled by a uniform distribution. To generate a pmf to approximate T, we divide $[t^-, t^+]$ into k equal-width sub-partitions, and then T can be approximated by $\{T|t_1 : p_1, \ldots, t_k : p_k\}$. Let

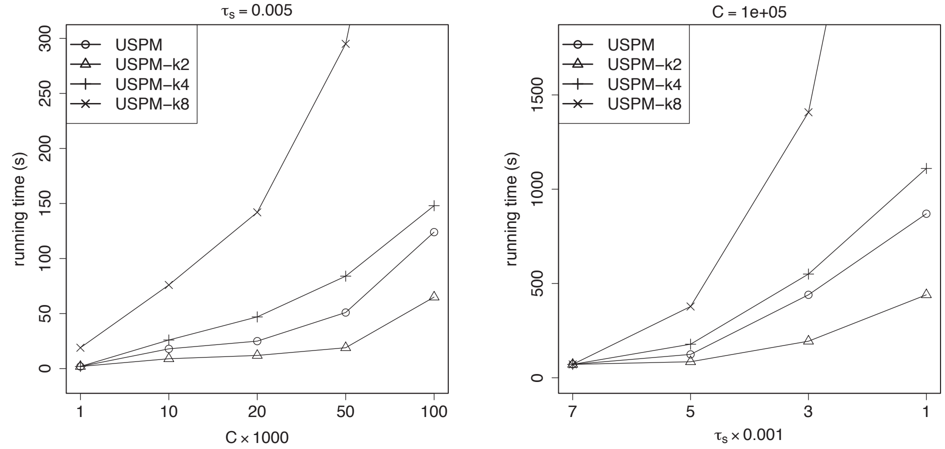
Table 2.1.
Precisions and recalls of approximation

$\tau_s(\%)$	*USPM-k*2		*USPM-k*4		*USPM-k*8	
	P	R	P	R	P	R
0.1	0.81	0.74	0.94	0.92	1.00	0.97
0.3	0.88	0.81	0.94	0.95	0.99	0.99
0.5	0.92	0.94	0.98	0.99	1	0.98
0.7	1	0.99	1	0.99	1	0.99

$[a, b]$ be the i^{th} sub-partition, then we have $t_i = (a + b)/2$ and $p_i = (b - a)/(t_r - t_l)$. Here k is a parameter to control the approximate level.

In the uncertain dataset $T10L4I10C100$, we vary the value of τ_s to test the effectiveness of approximation techniques. Table 2.1 shows the *Precision(P)* and *Recall(R)* of approximate USPM algorithms, where *USPM-kn* represents the approximate algorithm with $k = n$. The pmf approximation performs well, as we can see that both the *Precision* and *Recall* of *USPM-kn* are close to 1 when $k = 8$. Note that when $\tau_s = 0.7\%$, there are only 1-length p-FSPs mined in all algorithms. Therefore, the value of k does not affect the outputs and that is why all *uspm-kn* have the same performance in this case.

Figure 2.10 compares the running time of our approximate USPM algorithms with different values of k. In Figure 2.10(a), we set $\tau_s = 0.5\%$ and then compare the efficiency of *uspm*, *uspm-k*2, *uspm-k*4 and *uspm-k*8 under different C scales; in Figure 2.10(b), we set $C = 100\,000$ and vary τ_s from 0.1% to 0.7%. The trade off between accuracy and efficiency is that a larger value of k makes the approximation more accurate, while the time cost increases significantly at the same time.



(a) vary C

(b) vary τ_s

Figure 2.10. Compare the running time of approximate USPM algorithms

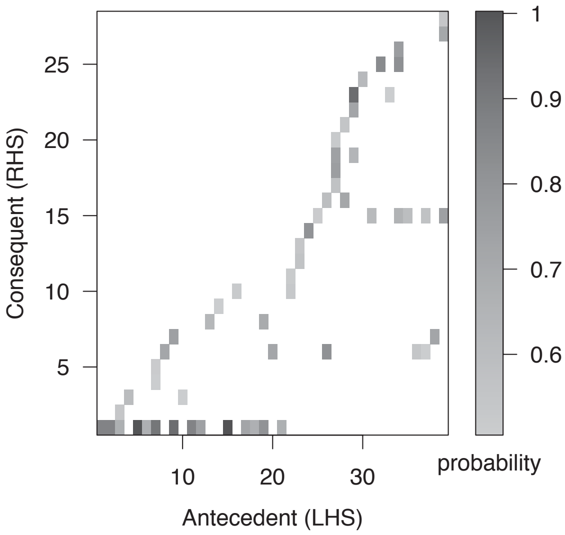
2.6.3 Mining Sequential Patterns in Stock Datasets

We also apply our USPM algorithm to a real world stock market dataset. The prices for 1025 stocks during 01-01-2008 to 03-20-2015 in Shanghai Stock Exchange Center are extracted from *Yahoo! Finance*.

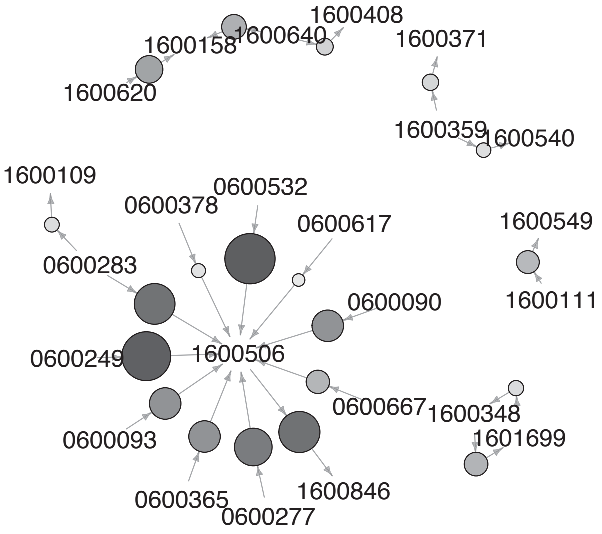
We define two types of events here. If the highest price of a stock A grows more than 8% above its opening price in one day, we denote this event as $1A$; if the lowest price of A drops more than 8%, we denote this event as $0A$, where 1 and 0 are two flags to indicate the increase or decrease of prices.

However, as the original dataset records the daily lowest and highest prices without their exact timestamps, the precise occurrence time of an event is unknown and is modeled by a uniformly distributed random variable in the indicated day. For example, if we observe the event $1A$ in 01-01-2009, then the event time of $1A$ is assumed to be uniformly distributed in the trading time period of this day.

We group the data by weeks. All events within one week forms a sequence, and this generates 373 uncertain sequences in total. We set $\tau_s = 2\%$, $\tau_p = 0.5$, $g_l = 1$



(a) matrix with 58 p-FSPs



(b) samples of 20 p-FSPs

Figure 2.11. Mined p-FSPs in the stock dataset

(*hour*) and $g_h = 48$ (*hour*). The *uspm* algorithm costs about 28 seconds to find 1490 p-FSPs, which consists of 1432 frequent items and 58 two-length p-FSPs.

Figure 2.11(a) shows the frequentness probability distribution of two-length p-FSPs using a matrix. A two-length sequential pattern can be represented by $\{A\} =>$

$\{B\}$, where $\{A\}$ is an itemset in antecedent (LHS) and $\{B\}$ is an itemset in consequent (RHS). Each unit in Figure 2.11(a) represents a p-FSP, and the darkness of the unit is determined by the frequentness probability of the pattern. We can see that there is one item in RHS which connects to many item in LHS and most of p-FSPs have moderate frequentness probability between 0.7 and 0.9.

Figure 2.11(b) uses a graph to show the connections between stocks by a sample of 20 p-FSPs. Each connection is a p-FSP and is associated with a node. The size of the node is larger and its color becomes darker if the frequentness probability of the pattern is higher. This graph helps to reveal and understand the relations between stocks.

We observe that the USPM algorithm finds some patterns which are consistent with our knowledge. E.g., $\langle(1600348)(1601699)\rangle$ is such a pattern, where 600348 and 601699 are the stock code of two coal mining companies in the same province. However, some hidden patterns are also found. For example, a p-FSP $\langle(0600365)(1600506)\rangle$ indicates that when the price of 600365, a wine-making company decreases, it is often followed by the price increase of 600506, which is a farming company.

2.7 Conclusion

In this chapter, we develop a sequential pattern mining algorithm in temporal uncertain databases. Uncertain timestamps in our model are represented by either partition-based uniform distribution functions or discrete probability mass functions. In order to mine accurate results, we design a recursive approach to efficiently compute temporal uncertainty and integrate it with the classic pattern-growth SPM algorithm for mining uncertain databases. The experimental results on both synthetic and real datasets prove that our algorithm is efficient and scalable.

3 SEQUENTIAL PATTERN MINING IN DATABASES WITH EXISTENTIAL UNCERTAINTY

3.1 Introduction

Sequential pattern mining (SPM) is an important data mining that provides inter-transactional analysis for timestamped data in sequence databases. In real applications, uncertainty is almost everywhere and it may cause probabilistic event existence in sequence databases, as shown in the following example.

Example 3.1 *In an employee tracking RFID network, the tag read by sensors are modeled by a relation see(t, aId, tId), which denotes that the RFID tag* tId *is detected by an antenna* aId *at time* t*. Since an RFID sensor can only identify a tag with a certain probability within its working range, the PEEX system [47] outputs an uncertain event such as meet(*100*, Alice, Bob,* 0.4*), which indicates that the event that Alice and Bob meet at time 100 happens with probability 0.4.*

Possible world semantics is widely used to interpret uncertain databases [15, 29]; however, it also brings efficiency and scalability challenges to uncertain SPM problems. Meanwhile, applications in the areas of biology, Internet and business informatics encounter limitations due to large scale datasets. While MapReduce is a widely used programming framework for processing big data in parallel, its basic framework can not directly be used in SPM because it does not support the iterative computing model which is required by most SPM algorithms.

In this chapter, we propose a sequential pattern mining algorithm in iterative MapReduce for large scale uncertain databases. And the main contributions are summarized as follows:

(1) We use possible world semantics to interpret uncertain sequence databases and analyze the naturally correlated possible worlds.

(2) We design a vertical format of uncertain sequence databases in which we save and reuse intermediate computational results to significantly reduces the time complexity.
(3) We design an iterative MapReduce framework to execute our uncertain algorithm in parallel.
(4) Extensive experiments are conducted in both synthetic and real uncertain datasets, which prove the efficiency and scalability of our algorithm.

3.2 Related Works

A lot of traditional database and data mining techniques have been extended to be applied to uncertain data [43]. Muzammal and Raman propose the SPM algorithm in probabilistic database using expected support to measure pattern frequentness, which has weakness in mining high quality sequential patterns [17, 48]. Zhao et al. define probabilistic frequent sequential patterns using possible world semantics and propose their complimentary uncertain SPM algorithm UPrefixSpan [15, 16]; however, it uses the depth-first strategy to search frequent patterns and cannot be directly extended to MapReduce framework. A dynamic programming approach of mining probabilistic spatial-temporal frequent sequential patterns is introduced in [40]; Wan et al. [39] propose a dynamic programming algorithm of mining frequent serial episodes within an uncertain sequence. However, dynamic programming also cannot be directly extended to MapReduce.

Jeong et al. propose a MapReduce framework for mining sequential patterns in DNA sequences with only four distinct items [49], in contrast to this paper where unlimited number of items are allowed; Chen et al. extend the classic SPAM algorithm to its MapReduce version SPAMC [50]. However, SPAMC relies on a global bitmap and it is still not scalable enough for mining extremely large databases. Miliaraki et al. propose a gap-constraint frequent sequence mining algorithm in MapReduce [51]. However, all these algorithms are applied in the context of deterministic data, while our work aims to solve large scale uncertain SPM problems.

sid	**eid**	**I**	**Pe**
1	1	{A,B}	0.8
1	2	{C}	0.8
2	1	{B}	1
2	2	{C}	0.8
2	3	{C}	0.4

Figure 3.1. An example of uncertain database

wid	**Possible world**
1	<e_{11}><e_{21}, e_{22}, e_{23}>
2	<e_{11}, e_{12}><e_{21}, e_{22}>
3	<e_{11}, e_{12}><e_{21}, e_{23}>
4	<e_{11}, e_{12}><e_{21}, e_{22}, e_{23}>
…	…

Figure 3.2. Possible worlds table

3.3 Problem Definition

3.3.1 Data Model of Existential Uncertainty

An uncertain database contains a collection of uncertain sequences. An uncertain sequence is an ordered list of uncertain events. An uncertain event is represented by $e = \langle sid, eid, I, p_e \rangle$. Here *sid* is the sequence id and *eid* is the event id. $\langle sid, eid \rangle$ identifies a unique event. I is an itemset that describes event e, and p_e is the existential probability of event e. Figure 3.1 shows an example of an uncertain sequence database. Here, for instance, the uncertain event $e_{11} = \langle 1, 1, \{AB\}, 0.8 \rangle$ indicates that the itemset $\{AB\}$ occurs in e_{11} with probability 0.8.

We use possible world semantics to interpret uncertain sequence databases. A possible world is instantiated by generating every event according to its existential probability. The number of possible worlds grows exponentially to the number of sequences and events. It is widely assumed that uncertain sequences in the sequence database are mutually independent, which is known as the *tuple-level independence* [28,43] in probabilistic databases. Events are also assumed to be independent of each other [15,29], which can be justified by the assumption that events are often observed independently in real world applications. Therefore, we can compute the existential probability of a possible world w in Equation (3.1).

$$P_e(w) = \prod_{\forall d_i \in w} \{ \prod_{\forall e_{ij} \in d_i} P(e_{ij}) * \prod_{e_{ij} \notin d_i} (1 - P(e_{ij})) \} \tag{3.1}$$

where $d_i \in w$ is a sequence in w and $e_{ij} \in d_i$ is an event in d_i. Here e_{ij} is instantiated from the original database and $P(e_{ij})$ is its existential probability. Figure 3.2 is a table which contains four possible worlds of the uncertain sequence database in Figure 3.1. Then, for example, we can compute the existential probability of possible world w_1 by $P(w_1) = (0.8 * 0.2) * (1 * 0.8 * 0.4) = 0.0512$.

3.3.2 Uncertain SPM Problem

A sequential pattern $\alpha = \langle X_1 \cdots X_n \rangle$ is *supported* by a sequence $\beta = \langle Y_1 \cdots Y_m \rangle$, denoted by $\alpha \sqsubseteq \beta$, if and only if there exists integers $1 \leq k_1 < \cdots < k_n \leq m$ so that $X_i.I \subseteq Y_{k_i}.I$, $\forall i \in [1, n]$. In deterministic databases, a sequential pattern s is frequent if and only if it satisfies $sup(s) \geq \tau_s$, where $sup(s)$ is the total number of sequences that support s and τ_s is the user-defined minimal threshold. In an uncertain database D, the frequentness of s is probabilistic and it can be computed by Equation (3.2).

$$P(sup(s) \geq \tau_s) = \sum_{\forall w, sup(s|w) \geq \tau_s} P(w) \tag{3.2}$$

Where w is a possible world in which s is frequent and $P(w)$ is the existential probability of w. The uncertain sequential pattern mining problem is defined as follows: *Given a sequence database D with existential uncertainty, a minimal support threshold τ_s and a minimal frequentness probability threshold τ_p, find every probabilistic frequent sequential pattern s in D which has $P(sup(s) \geq \tau_s) \geq \tau_p$.*

3.4 Uncertain Sequential Pattern Mining with Iterative MapReduce

3.4.1 Approximation of Frequentness Probability

Suppose $D = \{d_1, \ldots, d_n\}$ is an uncertain database and s is a sequential pattern. Because $d_1, \ldots, d_n$ in D are mutually independent, the probabilistic support of s in D, denoted by $sup(s)$, can be computed by Equation (3.3).

$$sup(s) = \sum_{i=1}^{n} sup(s|d_i) \tag{3.3}$$

Where $sup(s|d_i)$ $(i = 1, \ldots, n)$ are Bernoulli random variables, whose success probabilities are $P(sup(s|d_i) = 1) = P(s \sqsubseteq d_i)$. And we will discuss the computation of $P(s \sqsubseteq d_i)$ in section 3.4.2.

We find that $sup(s)$ is a Poisson-Binomial random variable, because it is the sum of n independent but non-identical Bernoulli random variables. And $sup(s)$ can be modeled by its probability mass function (pmf), denoted by $sup(s) = \{sup(s)|0 : p_0, 1 : p_1, \ldots, n : p_n\}$. Here $n = |D|$ is the number of sequences in D.

According to *central limit theorem*, $sup(s)$ converges to the Gaussian distribution when n goes to infinity. Therefore, in the large scale database D, we can approximate the distribution of $sup(s)$ by Equation (4.1).

$$sup(s) = \sum_{i=1}^{n} X_i \longrightarrow N(\sum_{i=1}^{n} p_i, \sum_{i=1}^{n} p_i * (1 - p_i)) \tag{3.4}$$

Here we approximate $sup(s)$ by the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, and then the approximated frequentness probability $P(sup(s) \geq \tau_s)$ can be computed in linear time.

3.4.2 Support Probability

The support probability $P(s \sqsubseteq d)$ is the probability that a sequential pattern s is supported by an uncertain sequence d and it can be computed in (3.5) according to possible world semantics.

$$P(s \sqsubseteq d) = \sum_{\forall w, s \sqsubseteq w} P(w) \tag{3.5}$$

Where w is a possible world of d which supports s and $p(w)$ is its existential probability. However, suppose each item in a k-length pattern s has m multiple occurrences in d in average, there are $O(k^m)$ possible worlds that may support s in the worst case. And directly enumerating all of them is usually too complex in practice.

Therefore, we design an incremental approach to compute support probability efficiently. Let l be the last item of sequential pattern s. In uncertain sequence d, suppose there are q possible occurrences of l in events $e_{k_1}, \ldots, e_{k_q}$, then all the possible worlds that may support s can be divided into q disjoint subsets $(g_1, \ldots, g_q)$ by the most recent occurrence of item l.

Let $P(g_i)$ be the probability that the latest occurrence of item l (the last item of s) is in e_{k_i}, then it can be computed by Equation (3.6).

$$P(g_i) = P(l \in e_{k_i}) * \prod_{t=i+1}^{q} P(l \notin e_{k_t}) \tag{3.6}$$

The amortized cost of Equation (3.6) is $O(1)$, when events are pre-sorted by their *eid*s. And the support probability $P(s \sqsubseteq d)$ can be computed in (3.7).

$$P(s \sqsubseteq d) = \sum_{i=1}^{q} P(s \sqsubseteq d | g_i) * P(g_i) = P(s \sqsubseteq d \cap g_i) \tag{3.7}$$

For example, given $d = \langle (B : 0.5)(C^1 : 0.4)(C^2 : 0.4) \rangle$ and $s = \langle BC \rangle$, according to possible world semantics, there are three possible worlds of d that may support s: $w_1 = \{BC^1\}$, $w_2 = \{BC^2\}$ and $w_3 = \{BC^1C^2\}$, and we divide them into two disjoint groups by the latest occurrence of item C in the possible worlds as $g_1 = \{w_1\}$ and $g_2 = \{w_2, w_3\}$. We first compute $P(g_1) = 0.4 * 0.6 = 0.24$ and $P(g_2) = 0.4$, then we have $P(s \sqsubseteq d) = 0.5 * 0.24 + 0.5 * 0.4 = 0.22$.

Suppose l is the last item of s, then $s' = s - \{l\}$ is a $(k-1)$-length sequential pattern. $P(s \sqsubseteq d|g_i)$ in (3.7) can be computed by (3.8).

$$P(s \sqsubseteq d|g_i) = \sum_{j=1}^{p} P(s' \sqsubseteq d|g_j) * P(g_j|g_i) = \sum_{j=1}^{p} P(s' \sqsubseteq d \cap g_j) * \delta(g_i, g_j) \tag{3.8}$$

Where g_j ($\forall j \in [1,p]$) are p disjoint subsets of possible worlds in which the latest occurrence of the last item of s' in the event e_{k_j}. And $\delta(g_j, g_i) = 1$, if the last item of s' occurs before the last item of s; otherwise, $\delta(g_j, g_i) = 0$.

By substituting (3.8) into (3.7), we can compute the support probability in (3.9).

$$P(s \sqsubseteq d) = \sum_{i=1}^{q} \sum_{j=1}^{p} P(s' \sqsubseteq d \cap g_j) * P(g_i) * \delta(g_i, g_j) \tag{3.9}$$

Therefore, if we save and reuse the values of $P(s' \sqsubseteq d \cap g_j)$, we can avoid repeated computation which reduces the time complexity of support probability computation from exponential to $O(p * q)$.

3.4.3 Vertical Data Structure

We develop a vertical data format D_k to save occurrences of k-length candidate patterns. The schema of D_k is $\langle sid, c, tid, P_c, P_i \rangle$, where sid identifies an uncertain sequence d, c is a candidate pattern and $\langle tid, P_c, P_i \rangle$ records an occurrence of c in d. Suppose i is the last item of c and e is the event identified by (sid, tid), then we have $P_c = P(c \sqsubseteq d \cap g_i)$, where g_i is a subset of possible worlds in which the latest occurrence of item i locates in event e. And $P_i = P(i \in e)$ is the existential probability of i in e.

We transform the original sequence database into its vertical format which is a set of candidate occurrences. Figure 4.4 shows an example of constructing the vertical data format D_k. Here D is the original database, and D_1 is transformed from D. For example, let $s = \langle A \rangle$, then we have two groups g_1 and g_2 of occurrences of s in sequence d_1. We compute $P_{c1}(s) = 1 * P(g_1) = 0.3 * 0.5 = 0.15$ and $P_{c2}(s) = 0.5$ and save the results in D_1. Thereafter, we can compute the support probabilities

D

sid	tid	I
1	1	A:0.3
1	2	A:0.5
1	3	B:0.4
2	1	A:0.4
2	2	B:0.8
2	3	B:0.7

D1

sid	1			2		
c	<A>		<B>	<A>	<B>	
tid	1	2	3	1	2	3
Pc	0.15	0.5	0.4	0.4	0.24	0.7
Pi	0.3	0.5	0.4	0.4	0.8	0.7

D2

sid	1	2	
c	<AB>	<AB>	
tid	2	2	3
Pc	0.26	0.096	0.28
Pi	0.4	0.8	0.7

Figure 3.3. An example of constructing the vertical data structure

$P(s \sqsubseteq d_1) = 0.65$ and $P(s \sqsubseteq d_2) = 0.4$ from D_1, which are used to calculate the frequentness probability. In this example, if we set $minsup = 1$ and $minprob = 0.5$, then $\langle A \rangle$ and $\langle B \rangle$ are two frequent patterns. 2-length candidates are generated by self-joining 1-length frequent patterns, and their occurrences are saved in D_2. For example, let $s' = \langle AB \rangle$, then $P(s' \sqsubseteq d_1) = 0.65 * 0.4 = 0.26$. Since there are two occurrences of item B in d_2, we first compute $P(g_1) = 0.8*0.3 = 0.24$ and $P(g_2) = 0.7$, then we have $P_{c1}(s') = 0.4 * 0.24 = 0.096$ and $P_{c2}(s') = 0.4 * 0.7 = 0.28$. Thereafter, the support probability $P(s' \sqsubseteq d_2) = 0.376$.

In our approach, we only refer to D_k in searching k-length frequent patterns. And D_k is usually in a much smaller size than the original database because it only contains occurrences of potential frequent candidate patterns.

3.4.4 Uncertain SPM in Iterative MapReduce

Our iterative MapReduce framework helps to traverse a huge sequence tree [36] in searching frequent patterns in parallel. In each iteration, we start a MapReduce job to search k-length frequent patterns on a cluster of computers.

Figure 4.2 shows our iterative MapReduce framework for uncertain SPM. In the first iteration, the original database is split and input to mappers; in the k^{th} ($k > 1$) iteration, the input data of a mapper is a chunk of D_{k-1}. We modify the data split

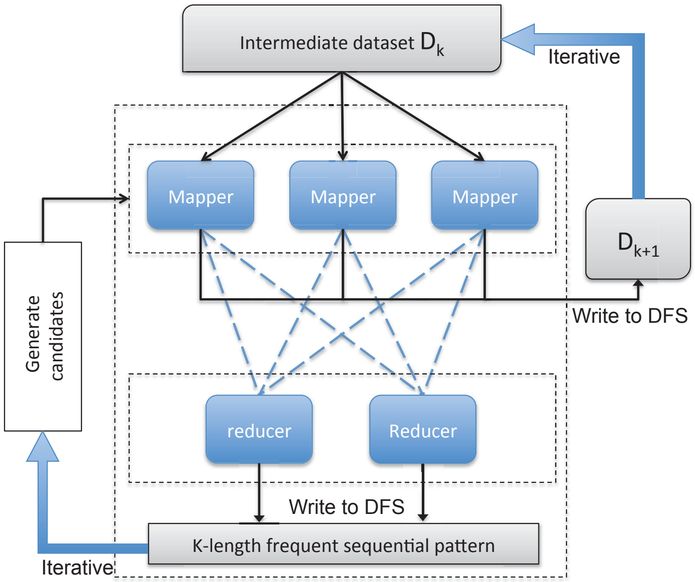


Figure 3.4. Iterative MapReduce framework for uncertain sequential pattern mining

function in MapReduce to make sure that all occurrences in one sequence are input to the same map function. A set of k-length candidate patterns are distributed to mappers, which is denote by C_k.

(1) *Mapper function*: The mapper function is shown in Algorithm 3.1. It first constructs d_k from d_{k-1} and C_k, where $d_{k-1} \in D_{k-1}$ contains occurrences of $(k-1)$-length frequent patterns in one uncertain sequence. Given a candidate pattern c, the mapper computes the support probability $p = P(c \sqsubseteq d_k)$ using the newly updated data structure and outputs a key-value pair $\langle c, \langle \mu, \sigma^2 \rangle \rangle$ if $p = P(c \sqsubseteq d) > 0$. Here $\mu = p$ and $\sigma^2 = p * (1 - p)$ are the mean and variance of the Bernoulli random variable $sup(c|d_k)$. Thereafter, d_k is written to distributed file system (DFS) to be used in the next iteration.

(2) *Combiner function*: We design a combiner function in Algorithm 3.2 to help improve the performance. Suppose a mapper function emits n key-value pairs $\langle c, \langle \mu_i, \sigma_i^2 \rangle \rangle$ $(i = 1, \ldots, n)$ which are associated with the identical pattern c. As the value filed of the mapper output is associative and commutative, they can be condensed to a sin-

Algorithm 3.1: Map(Key *key*, Value *value*, Context *context*)

```
$d_{k-1} \leftarrow$ pase($value$)                 /* $d_{k-1} \in D_{k-1}$ parsed from value */
$C_k \leftarrow$ DistributedCache.file
$d_k \leftarrow$ construct from $C_k$ and $d_{k-1}$
foreach $c \in C_k$ do
    $p \leftarrow P(c \sqsubseteq d_k)$            /* computed by summing $P_c(c)$ in $d_k$ */
    $key \leftarrow c$;
    $value \leftarrow \langle p, p * (1-p) \rangle$        /* composited value */
    context.collect($key$, $value$)
end
DFS.file $f$ = new DFSFile("$D_k$");
$f$.append($d$);
```

Algorithm 3.2: Combine(Key *key*, Iterable *values*, Context *context*)

```
$\mu \leftarrow 0$, $\sigma^2 \leftarrow 0$
foreach $value \in values$ do
    $\mu = \mu + value.\mu$
    $\sigma^2 = \sigma^2 + value.\sigma^2$
end
$context.collect(key, \langle \mu, \sigma^2 \rangle)$
```

gle pair $\langle c, \langle \sum_1^n u_i, \sum_1^n \sigma_i^2 \rangle \rangle$. Then each mapper sends only one key-value pair to the reducer for each candidate pattern, which dramatically reduce the total bandwidth cost of data shuffling.

(3) *Reducer function*: Algorithm 3.3 shows the reducer function. The input key-value pair of the reducer is in the form of $\langle c, \langle \mu_i, \sigma_i^2 \rangle \rangle$, where $\mu_i = \sum p$ and $\sigma_i^2 = \sum p * (1-p)$ are the partially aggregated mean and variance of the probabilistic support of candidate c. The reducer function accumulates the overall mean and variance of c in the entire uncertain database and uses the Gaussian distribution

Algorithm 3.3: Reduce(Key *key*, Iterable *values*, Context *context*)

```
$c \leftarrow key$
$\mu \leftarrow 0$, $\sigma^2 \leftarrow 0$
foreach $value \in values$ do
    $\mu = \mu + value.\mu$
    $\sigma^2 = \sigma^2 + value.\sigma^2$
end
$sup(c) \sim N(\mu, \sigma^2)$
$\tau_s \leftarrow context.minsup$, $\tau_p \leftarrow context.minprob$
if $P(sup(c) \geq \tau_s) \geq \tau_p$ then
    DFS.file f = new DFSFile("frequent pattern");
    $f$.append($c$);
end
```

to approximate the distribution of overall support $sup(c)$. Given $minsup = \tau_s$ and $minprob = \tau_p$, the reducer outputs the probabilistic frequent sequential patterns to the file, if $P(sup(c) \geq \tau_s) \geq \tau_p$; otherwise, c is not probabilistic frequent and is discarded by the reducer.

A MapReduce iteration is finished after all k-length probabilistic frequent sequential patterns are discovered and written to DFS files. After that, we self-join k-length frequent patterns to generate $(k+1)$-length candidate patterns for the next iteration. This process continues until all frequent patterns are discovered.

3.5 Evaluation

In this section, we implement our uncertain SPM algorithm in iterative MapReduce, denoted by *IMRSPM*, and evaluate its performance using both synthetic and real world datasets in a 10-node Hadoop cluster.

A naïve method directly enumerates possible worlds table without reusing previous computational results. We implement this naïve approach in Iterative MapReduce as *baseline*, which is denoted by *BL* here. We also compare our algorithm with the single-machine uncertain sequential pattern mining algorithm, denoted by *UPrefix* [15, 16], to show the benefit from parallel computing.

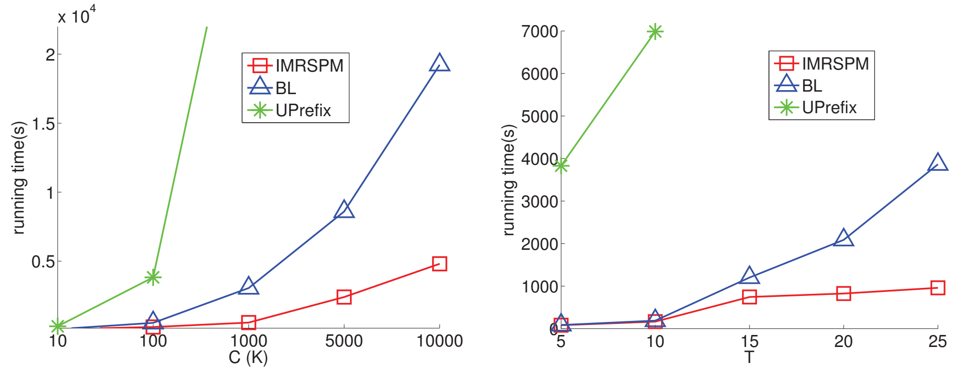
3.5.1 Synthetic Dataset Generation

The IBM market-basket data generator [46] uses the following parameters to generate sequence datasets in various scales: (1) C: number of customers; (2) T: average number of transactions per sequence; (3) L: average number of items per transaction per sequence; (4) I: number of different items.

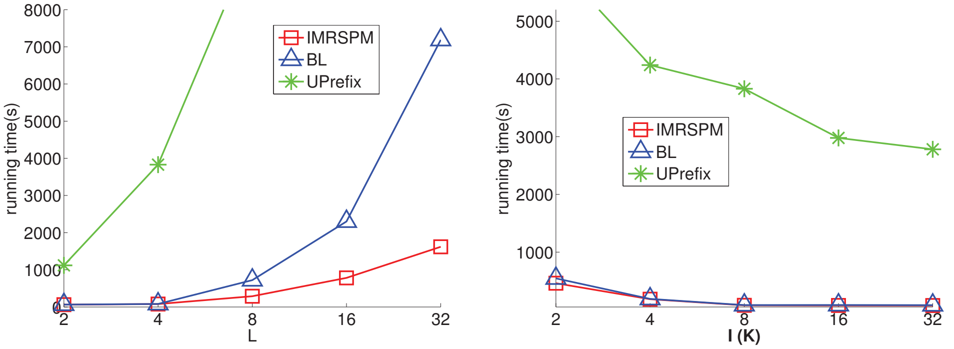
We assume that an event existential probability follows normal distribution $t \sim N(\mu, \sigma^2)$, where μ is randomly drawn from range $[0.7, 0.9]$ and σ is randomly drawn from range $[1/21, 1/12]$. Then we draw a value from t and assign it to an event in the original synthetic datasets as its existential probability. This approach has been used in previous work [11] to generate synthetic uncertain datasets. We name a synthetic uncertain dataset by its parameters. For example, a dataset T4L10I10C10 indicates $T = 4$, $L = 10$, $I = 10 * 1000$ and $C = 10 * 1000$.

3.5.2 Scalability

In Figure 3.5, we evaluate the scalability of IMRSPM on synthetic datasets generated by different parameters. Here we set $minsup = 0.2\%$ and $minprob = 0.7$. Figure 6.2(a) shows the running time variations of IMRSPM when C varies from 10 000 to 10 000 000, where $T = 4$, $L = 4$, $I = 10\,000$. Figure 6.2(b) shows the running time variations of IMRSPM when T varies from 5 to 25, where $C = 100\,000$, $L = 4$, $I = 10\,000$. Figure 6.2(c) shows the running time variations of IMRSPM when L varies from 2 to 32, where $C = 100\,000$, $T = 4$, $I = 10\,000$. Figure 6.2(l) shows the running time variations of IMRSPM when I varies from 2 000 to 32 000, where $C = 100\,000$, $T = 4$, $L = 4$.



(a) scale of C (b) scale of T



(c) scale of L (d) scale of I

Figure 3.5. Scalability of IMRSPM-A algorithm

In Figure 3.5, we observe the following phenomenons:

(1) IMRSPM outperforms BL under every setting of the parameters, which proves the effectiveness of our incremental temporal uncertainty management approach; meanwhile, IMRSPM is much more scalable than UPrefix, which demonstrates the advantage of using iterative MapReduce framework.

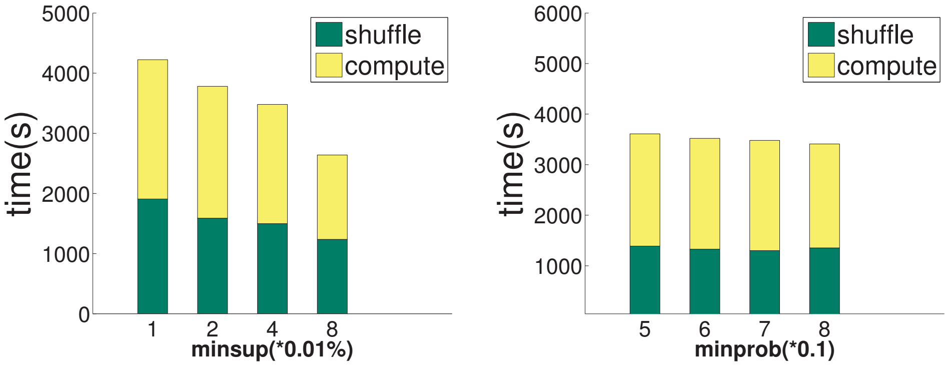
(2) The running time increase with the increment of C, T, L, as increasing these parameters generates larger scale datasets. Furthermore, when T or L are set to larger values, there are more repeated items in uncertain sequences. And our incremental uncertainty management approach shows its effectiveness in improving the efficiency especially in such cases.

(3) The running time slightly drops with the increment of I. When the value of I grows, the number of repeated item in one sequence become less because items are randomly selected from a fixed set of items.

3.5.3 Mining Customer Behavior Patterns from Amazon Reviews

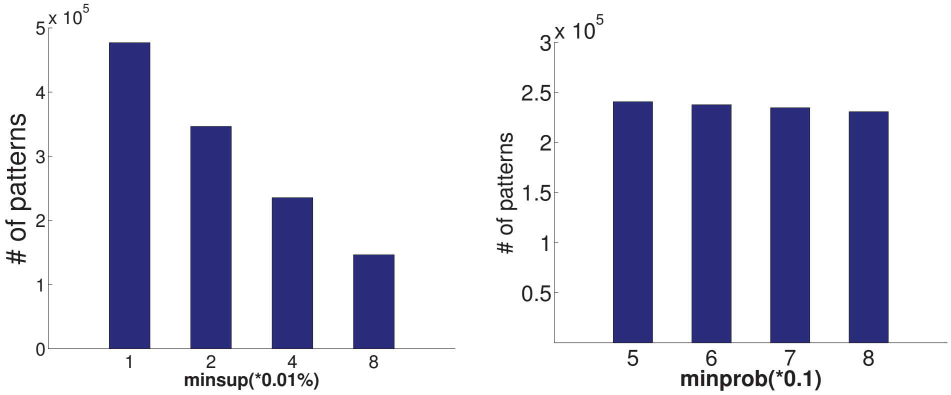
We apply our IMRSPM algorithm in Amazon review dataset [52] to discover customer behavior patterns. The Amazon review dataset includes 34 686 770 reviews of 2 441 053 products from 6 643 669 customers between June 1995 to March 2013. Each review is scored by an integer between 1 to 5, which indicates a user opinion toward a product. However, this score is a lose measurement of subjective satisfaction. Suppose a customer gives a score t to a product, then we believe that the probability that this customer likes this product is $p = t/5$. An ordered list of user reviews is regarded as an uncertain sequence. A probabilistic frequent sequential pattern $\langle A, B\rangle$ mined from this database can be explained as: if a customer likes product A, then it is very likely that he/she will like product B in the future.

For example, given $minsup = 0.005\%$ and $minprob = 0.7$, we have discovered the sequential pattern ⟨B000TZ19TC → B000GL8UMI⟩. Here B000TZ19TC is the Amazon Standard Identification Number (ASIN) of the book *Fahrenheit 451* published in 1953. And this pattern reveals that users who now like product B000TZ19TC may also like B000GL8UMI in the future, which is a newer edition of the same book published in 1963. Meanwhile, we also discover other non-trivial patterns such as ⟨B000MZWXNA → B000PBZH6Q⟩, where B000MZWXNA is associated with the book *The Martian Way* and ASIN B000PBZH6Q identifies the book *Foundation*. Figure 3.6 and Figure 3.7 show the effect of user-defined parameters *minsup* and *minprob* in Amazon dataset. We initially set $minprob = 0.7$ and $minsup = 0.04\%$. In Figure 3.6(a) and 3.7(a), we vary the value of *minsup* from 0.02% to 0.04%; while *minprob* is varied from 0.5 to 0.8 in Figure 3.6(b) and 3.7(b). From Figure 3.6 and Figure 3.7 , we observe that:



(a) vary *minsup* (b) vary *minprob*

Figure 3.6. Effect of user-define parameters in efficiency



(a) vary *minsup* (b) vary *minprob*

Figure 3.7. Effect of user-define parameters in number of patterns

(1) In Figure 3.6(a), the running time of IMRSPM decreases with the increment of *minsup*; meanwhile, the effect of *minsup* to the computing time is more significantly than that to the shuffling time. The reason is that fewer frequent patterns are mined when *minsup* is larger, which can be proved by Figure 3.7(a).

(2) The performance remains relatively stable to the variation of *minprob*. The probabilistic support of a sequential pattern is bounded to its expected value (*Chernoff*

bound). Thus, the frequentness of a large number of candidate patterns becomes deterministic, and this explains why the running time and the number of frequent patterns do not significantly fluctuate in Figure 3.6(b) and 3.7(b).

3.6 Conclusions

In this chapter, we develop a scalable sequential pattern mining algorithm with iterative MapReduce in sequence databases with existential uncertainty. We adopt possible worlds to interpret the uncertain database and design an incremental approach to track patterns' frequentness probabilities. We extend the Apriori-like SPM framework to MapReduce, and then integrate our uncertainty management technique into the distributed platform and optimize the serialization cost by a new data structure. The experimental results on synthetic datasets demostrates that our approach is efficient and scalable. And we also apply our method to discover customer behavior patterns in Amazon review dataset, which contains millions of uncertain sequences.

4 DISTRIBUTED UNCERTAIN SEQUENTIAL PATTERN MINING

4.1 Introduction

Sequential pattern mining (SPM) is one of the most important applications in data mining. It is widely used to analyze customer behaviors in market-basket databases. For example, online shopping websites usually collect customer purchasing records in databases where sequential patterns are mined to reveal buying habits of consumers. However, in many real applications, events occurring in a sequence may be uncertain for many reasons. For instance, data collected by sensors are inherently noisy; in privacy protection applications, artificial noises are added deliberately; data modeling techniques such as classification may also produce indeterministic results.

Example 4.1 *Consider an online travel website. To increase sales, a large group of customers are analyzed in order to discover sequential patterns of user-interested products. These patterns are useful in intelligent marketing. For example, by providing a special hotel offer to a customer who books a late-night flight, the website is able to encourage hotel purchases.*

Figure 4.1(a) records user preferences to various travel products. For example, in the session B, the customer is at first surely attracted by a rental car and then shows interests in a hotel with a probability of 0.7. Here user preferrences are estimated by posterior probabilities of Naïve Bayesian models which takes implicit feedback (e.g. views, clicks, purchases, likes, shares etc.) into consideration. The website uses a database that represents each visiting session as a single sequence, also shown in Figure 4.1(b).

Session	timestamp	Products	Prob.
A	1	flight	0.7
A	2	flight, hotel	0.6
A	3	hotel	0.8
B	1	car	1.0
B	2	hotel	0.7

SID	Sequence
S_A	<1, (flight),0.7>;<2, (flight, hotel), 0.6>; <3, (hotel), 0.8)>
S_B	<1, (car), 1.0>;<2, (hotel), 0.7>

(a) uncertain sequence database

world	Sequence	Prob.
1	<(flight)>; <(flight, hotel)>; <(hotel)>	0.336
2	<(flight)>;<(flight, hotel)>	0.084
3	<(flight)>;<(hotel)>	0.224
4	<(flight)>	0.056
5	<(flight, hotel)>;<(hotel)>	0.144
6	<(flight, hotel)>	0.036
7	<(hotel)>	0.096
8	ϕ	0.024

(b) Possible worlds of S_A

Figure 4.1. An example of uncertain sequence databases

4.1.1 Problem Statement

The uncertain model applied here is based on existential *uncertain events*, which is also called *sequence-level* uncertainty in [15].

Definition 4.1.1 *An uncertain event is an event e whose presence in a sequence d is defined by an existential probability $P(e \in d) \in (0, 1]$.*

Definition 4.1.2 *An uncertain sequence d is an ordered list of uncertain events. An uncertain sequence database is a collection of uncertain sequences.*

Definition 4.1.3 *A sequential pattern $s = \langle s_1, \ldots, s_n \rangle$ is an ordered list of itemsets where the itemset $s_i \in s$ is also called an* element *of s.*

In traditional certain databases, a sequential pattern $s = \langle s_1, \ldots, s_n \rangle$ is *supported* by a sequence $d = \langle e_1, \cdots, e_m \rangle$, denoted by $s \preceq d$, if there exists n integers $1 \leq k_1 < \ldots, k_n \leq m$ that have $s_i \subseteq e_{k_i}$ for $i \in [1, n]$. A sequential pattern is *frequent* if at least τ_s sequences support it, where τ_s is a user-specified threshold. However, in an uncertain database D, the support of a pattern s, denoted by $sup(s)$, is uncertain. We define *probabilistic frequent sequential patterns* (p-FSP) as follows:

Definition 4.1.4 (Probabilistic Frequent Sequential Pattern) *A sequential pattern s is a probabilistic frequent sequential pattern (p-FSP) if its probability of being frequent is at least τ_p, denoted by $P(sup(s) \geq \tau_s) \geq \tau_p$.*

Here τ_p is the user-defined minimum confidence in the frequentness of a sequential pattern. We are now able to specify the uncertain SPM problem as: *Given* τ_s, τ_p, *find all p-FSPs in* D.

In an uncertain database, sequences are often assumed to be mutually independent, which is also known as the *tuple-level independence* [28, 43]. Therefore, the overall support $sup(s)$ in D is a sum of uncertain supports in every single sequence. And the probabilistic support of s in an uncertain sequence d_i can be modeled by a Bernoulli random variable $X_i \sim B(1, p_i)$, where $p_i = P(s \preceq d_i)$ is the probability that d_i supports s.

When the size of D grows, we approximate the distribution of $sup(s)$ by the Gaussian distribution in Equation (4.1), according to *the central limit theory.*

$$sup(s) \xrightarrow{|D|\rightarrow\infty} \mathcal{N}\big(\sum_{i=1}^{|D|}\mu_i, \sum_{i=1}^{|D|}\sigma_i^2\big) \tag{4.1}$$

Where $\mu_i = p_i$ and $\sigma_i^2 = p_i * (1 - p_i)$ are the mean and variance of $X_i \sim \mathcal{B}(1, p_i)$. We are now able to compute the *approximate frequentness probability* of s by:

$$P(sup(s) \geq \tau_s) = 1 - P\big(sup(s) \leq \tau_s - 1\big) = 1 - \Phi(\frac{\tau_s - 1 - \mu}{\sigma}) \tag{4.2}$$

With this approximation, s is a p-FSP if $P(sup(s) \geq \tau_s) \geq \tau_p$.

Now the key issue of uncertain SPM is the computation of support probabilities. In this chapter, we focus on the problem of calculating support probabilities in large scale databases and extracting all p-FSPs. Meanwhile, in order to mine highly scalable databases, we extend the Apriori-like framework of uncertain SPM to the distributed computing platform Spark [53], which allows us to load a large amount of data into a cluster's memory and query it repeatedly.

4.1.2 Contribution

In this chapter, we propose a **D**istributed **S**equential **P**attern (DSP) mining algorithm in large scale uncertain databases. Our main contributions are summarized:

(1) We propose a SPM framework for mining large scale uncertain databases in Spark.
(2) We develop a distributed dynamic programming method to compute support probabilities with linear time and space complexity.
(3) We design a new data structure by extending the prefix-tree to save intermediate results space efficiently.
(4) Extensive experiments conducted in various scales shows that our algorithm is orders of magnitude faster than both direct extension and existing works.

4.2 Related Works

Uncertain data mining has been an active area of research recently. Many traditional database and data mining techniques have been extended to be applied to uncertain databases [43]. Muzammal et al. propose a solution of SPM in probabilistic database [17] using the *expected support* to measure pattern frequentness. Zhao et al. define *probabilistic frequentness* of sequential patterns under possible world model and propose a pattern-growth uncertain SPM algorithm [15]. These two interestingness criteria is studied in [54] from a complexity-theoretic perspective. In order to improve the efficiency, approximation techniques are also explored to mine large scale uncertain databases [16]. Li et al introduce a dynamic programming approach to mine sequential patterns in a specific spatial-temporal uncertain model [40]. Wan et al. [39] propose a dynamic programming algorithm of mining frequent serial episodes within one uncertain sequence. However, all the above mentioned methods can only be executed in a single machine and may have scalability issues in mining large databases.

Chen et al. extend the classic SPAM algorithm to its MapReduce version SPAMC [50]. Huang et al. propose a distributed progressive SPM algorithm in [55]. Miliaraki et al. propose a gap-constraint frequent sequence mining algorithm in MapReduce [51]. The classic SPM algorithm PrefixSpan [35] is extended to its Spark version [56]. However, these algorithms are applied in the context of deterministic data, while our work aims to solve large scale uncertain SPM problems. An iterative MapReduce implementation of uncertain SPM in [57] is relatively close to our work; however, it has a quadratic complexity of support probability computation, and the time cost of that in our algorithm is linear.

4.3 Uncertain SPM Framework in Spark

First of all, we define the following two types of sequential pattern extension.

Definition 4.3.1 (Item-extended Pattern) *An item-extended pattern s is a sequential pattern generated by adding a new item i to the first element of another sequential pattern s', denoted by $s = \{i\} \cup s'$.*

Definition 4.3.2 (Sequence-extended Pattern) *A sequence-extended pattern s is a sequential pattern generated by adding a new itemset $\{i\}$ to another sequential pattern s' as its first element, denoted by $s = \{i\} + s'$.*

For example, let $s' = \langle (b)(d) \rangle$, then $s_1 = \langle (a, b)(d) \rangle$ is an item-extended pattern of s' and $s_2 = \langle (a)(b)(d) \rangle$ is sequence-extended froms'.

Considering the Apriori property of p-FSPs in Lemma 4.1 [15, 16], we can prune a pattern if it is extended from a pattern which is not a p-FSP.

Lemma 4.1 *If s is extended from s' and s is a p-FSP, then s' is also a p-FSP.*

Figure 4.2 shows the Apriori-like uncertain SPM framework in distributed computing platform Spark. An uncertain sequence database $D = \{d_1, \ldots, d_n\}$ is abstracted by an RDD [53] in Spark. Uncertain sequences in the RDD are allocated to a cluster of machines and can be processed in parallel.

Map. A map function is used to compute support probabilities. A set of candidate patterns are broadcasted to all the mappers. For each candidate pattern c, the map function first computes the support probability $p_i = P(c \preceq d_i)$ in the uncertain sequence d_i, then it emits a key-value pair as $\langle c, (\mu_i, \sigma_i^2) \rangle$, if $p_i > 0$. The key field

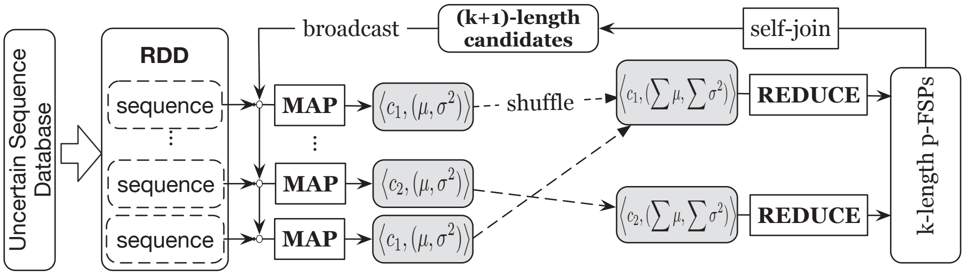

Uncertain Sequence Database
RDD
sequence
sequence
sequence
MAP
MAP
MAP
broadcast
(k+1)-length candidates
self-join
$\langle c_1,(\mu,\sigma^2)\rangle$
$\langle c_2,(\mu,\sigma^2)\rangle$
$\langle c_1,(\mu,\sigma^2)\rangle$
shuffle
$\langle c_1,(\sum\mu,\sum\sigma^2)\rangle$
$\langle c_2,(\sum\mu,\sum\sigma^2)\rangle$
REDUCE
REDUCE
k-length p-FSPs

Figure 4.2. A framework of uncertain SPM in Spark

here is the pattern c; the composite value field contains both mean μ_i and variance σ_i^2 of the Bernoulli distributed probabilistic support $X_i \sim \mathcal{B}(1, p_i)$ of c in this sequence. The key-value pairs are designed to be associative and commutative so that mappers can aggregate them in local machines before shuffling to reducers, which saves a lot of network bandwidth usage.

Reduce. Pairs with the same key are shuffled to one reducer. In a reduce function, it computes the approximate frequentness probability for each candidate by Equation (4.2). All candidates with $P(sup(c) \geq \tau_s) \geq \tau_p$ are saved to a set of k-length p-FSPs, denoted by S_k.

Self-join. we self-join all k-length p-FSPs in S_k to generate a set of $(k+1)$-length candidate patterns in C_{k+1}. Let s_1 and s_2 be two p-FSPs in S_k. Suppose s'_1 is the pattern generated by removing the first item i in s_1 and s'_2 is the pattern generated by removing the last item of s_2. If $s'_1 = s'_2$, we join s_1 and s_2, denoted by $s_1 \bowtie s_2$, to generate a (k+1)-length candidate c according to the following rules: If s_1 is sequence-extended, $c = \{i\} + s_2$; if s_1 is item-extended, $c = \{i\} \cup s_2$. For example, let $s_1 = \langle(a)(bc)\rangle$, $s_2 = \langle(bc)(d)\rangle$ and $s_3 = \langle(c)(de))\rangle$, then $s_1 \bowtie s_2 = \langle(a)(bc)(d)\rangle$; while $s_2 \bowtie s_3 = \langle(bc)(de)\rangle$.

Stop criterion. If either S_k or C_{k+1} is empty, we terminate the mining process; otherwise, C_{k+1} is broadcasted to all map functions for the next iteration.

4.4 A Distributed Dynamic Programming Approach

4.4.1 Dynamic Programming in Support Probability Computation

We adopt a DP method to compute support probabilities. The key is to consider it in terms of sub-problems. Here we first define $P_{i,j}$ in Definition 4.4.1.

Definition 4.4.1 *$P_{i,j} = P(s_i^n \preceq d_j^m)$ is the probability that s_i^n is supported by d_j^m, where $s_i^n = \langle s_i, \ldots, s_n \rangle$ is a subsequence of a sequential pattern s and $d_j^m = \langle e_j, \ldots, e_m \rangle$ is a subsequence of an uncertain sequence d.*

Therefore, $P(s \preceq d) = P_{1,1}$. The idea here is to split the problem of computing $P_{i,j}$ into sub-problems $P_{i,j+1}$ and $P_{i+1,j+1}$. And this can be achieved as follows: in condition of $s_i \subseteq e_j$, $P_{i,j}$ is equal to the probability that s_{i+1}^n is supported by d_{j+1}^m; if $s_i \not\subseteq e_j$, $P_{i,j}$ is equal to the probability that s_i^n is supported by d_{j+1}^m. By splitting the problem in this way we can use the recursion in Lemma 4.2 to compute $P_{i,j}$ by means of the paradigm of dynamic programming.

Lemma 4.2

$$P_{i,j} = P(s_i \subseteq e_j) * P_{i+1,j+1} + P(s_i \not\subseteq e_j) * P_{i,j+1} \tag{4.3}$$

where $P(s_i \subseteq e_j)$ is the probability that s_i is contained in event e_j. And $P(s_i \subseteq e_j) = P(e_j \in d)$, if $s_i \subseteq e_j$; otherwise, $P(s_i \subseteq e_j) = 0$.

Proof Referring to *the law of total probability*, we have:

$$P_{i,j} = P(s_i^n \preceq d_j^m | s_i \subseteq e_j) * P(s_i \subseteq e_j) + P(s_i^n \preceq d_j^m | s_i \not\subseteq e_j) * P(s_i \not\subseteq e_j)$$

where $P(s_i^n \preceq d_j^m | s_i \subseteq e_j) = P_{i+1,j+1}$ is the probability that $s' = \langle s_{i+1}, \ldots, s_n \rangle$ is supported by sequence $\langle e_{j+1}, \ldots, e_m \rangle$. And similarly we have $P(s_i^n \preceq d_j^m | s_i \not\subseteq e_j) = P_{i,j+1}$ is the support probability of s_i^n in d_{j+1}^m. ■

This dynamic schema is an adoption of the technique previously used in solving uncertain SPM [17] and frequent episode mining problems [39]. Using this dynamic

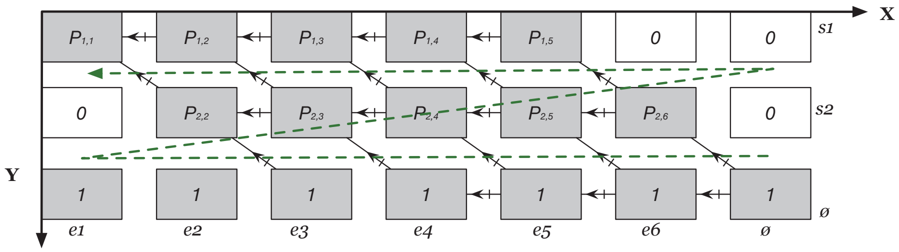


Figure 4.3. An example of dynamic programming process

programming scheme, we can compute the support probability by calculating the cells depicted in Figure 4.3. In the matrix, each cell relates to a probability $P_{i,j}$, with i marked on the x-axis and j marked on the y-axis. Referring to Lemma 4.2, we can compute $P_{i,j}$ from $P_{i,j+1}$ and $P_{i+1,j+1}$ which are cells to the right and lower right of $P_{i,j}$. By definition, if $s = \phi$, then $P(s \preceq d) = 1$; meanwhile, $P(s \preceq d) = 0$ if $s \neq \phi$ and $d = \phi$. Therefore, we iterate the cells from $P_{n+1,m+1}$ to $P_{1,1}$ so that we finally obtain $P(s \preceq d) = P_{1,1}$. The time complexity is $O(n * m)$, as we only need to iterate each cell once.

4.4.2 Distribute Dynamic Programming Schema

A straight forward extension of the dynamic programming approach in Spark needs to build a $n * m$ matrix for every support probability computation, and this might dramatically slow down the entire process because of expensive garbage collection overhead in Spark. Here we refine the original DP schema to make it more memory-efficient. First of all, we define $P_{s,j}$ as follows.

Definition 4.4.2 *Given a sequential pattern s and an uncertain sequence d, $P_{s,j}$ is defined to be the support probability $P(s \preceq d_j^m)$ where $d_j^m = \langle e_j, \ldots, e_m \rangle$ is a subsequence of $d = \langle e_1, \ldots, e_m \rangle$.*

The idea is that, in order to mine k-length p-FSPs, we save and reuse computational results in the $(k\text{-}1)^{th}$ iteration. Based on the extension type of sequential pattern s, we have different dynamic programming schemas.

Sequence-extended. If $s = \{i\} + s'$ is sequence-extended from another pattern s', then we can compute the values of $P_{s,j}$ from $P_{s',j+1}$ and $P(s_1 \subseteq e_j)$ by Equation (4.4), according to Lemma 4.2.

$$P_{s,j} = P(s_1 \not\subseteq e_j) * P_{s,j+1} + P(s_1 \subseteq e_j) * P_{s',j+1} \tag{4.4}$$

where $s_1 = \{i\}$ is the first element of s.

Item-extended. Let $s = \{i\} \cup s'$, then s'_1 is a strict subset of s_1 and we have

$$P(s_1 \subseteq e_j) = \begin{cases} P(s'_1 \subseteq e_j) & \text{if } i \in e_j \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Referring to Lemma (4.2), we can compute $P_{s,j}$ by Equation (4.6).

$$P_{s,j} = \begin{cases} P_{s,j+1}, & \text{if } i \notin e_j \\ P(s_1 \not\subseteq e_j) * P_{s,j+1} + P(s'_1 \subseteq e_j) * P(s_2^n \preceq d_j^m) & \text{otherwise} \end{cases} \tag{4.6}$$

Note that $s_2^n = s'^n_2 = \langle s_2, \ldots, s_n \rangle$, then $P_{s',j}$ can be computed by:

$$\begin{aligned} P_{s',j} &= P(s'_1 \not\subseteq e_j) * P_{s',j+1} + P(s'_1 \subseteq e_j) * P(s'^n_2 \preceq d_j^m) \\ &= P(s'_1 \not\subseteq e_j) * P_{s',j+1} + P(s'_1 \subseteq e_j) * P(s_2^n \preceq d_j^m) \end{aligned} \tag{4.7}$$

Therefore, we derive Equation (4.8) by substituting Equation (4.7) into Equation (4.6).

$$P_{s,j} = \begin{cases} P_{s,j+1} & \text{if } i \notin e_j \\ P(s_1 \not\subseteq e_j) * (P_{s,j+1} - P_{s',j+1}) + P_{s',j} & \text{otherwise} \end{cases} \tag{4.8}$$

Now we are able to compute $P_{s,j}$ from only the values of $P_{s',j}$, $P_{s',j+1}$ and $P(s_1 \subseteq e_j)$.

Equation (4.4) and (4.8) constitute our distributed dynamic programming structure for computing support probabilities in parallel. And the time complexity is $O(m)$ where $m = |d|$ is the number of events in d.

Algorithm 4.1: compute support probability from intermediate results

```
Input: $L_1$: a list of $n$ non-zero $P(s''_1 \subseteq e_i)$ values ordered by eid $i$
       $L_2$: a list of $m$ non-zero $P_{s',j}$ values ordered by eid $j$
$L_s \leftarrow \phi$, $p \leftarrow (n-1)$, $q \leftarrow (m-1)$
$P_{s',j+1} \leftarrow 0$
while $p \geq 0$ do
    $P(s_1 \subseteq e_i) = P(s'' \subseteq e_i) \leftarrow L_1[p]$          // at event $e_i$
    $P_{s',j} \leftarrow L_2[q]$                                       // at event $e_j$
    while $j < i \wedge q \geq 0$ do
        /* find the nearest event $e_j$ with non-zero value $P_{s',j}$      */
        $P_{s',i+1} = P_{s',j} \leftarrow L_2[q]$
        $q \leftarrow q - 1$
    end
    $q \leftarrow q + 1$
    if $L_2[q-1] = P_{s',i}$ then $P_{s',i} \leftarrow L_2[q-1]$
    else $P_{s',i} \leftarrow L_2[q]$
    compute $P_{s,i}$ by Equation (4.4) or (4.8) and insert it to the head of $L_s$
    $p \leftarrow p - 1$
end
return $L_s$
```

4.4.3 Memory-Efficient Distributed SPM Algorithm

Lemma 4.3 *It is not necessary to save the value of $P_{s,j}$, if $P(s_1 \subseteq e_j) = 0$.*

Proof Referring to Equation (4.4) and (4.8), if $P(s_1 \subseteq e_j) = 0$, then $P_{s,j} = P_{s,k}$ where e_k is the nearest event of e_j which has $k > j$ and $P(s_1 \subseteq e_k) > 0$. ■

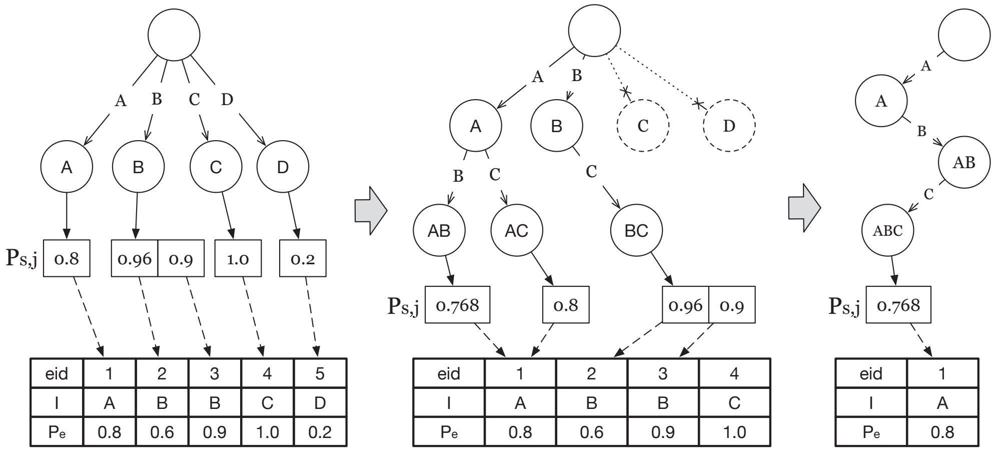
Let s be a k-length sequential pattern generated by joining two (k-1)-length patterns as $s = s'' \bowtie s'$. Then, the first element of s and s'' are identical when $k \geq 2$.

For example, let $s = \langle(a)(bc)(d)\rangle$ and $s = s'' \bowtie s'$, then $s'' = \langle(a)(bc)\rangle$, $s' = \langle(bc)(d)\rangle$, and we have $s_1 = s''_1 = (a)$.

Suppose L_1 is a list of non-zero $P(s''_1 \subseteq e_i)$ values and L_2 is a list of $P_{s',j}$ values with $P(s'_1 \subseteq e_j) > 0$. Algorithm 9 computes the values of $P_{s,i}$ from L_1 and L_2. For each value of $P(s''_1 \subseteq e_i) > 0$, we have $P(s \subseteq e_i) = P(s''_1 \subseteq e_i)$ at event e_i because $s_1 = s''_1$. Then we search the nearest event e_k, which satisfies $P_{s',k} > 0$ and $k > i$, to the right of e_i. Thus, we have $P_{s',i+1} = P_{s',k}$, by Lemma 4.3. If $P(s' \subseteq e_i) > 0$, the value of $P_{s',i}$ must have been saved and we can directly read it from L_2; if $P(s' \subseteq e_i) > 0$, $P_{s',i} = P_{s',i+1}$. Now that we have the values of $P(s \subseteq e_i)$, $P_{s',i+1}$ and $P_{s',i}$, we can compute $P_{s,i}$ by either Equation (4.4) or (4.8). Thereafter, the support probability is $P(s \preceq d) = L_s[0]$. The time complexity of Algorithm 9 is linear because both L_1 and L_2 are iterated only once.

We extend the data structure *prefix-tree* to save intermediate results such as L_1 and L_2 in each iteration of uncertain SPM. The root of the prefix tree is the empty pattern ϕ. Each edge in the tree is associated with an item. The key of a node is identified by the path from root to that node. Values are not associated with inner nodes; only leaf nodes point to a list of $P_{s,j}$ values. Each value of $P_{s,j}$ is linked to the event e_j where $P(s_1 \subseteq e_j) > 0$. Here s_1 is the first element of s.

In the prefix-tree, all the descendants of a node share a common prefix of the pattern associated with that node. Thus, it is more space-efficient, comparing to save sequential patterns individually. Figure 4.4 shows an example of computing and updating support probabilities by this new data structure. In Figure 4.4(a), the leaf nodes are 1-length sequential patterns linked to the events in an uncertain sequence. For example, the node containing pattern $s = \langle B\rangle$ is associated with two values ($P_{s,2} = 0.96$, $P_{s,3} = 0.9$) which point to uncertain events e_2 and e_3. The support probability of $\langle B\rangle$ is $P(s \preceq d) = P_{s,2}$ because e_2 is the first event where $P(s_1 \subseteq e_2) > 0$. The time cost of searching a k-length pattern in the prefix-tree is $O(k)$, and k is usually small.



(a) 1-length (b) 2-length (c) 3-length

Figure 4.4. Computing support probabilities in the prefix-tree

Another benefit of our algorithm is that we are no longer generating candidates in a centralized node; instead, we can broadcast p-FSPs to all mappers and generate candidates in parallel. In a map function applied to sequence d, if a p-FSP s is not supported, any extension of s is impossible to be supported by d. In order to generate $(k+1)$-length candidates that are potentially supported by d, we only need to join k-length p-FSPs that are supported and saved in the prefix-tree of d. In Figure 4.4(b), suppose S_1 is a set of 1-length p-FSPs and $\langle D \rangle \notin S_1$, then we can prune node D from the prefix tree and generate 2-length candidates from three 1-length p-FSPs: $\langle A \rangle$, $\langle B \rangle$ and $\langle C \rangle$ for this sequence.

In building the 2-length prefix-tree of Figure 4.4(b), we take the candidate pattern $s = \langle (A)(B) \rangle$ as an example. We first search leaf nodes associated with $s'' = \langle A \rangle$ and $s' = \langle B \rangle$ in the 1-length pattern tree. Then we retrieve $P(s''_1 \subseteq e_1) = 0.8$, $P_{s',2} = 0.96$ and $P_{s',3} = 0.9$ and compute $P_{s,1} = 0.8 * 0.96 = 0.768$ by Equation (4.4). Thereafter, we generate a new leaf node $\langle AB \rangle$ for the 2-length prefix tree.

After expanding the tree for every possible 2-length candidate, we eliminate all 1-length patterns that have not been extended. For instance, node with $\langle C \rangle$ is prune because no 2-length candidate starting with item C are potentially supported in the sequence.

The 2-length prefix tree is saved to build 3-length prefix-tree in Figure 4.4(c). And this process continues until no new candidates are generated.

4.5 Evaluation

We implement our algorithms in Spark and evaluate the performance in large scale datasets. The uncertain SPM algorithm which directly adopts dynamic programming in section 4.4.1 is denote by *basic*. We denote our distributed uncertain SPM algorithm in section 4.4.3 by *dsp*. We also implement the IMRSPM algorithm [57] in Spark and name it as *uspm* here.

We employ the IBM market-basket data generator [46] to generate sequence datasets in different scales by varying the parameters:

(1) C: number of sequences;

(2) T: average number of events per sequence;

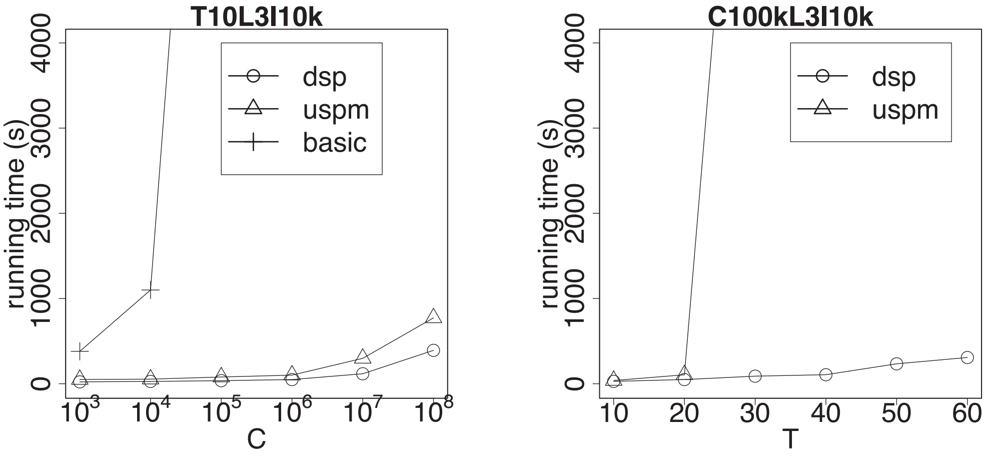
(3) L: average number of items per event per sequence;

(4) I: number of different items.

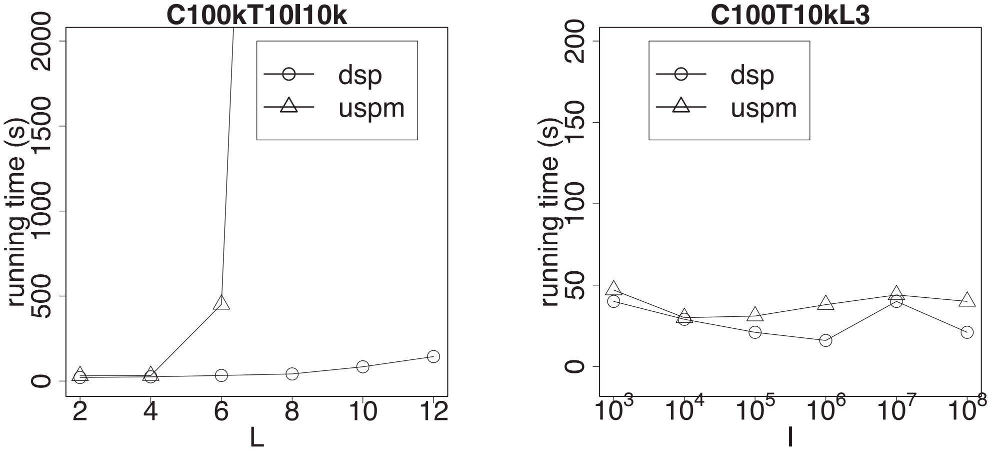
An existential probability α is added to each event in the synthetic datasets, where $\alpha \in [0.5, 0.9]$ is a parameter to control uncertain levels. We name a synthetic uncertain dataset by its parameters. For example, a dataset C10kT4L10I10k indicates $C = 10k = 10 * 1000$, $T = 4$, $L = 10$ and $I = 10k = 10 * 1000$.

4.5.1 Scalability

In this section, we evaluate the scalability of our DSP algorithm on various synthetic datasets in a Spark cluster with 100 nodes. Initially, we set uncertain level $\alpha = 0.8$ and frequentness probability threshold $\tau_p = 0.7$. In Figure 4.5(a) to 4.5(d), we set $\tau_s = 2\% * C$; in Figure 4.6(a) to 4.6(d), we have $\tau_s = 1\% * C$; and $\tau_s = 0.5\% * C$ in Figure 4.7(a) to 4.7(d). Under each setting of τ_s, we vary the values of C, T,L



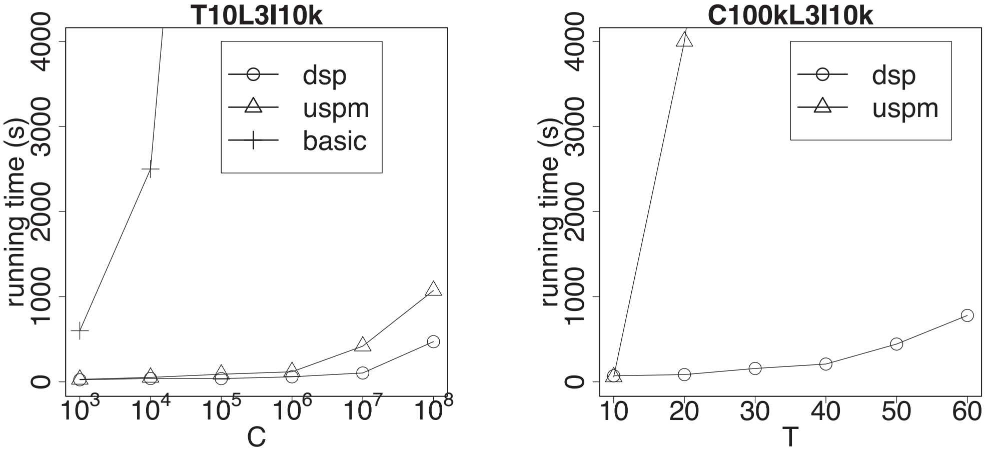
(a) C, $\tau_s = 2\%$ (b) T, $\tau_s = 2\%$



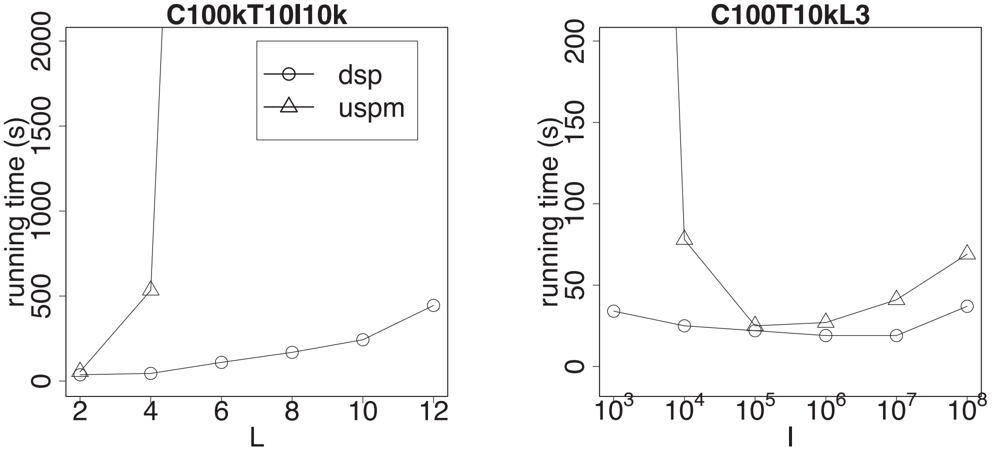
(c) L, $\tau_s = 2\%$ (d) I, $\tau_s = 2\%$

Figure 4.5. Scalability of DSP with $\tau_s = 2\%$

and I to evaluate the performance of DSP in different scales: (1) Figure 4.5(a), 4.6(a) and 4.7(a) show the running time variations of DSP when C varies from 1000 to 100 000 000, where $T = 10$, $L = 3$ and $I = 10k$. (2) Figure 4.5(b), 4.6(b) and 4.7(b) show the running time variations of DSP when T varies from 10 to 60, where $C = 100k$, $L = 4$, $I = 10k$. (3) Figure 4.5(c), 4.6(c) and 4.7(c) show the running time variations when L varies from 2 to 12, where $C = 100k$, $T = 4$, $I = 10k$. (4) Figure 4.5(d), 4.6(d) and 4.7(d) show the running time variations when I varies from 10 000 to 10 000 000, where $C = 100k$, $T = 4$, $L = 4$.



(a) C, $\tau_s = 1\%$ (b) T, $\tau_s = 1\%$



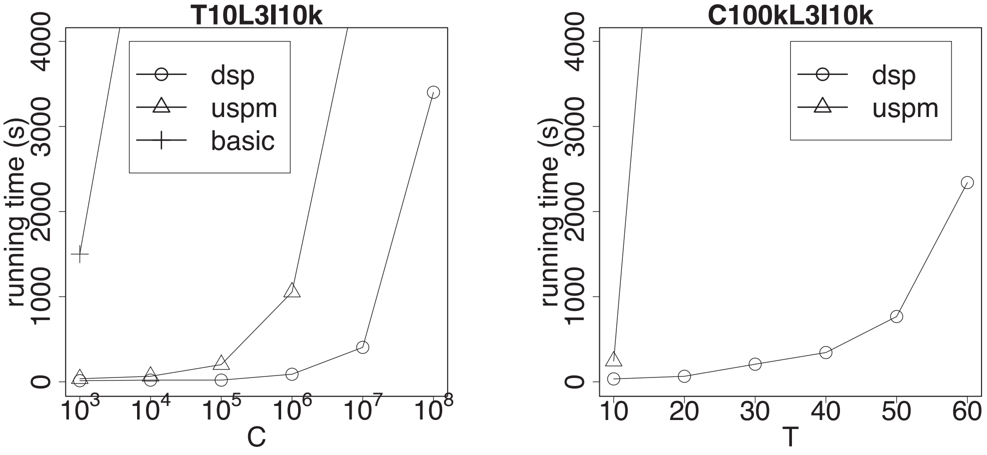
(c) L, $\tau_s = 1\%$ (d) I, $\tau_s = 1\%$

Figure 4.6. Scalability of DSP with $\tau_s = 1\%$

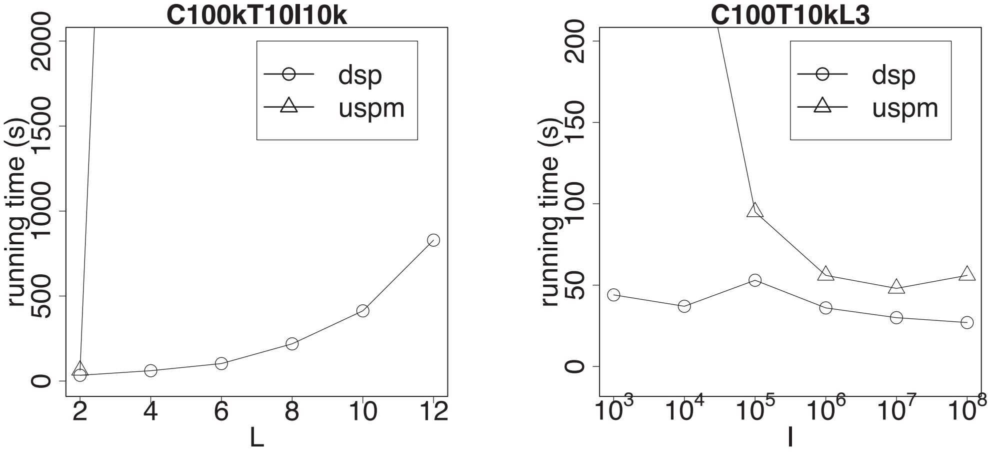
And we observe the following phenomenons from Figure 4.5, 4.6 and 4.7.

(1) *dsp* outperforms *basic* and *uspm* under every setting of the parameters. Specifically, *dsp* is orders of magnitude faster than either *basic* or *uspm* in datasets with larger values of T or L. When $C > 10k$, *basic* cannot finish because of the garbage collection overhead from Spark. This indicates that directly extending dynamic programming to Spark is not workable and also proves the advantage of our refined schemas.

(2) The running time increase with the increment of C, T, L. This is intuitive be-



(a) C, $\tau_s = 0.5\%$ (b) T, $\tau_s = 0.5\%$

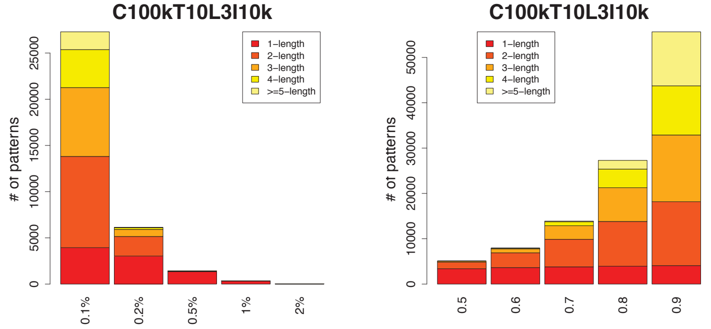


(c) L, $\tau_s = 0.5\%$ (d) I, $\tau_s = 0.5\%$

Figure 4.7. Scalability of DSP with $\tau_s = 0.5\%$

cause increasing these parameters generates larger scale datasets. Comparing to the C scale, both *dsp* and *uspm* are more sensitive to the increment of T and L. *uspm* fails quickly when T or L becomes larger; however, *dsp* still performs well even with large T or L values.

(3) The running time first drops and then arises with the increment of I. When the value of I grows, the item occurrences become more sparse, and fewer p-FSPs are mined under the same thresholds; meanwhile, the volume of data shuffled from mapper to reducer via network increases because less key-value pairs are able to be



(a) vary τ_s (b) vary α

Figure 4.8. Number of p-FSPs with different τ_s and α settings

pre-aggregated locally, which slows down the process when I is extremely large.
(4) The running time increases with the decrement of τ_s in the same dataset. This is reasonable because more p-FSPs are found when τ_s is set to a smaller value.

Figure 4.8(a) shows the number of p-FSPs in the dataset C100kT10L3I10k with uncertain level $\alpha = 0.8$, where we vary the value of τ_s from 0.1% to 2%; Figure 4.8(b) shows the effect of uncertain level α to the number of p-FSPs in C100kT10L3I10k, where $\tau_s = 0.1\%$ and $\alpha \in [0.5, 0.9]$. In Figure 4.8, we observe:
(1) With the increment of τ_s, the number of p-FSPs decreases dramatically. This is because a larger minimal support threshold makes fewer candidates be probabilistically frequent and is consistent to the phenomenon that the running time decreases with the increment of τ_s.
(2) With the increment of α, the number of p-FSPs increases, which shows the effect of data uncertainty in SPM problems. When uncertain level is high (α is small), there are fewer precise information in the data, which makes it more difficult to find p-FSPs under the same thresholds.

4.6 Conclusions

In case that no gap constraints are specified, we can further speed up the process of mining sequence databases with existential uncertainty. In this chapter, we design a dynamic programming approach to compute support probability in linear time and extend it to distributed computing platform Spark. In order to minimize the memory cost, we optimize the original schema of the dynamic programming method and design a new data structure to save intermediate results efficiently. The extensive experiments proves that our algorithm is efficient and highly scalable for large uncertain databases.

5 UNCERTAIN NEURAL NETWORK CLASSIFIER

5.1 Introduction

Classification is one of the most important applications in data mining and machine learning. Classification is the process of building a model that can describe and predict the class label of data based on feature vector [58]. And a classifier can be seen as a function that maps the features vector into class labels. Classification problem has been well studied in the recent decade; however, the new challenge in classifying uncertain data makes it to the front of research again.

In real world applications as sensor networks, the underlying data is not always accurate and precise. For example, the output of a humidity monitoring sensor may vary in a relative large range at the same position and the same time, because of its precision limitation or other reasons as random noises. In practice, this type of uncertainty are usually assumed to be Gaussian distributed. And one trivial approach of handling such uncertain data is to use the mean of the data samples to approximate the true data value. However, this simple method can only remove the influence of symmetric white noise, but cannot even save the tail behavior of the data uncertainty. A more sophisticated approach is to use a pair of parameters, mean and standard deviation, to model the Gaussian distributed data uncertainty. Table 5.1 shows an example of such an uncertain dataset.Here, the uncertain attribute value is assumed to be in Gaussian, whose mean is μ, and standard derivative is σ. And in this chapter, we design a neural network classifier based upon this specific type of uncertain data.

An artificial neural network (ANN), usually called Neural Network (NN), is a mathematical model that consist of an interconnected group of artificial neurons, and it processes information using a connection approach to computation. Modern neural networks are widely used to model complex relationships between inputs X

Table 5.1.
An example of uncertain dataset

ID	Class Label	Uncertain Attribute (μ, σ)
1	Y	(105,5)
2	N	(110,10)
3	N	(70,10)
4	Y	(120,18)

and outputs Y, as a function $f : X \to Y$. However, in conventional neural networks, the inputs X are scalar values, and we extend it to handle Gaussian distributed uncertain data in next sections.

5.2 Uncertain Perceptron

Let us start from *Perceptron*, which is a simple type of artificial neural network. Perceptron is a classic model which constructs a linear classifier as shown in equation (5.1).

$$y = F(\sum_{i=1}^{n} x_i * \omega_i + \theta) \tag{5.1}$$

$$F(s) = \begin{cases} 1, & \text{s}\geq\text{0} \\ 0, & \text{s<0} \end{cases} \tag{5.2}$$

where, $x = (x_1, ..., x_n)$ is the input vector, $\omega = (\omega_1, ..., \omega_n)$ is the weight vector, F is the activation function, and y is the predicted class label by the perceptron. Now we revise the function of conventional perceptron and design a new uncertain perceptron classifier for uncertain datasets. Here, for the simple of illustration, we take a 2-dimensional uncertain dataset as an example. Assume the uncertain dataset

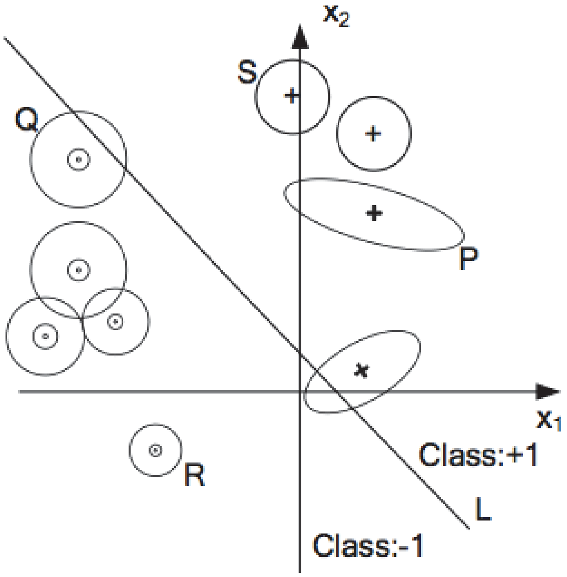


Figure 5.1. Geometric representation of uncertain perceptron

UD has two attributes $X = (X_1, X_2)$, and one class label Y. Each attribute is represented by the probability distribution function (pdf) of a Gaussian distribution as $X_i \sim N(\mu_i, \sigma_i)$, and the class label Y is selected from the set $\{+1, -1\}$. Figure 5.1 is a geometric representation of a linear classification for a 2-dimensional uncertain dataset. In this figure, every data instance is represented by an area instead of a point, because the value of each attribute is probabilistic, and we only have the knowledge about its distribution over this area. The straight line L in the figure represents the classification boundary, which corresponds to equation (5.3):

$$\sum_{i=1}^{2} \omega_i x_i + \theta = 0 \tag{5.3}$$

We define a parameter t as:

$$t = \sum_{i=1}^{2} \omega_i x_i + \theta \tag{5.4}$$

We assume that uncertain attributes X_1, X_2 follows the Gaussian distribution as $X_i = N(\mu_i, \sigma_i)$, independently. Then, t follows a distribution as:

$$f(t) \sim N(\sum_{i=1}^{2} \omega_i \mu_i + \theta, \sum_{i=1}^{2} \omega_i^2 \sigma_i^2) \tag{5.5}$$

From equation (5.5), we are able to track the propagation of data uncertainty and represent the linearly aggregated results by its pdf in this case. In conventional perceptron model, the value of t indicates the class label of the input data instance, referring to the the activation function in equation (5.2). However, in its uncertain version, the value of t is probabilistic, and its *pdf* is shown in equation (5.5). Therefore, we cannot assign a label to an instance based on the value of t any more, but instead, we define a new activation function as in equation (5.6).

$$F(s) = \begin{cases} +1, & \text{s} \geq 0.5 \\ -1, & \text{s} < 0.5 \end{cases} \tag{5.6}$$

where, $s = P(t > 0)$. If $P(t > 0) = 1$, t is definitely larger than 0, which means this instance is in class +1, and locates above the classification boundary L definitely, like Point P in figure 5.1. Similarly, if $P(t \geq 0) = 0$, it means the instance is in class -1, and it locates below L as Point R. If $P(t \geq 0) \in (0, 1)$, it means that this instance is possible to be in either class, and we assign its label by the class with larger possibility. In the binary classification problem, an instance is more likely to be in one class if and only if its probability of being in that class is larger than 0.5. Thus, we use the threshold 0.5 in our new activation function.

Figure 5.2 shows the structure of uncertain perceptron. Here, the input (μ_i, σ_i) is the mean and standard deviation of i^{th} Gaussian distributed uncertain attribute. The weighted sum t of these Gaussian distributed inputs is in the Gaussian distribution as $t \sim N(\mu_t, \sigma_t)$, and thus, we can compute the value of s as $s = P(t > 0)$.

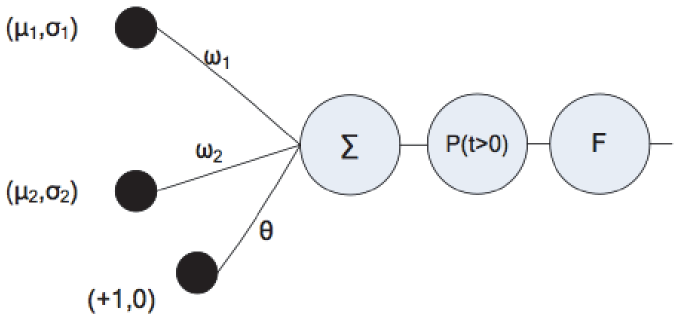


Figure 5.2. Structure of uncertain perceptron

5.3 Uncertain Multilayer Perceptron Neural Network

An uncertain multilayer perceptron neural network (UNN) is designed by adding a hidden layer of uncertain neurons between input and output layers. Figure 5.3 shows the layer structure of an example of uncertain neural networks. Here, the inputs are the Gaussian distributed uncertain attributes X_i, represented by their mean and standard deviation (μ_i, σ_i). In the hidden layer, its transfer function is defined as $F_H = p(T > 0)$, where $T = \sum \omega_i * Xi + \theta$, and $T \sim N(\sum_{i=1}^{2} \omega_i * \mu_i + \theta, \sum_{i=1}^{n} \omega_i^2 * \sigma_i^2)$. The output layer neurons uses Sigmoid function as the activation function F_O, and then the outputs are limited in the range $[0, 1]$, to model the membership in each class.

A straight-forward way to process the uncertain information is to use its mean value to represent the uncertain data points. Then, uncertain data have the same form as certain data, and traditional neural network can be trained as a classifier. We call this approach AVGNN (for averaging). This approach does not fully utilize uncertain information and may result in loss of accuracy in some cases. In figure 5.4, we use an example to show the principle of potential improvement in classifying accuracy of our UNN algorithm to this naïve AVGNN algorithm. Figure 5.4 shows an example of classifying an 2-d uncertain data set by our uncertain neural network classifier. Here, line L_1 and line L_2 reflect the training results in hidden layer of the

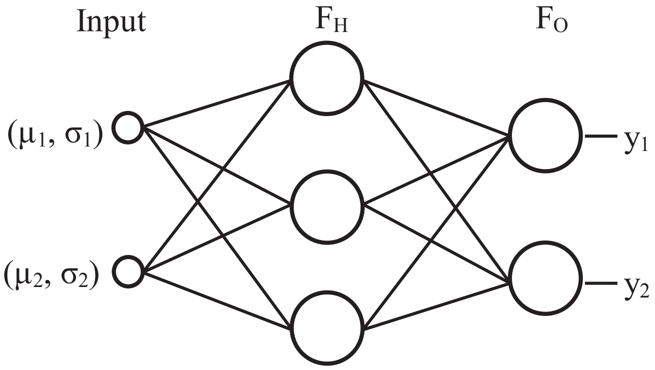


Figure 5.3. Multilayer neural network structure

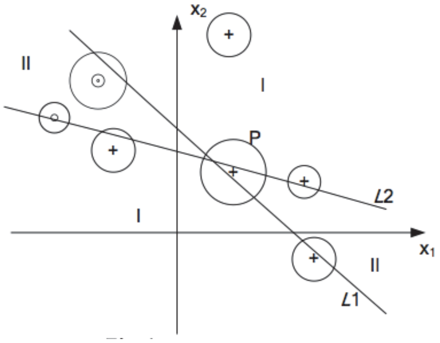


Figure 5.4. An example to show the improvement of classification accuracy

uncertain neural network. Suppose P is a test data point, and then we predicts the label of P by our classifier. Because the expectation of the possible positions for P locates in area II, it will be assigned to class II by AVG algorithm. However, if we take a look at the detail distribution of P in figure 5.4, we can see that P has a larger probability to be in area I than that in area II, and it is more reasonable that we assign it to class I. And in such a case, our algorithm UNN will classify test data point P to class I correctly, because it computes the probability of P belonging to both class I and class II by its probability distribution and then assign it to the class with larger probability. Therefore, UNN outperforms the naïve algorithm AVG in classification accuracy.

We adopt a Levenberg-Marquardt back propagation algorithm to train this supervised feed-forward neural network. It requires that all the activation functions have derivatives. Given the hidden layer activation function as in equation (5.7) , then, its derivative is computed in equation (5.8). And equation (5.8) can be computed by substituting two equations in (5.9) and (5.10).

$$F(\mu, \sigma) = P(T > 0) = \int_0^\infty \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(t-\mu)^2}{2\sigma^2})dt \tag{5.7}$$

$$\frac{\mathrm{d}F}{\mathrm{d}\omega_i} = \frac{\partial F}{\partial \mu} * \frac{\mathrm{d}\mu}{\mathrm{d}\omega_i} + \frac{\partial F}{\partial \sigma} * \frac{\mathrm{d}\sigma}{\mathrm{d}\omega_i} \tag{5.8}$$

$$\begin{aligned} \frac{\partial \mu_t}{\partial \omega_i} &= \mu_i \\ \frac{\partial \sigma_t^2}{\partial \omega_i} &= 2\sigma_i^2 \mu_i \end{aligned} \tag{5.9}$$

$$\frac{\mathrm{d}F}{\mathrm{d}(\mu, \sigma^2)} = (\frac{\partial F}{\partial \mu}, \frac{\partial F}{\partial \sigma^2}) = (\frac{1}{\sqrt{2\pi}} e^{-\frac{\mu^2}{2\sigma}}, -\frac{\mu}{2\sqrt{2\pi}\sigma^3} e^{-\frac{\mu^2}{2\sigma}}) \tag{5.10}$$

After we get the derivatives of these activation functions, we now can train the network by traditional gradient decent approach as Levenberg-Marquardt back propagation algorithm.

The activation function in the hidden layer of our uncertain neural network has its output between 0 and 1. When two different instances that are absolutely in the same class, both of their output of the activation function is 1. And the contribution of these two points in training the classifier are the same, no matter the difference of their positions to the decision boundary. This may cause the network training process be time consuming in some datasets. Therefore, we modify the hidden layer activation function to make it able to measure the distance between training points to the decision boundaries. In our uncertain data model, the data uncertainty is represented by Gaussian distribution, which is symmetric *w.r.t.* its mean. Thus, we design a new hidden layer activation function in equation (5.11) to accelerate the training process.

$$F_H(\mu, \sigma) = \begin{cases} \mu * P(T > 0) & , \mu > 0 \\ 0 & , \mu = 0 \\ \mu * P(T < 0) & , \mu < 0 \end{cases} \tag{5.11}$$

where, T is a Gaussian distributed random variable as $T \ N(\mu, \sigma)$. This new activation function actually contains two parts: $P(T > 0)/P(T < 0)$ is the probability that a data point locates in the up side or down side of the decision boundary and μ is the expected distance of the data point to the decision boundary. Next, we show that the new activation function F_n is differentiable. Equation (5.12) shows that F_n is continues at 0, because

$$\lim_{\mu \to 0+} F_n(\mu, \sigma) = \lim_{\mu \to 0-} F_n(\mu, \sigma) = F_n(0, \sigma) = 0 \tag{5.12}$$

And, we compute the derivative of F_n by substituting equation (5.13) to equation (5.10).

$$\frac{\mathrm{d}F_n}{\mathrm{d}(\mu, \sigma^2)} = (\frac{\partial F_n}{\partial \mu}, \frac{\partial F_n}{\partial \sigma^2}) \tag{5.13a}$$

$$\frac{\partial F_n}{\partial \mu} = \begin{cases} \eta + \frac{\mu}{\sqrt{2\pi}\sigma} e^{-\frac{\mu^2}{2\sigma}} & , \mu > 0 \\ 1/2 & , \mu = 0 \\ 1 - \eta - \frac{\mu}{\sqrt{2\pi}\sigma} e^{-\frac{\mu^2}{2\sigma}} & , \mu < 0 \end{cases} \tag{5.13b}$$

$$\frac{\partial F_n}{\partial \sigma^2} = -\frac{\mu^2}{2\sqrt{2\pi}\sigma^3} e^{-\frac{\mu^2}{2\sigma}} \tag{5.13c}$$

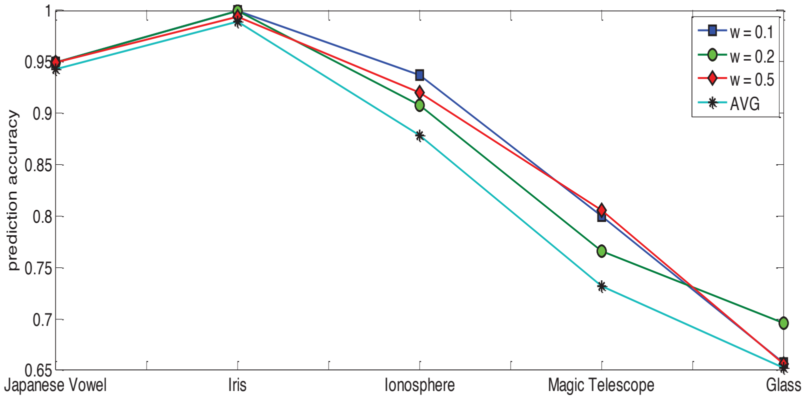


Figure 5.5. Prediction accuracy comparison of UNN and AVG

5.4 Experimental Results

5.4.1 Experiments on Accuracy

We implement our UNN approach in Matlab 6.5. We take five real datasets from UCI repository [59], which are Japanese Vowel, Iris, Ionosphere, Magic Telescope, Glass. For Japanese Vowel dataset, we directly use the raw data to estimate its Gaussian distribution; for all other datasets, we add a zero-mean Gaussian distribution to the original data, whose standard deviation equals to $\omega \cdot x$. Here, ω is the control parameter and x is the original data value. As a comparison of the performance, we also implement the AVG classifier which take the expectations of the uncertain attributes as the input. Table 5.2 shows the performances of both UNN and AVG in these datasets, and figure 5.5 compares the classification accuracies between our UNN algorithm and AVG classifier. UNN outperforms AVG in almost all the datasets. And in some dataset as Inosphere and Magic Telescope, UNN improves the accuracy by 6% to 7%. The reason is that UNN utilizes the detail distribution of uncertain data to estimate the membership in each class. The classification and prediction process is more sophisticated and comprehensive than AVG, and thus has the potential to achieve a better performance.

Table 5.2.

Experimental results of applying UNN and AVG classifiers to five uncertain datasets

Dataset	Uncertain Level	UNN		AVG	
		Train	Test	Train	Test
Japanese Vowel	estimate from raw data	98.50%	94.95%	99.17%	94.31%
Iris	$\omega_1 = 0.1$	98.05%	99.93%	99.17%	98.89%
	$\omega_2 = 0.2$	98.33%	99.93%		
	$\omega_3 = 0.5$	97.78%	98.89%		
Ionosphere	$\omega_1 = 0.1$	92.75%	93.71%	97.17%	87.86%
	$\omega_2 = 0.2$	94.50%	90.73%		
	$\omega_3 = 0.5$	99.13%	92.05%		
Magic Telescope	$\omega_1 = 0.1$	96.93%	80.01%	99.67%	73.17%
	$\omega_2 = 0.2$	97.50%	76.58%		
	$\omega_3 = 0.5$	97.50%	80.56%		
Glass	$\omega_1 = 0.1$	77.05%	65.75%	74.02%	65.22%
	$\omega_2 = 0.2$	76.00%	69.59%		
	$\omega_3 = 0.5$	79.02%	65.57%		

5.4.2 Experiments on Efficiency

As discussed above, an alternative activate function is designed to improve the efficiency of networks training process. Here we present an experiment which compares the efficiency of two networks with different hidden layer activate functions. In this experiment, we name the network using the original function (equation 5.7) as UNN-O and the networks using the modified activate function (equation 5.11) as UNN-M.

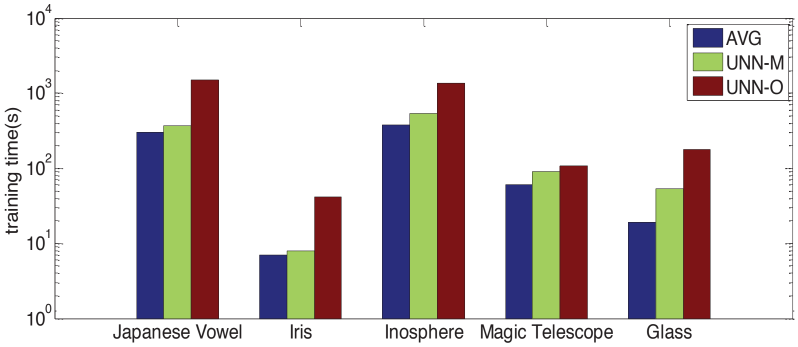


Figure 5.6. Compare the training time of UNN-O, UNN-M and AVG

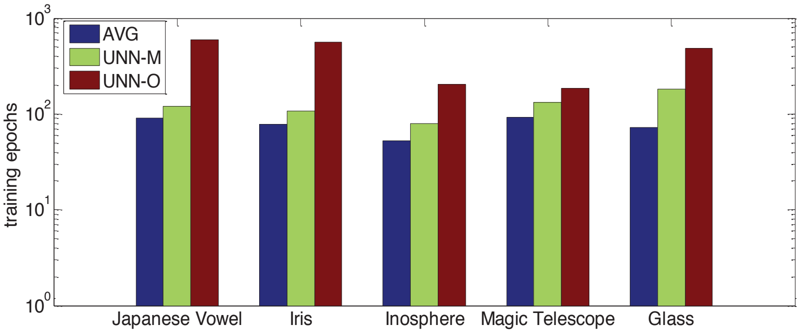


Figure 5.7. Compare the number of training epochs of UNN-O, UNN-M and AVG

The training times of UNN-O and UNN-M are shown in figure 5.6, and the number of training epochs are shown in figure 5.7. UNNs generally require more training time and epochs than AVG, because of the more complex computation in managing data uncertainty. However, compared to UNN-O, the figures also indicate that efficiency of UNN-M is significantly improved, which proves the effectiveness of using our new activation function.

5.5 Conclusion

In this section, we propose an uncertain neural network (UNN) model for classifying and predicting uncertain data. We use the probability distribution function to represent the uncertain data attribute in our data model, and redesign the neural network functions so that they can directly work on Gaussian distributed numeric uncertain attributes. Our extensive experiments proves that our UNN classifier has higher classification accuracy than the naïve approach. Since the usage of pdf may increase the computational complexity, we design a new activation function to improve the efficiency. As one of the supervised algorithms, UNN has the great potential to be a very powerful tool of classifying uncertain data.

6 A NAÏVE BAYESIAN CLASSIFIER IN CATEGORICAL UNCERTAIN STREAMS

6.1 Introduction

Data streams are widely used to model data in a lot of applications such as sensor networks, RFID networks, and network monitoring systems. Data uncertainty, which makes data imprecise or misleading, originates from many sources as data collection error, measurement precision limitations, sampling error, and transmission error. For example, in the traffic surveillance system, because we only record the state of a vehicle at the recording time, the exact state of the vehicle at any other time can only be inferred from data probabilistically. When the state of the vehicle is categorical like normal/abnormal, the uncertain state of the vehicle can be represented by a discrete pdf as normal:0.4, abnormal:0.6. A categorical attribute is an attribute with finite possible values, and a categorical uncertain attribute is a categorical attribute whose value is probabilistic. Data streams with categorical uncertain attributes are so called categorical uncertain data streams.

Data uncertainty has to be carefully managed; otherwise it would significantly downgrade the underlying performance of various data mining applications. A typical approach is to use the expectation of the probabilistic attribute to manage data uncertainty [60,61]. However, the expectation is only one of the statistic observations of uncertain data, and lots of useful information is lost if we simply use the expectation to model uncertain data. Some other methods adopt the possible world semantic [29, 62] to enumerate all possible databases to analyze uncertain data. Although these methods lead to an accurate result, it is usually too complex to be used in uncertain data stream mining, because of its exponential computational complexity.

We develop a new approach to manage data uncertainty to induce naïve Bayesian classifier in categorical uncertain data streams. We believe that data in one class are similar in some aspects, which is the reason why they are classified to the same class. In order to analyze the relation between the uncertain features of a data instance and its class label, we innovatively map uncertain attribute values to data points in the Euclidean space, where the coordinates of data points are transformed from vector-valued pdfs. Classification model is a function which maps the features into class labels. In previous uncertain classifiers [61, 63–65], uncertain attribute values are treated as random variables, and the classification model in uncertain data maps possible values of the uncertain feature into class labels. However, this approach has the exponential complexity, and various assumptions are made to make it practical. Meanwhile, the model is obscure to be understood because of its probabilistic intrinsics. However, by mapping the pdfs into data points in the Euclidean space, it helps to reveal the relationship between the space of pdfs and the space of class labels, and also helps to understand the classification rules by a new insight of the data uncertainty.

Data stream classification is very sensitive to both memory and computation cost, because data are coming continuously with a fast rate. Typically, the algorithm can only scan the data in one-pass, and this one-pass constraint dictates the choice of data structures and algorithms that can be used in data stream classification [66]. Meanwhile, managing pdf-represented uncertain data usually requires more computational resources, comparing to that in mining certain data. These new challenges bring the uncertain data stream mining problems up to the front recently.

In this chapter, we propose a novel algorithm to induce the naïve Bayesian classifier in categorical uncertain data streams. Our main contributions are listed as follows.

- We model the uncertain categorical data streams by mapping the pdfs of uncertain categorical attribute to data points in the Euclidean space, and estimate the density of points by the multi-dimensional kernel density estimator.

- We build a distance based and a density based naïve Bayesian model to classifying uncertain categorical data streams.
- We develop the new pre-binning technique to discrete the pdfs, which significantly improves the computational and space-efficiency.
- The experimental results in real world data streams prove the effectiveness and efficiency of our approach.

6.2 Related Works

Recently, many classical data mining algorithms are revised for uncertain data [29, 43,60,67]. Specifically, a lot of classifiers are extended to the uncertain versions such as uncertain decision tree classifier [63], uncertain SVM classifier [65] and uncertain rule-based classifier [62]. A naïve Bayesian classifier for uncertain data is proposed in [61], which represents the data uncertainty by continues random variable in either sample-based or formula-based probability distribution. It uses the expectation of the random variables to handle data uncertainty, which has the inherent weakness. The approach of mapping vector-valued pdf to data points in multi-dimensional Euclidean space is previously used in indexing uncertain data [68]. In our algorithms, we adopt this mapping to construct more sophisticated classification model to reveal relationship between pdfs and class labels.

Many uncertain data stream mining clustering algorithms are devised in recent years [69, 70]. And [71] introduced a new Gaussian mixture model for processing uncertain data streams. Though data streams classification has been well studied [66, 72], the uncertain data streams bring the classification problem back to the front, and our algorithms are propose to solve this new problem.

Table 6.1.
An example of a categorical uncertain data stream

Id	Color	Class
1	Red:0.2, Green:0.6, Blue:0.2	1
2	Red:0.6, Green:0.2, Blue:0.2	2
3	Red:0.2, Green:0.5, Blue:0.3	1
4	Red:0.5, Green:0.3, Blue:0.2	2
5	Red:0.7, Green:0.2, Blue:0.1	2
6	Red:0.3, Green:0.6, Blue:0.1	1
7	Red: 0.5, Green: 0.1,Blue: 0.4	2
8	Red: 0.5, Green: 0.2, Blue: 0.3	2
...	...	...

6.3 Problem Statement

An categorical uncertain attribute, denoted by A^U, is represented by its discrete pdf as $\langle a_1 : p_1, a_2 : p_2, , a_n : p_n\rangle$, where $\{a_1, a_2, \cdots, a_n\}$ is the set of all possible values of attribute A^U, and p_i is the probability that $A^U = a_i$. Table 6.1 shows an example of uncertain categorical data stream. Here *Color* is the uncertain attribute which has three possible values as $\{Red, Green$ and $Blue\}$. An uncertain instance is then represented by a vector valued pdf $\{Red : P_r, Green : P_g, Blue : P_b\}$.

We map a vector valued pdf $\langle a_1 : p_1, a_2 : p_2, , a_d : p_d\rangle$ of an uncertain categorical attribute A^U to a point in the d-dimensional Euclidean space whose coordinate is $\langle p_1, p_2, \ldots, p_d\rangle$. Fig. 6.1 shows the data points corresponding to uncertain instances in Table reftb:exmp. The distance between two data points measures the similarity of their corresponding vector valued pdfs.

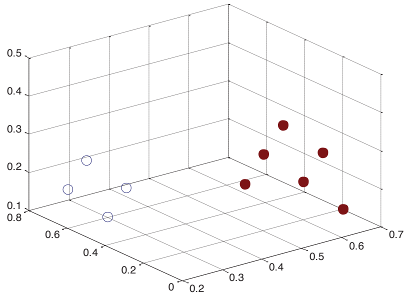


Figure 6.1. Mapping vector valued pdfs into Euclidean points

Mapping discrete pdfs to data points helps to manipulate categorical data uncertainty conveniently. First, we transfer probabilistic data instances to fixed points, which enables the directly use of traditional data mining techniques. For example, we can adopt a multi-variable kernel density estimator to estimate the density distribution of pdfs. Second, we can directly use pdfs as the input, instead of the probabilistic attribute values. This property helps train a classification model to reveal the relations between pdfs and class labels. Third, we are now able to obtain an intuitive understanding of the data model in uncertain data.

We incorporate the new uncertainty management into naïve Bayesian classification model to design a classifier for uncertain data streams. The naïve Bayesian classification model is shown in Equation (6.1).

$$P(c_i|X) = \frac{\prod_{j=1}^{n} P(x_j|C_i)}{P(X)} * P(C_i) \tag{6.1}$$

Where, $X = \langle x_1, x_2, \cdots, x_n \rangle$ is a test instance, C_i is a class label. Here x_j is a categorical uncertain attribute. The posterior probability $P(C_i|X)$ indicates the membership of X in the class C_i, and X is assigned to the class with maximal membership. We usually use Equation (6.2) instead of Equation (6.1) in classification, because P(X) is a constant.

$$P(C_i|X) = \prod_{j=1}^{n} P(x_j|C_i) * P(C_i) \tag{6.2}$$

In traditional certain databases, the likelihood $P(X|C_i)$ is estimated by the frequency of event X in class C_i. However, we cannot count the frequency of probabilistic attribute values in uncertain data. Therefore, by mapping vector valued pdfs to Euclidean points, we cab measure $P(X|C_i)$ by the density at point X, and estimate it from uncertain data instances in class C_i.

A naïve approach is to use the mean m_i of all data points in C_i to represent the overall density distribution in class C_i so that $P(X|C_i)$ is measured by the distance between m_i and X. A more sophisticated approach is to estimate the density distribution from data, and the density at X is used to measure $P(X|C_i)$. Data stream usually requires one-pass scan in building classification models. And it is very important to train the model with bounded memory and computation cost in uncertain data stream classification.

6.4 Naïve Bayesian Classifiers in Uncertain Streams

In this section, we propose two approaches to induce naïve Bayesian classifier in categorical uncertain data streams.

6.4.1 A Distance Based Approach

As attributes are assumed to be independent in naïve Bayesian model, we first analyze the computation of posterior probability for one attribute. In uncertain case, a straightforward extension to the traditional approach is to define a new point P_i to represent the data distribution in the class C_i, and calculate $P(X|C_i)$ by the distance between X and P_i.

Suppose A^U is an uncertain categorical attribute with d possible values, we define point P_i as the closest point to all the n observed points in C_i. Let $P_i = (P_{i_1}, P_{i_2}, \ldots, P_{i_d})$, then it minimizes the d-dimensional Euclidean distance in Equation (6.3).

$$\arg\min_{P_i} \sum_{j=1}^{n} \sqrt{(p_{j_1} - P_{i_1})^2 + \cdots + (p_{j_d} - P_{i_d})^2} \tag{6.3}$$

Where, $p_j = (p_{j_1}, p_{j_2}, \cdots, p_{j_d})$ are the n data points in C_i. By solving Equation (6.3), the coordinate of P_i is computed in Equation (6.4).

$$P_i = \frac{\sum_{j=1}^{n} p_j}{n} \tag{6.4}$$

Where n is the number of instances in C_i. We can see that P_i is the mean of data observations in C_i, the point P_i is also called the *center* of the class C_i. The uncertain attribute $(a_1 : p_1, a_2 : p_2, \ldots, a_d : p_d)$ of the test instance t is mapped to the data point $p_t = (p_1, p_2, \ldots, p_d)$. The distance between p_t and P_i reflects the membership of p_t belonging to class C_i. Here we use *dot product* to measure the similarity between two discrete pdfs [6], and then $P(X|C_i)$ is computed by Equation (6.5).

$$P(X|C_i) = X \cdot P_i \tag{6.5}$$

In data stream classification, when a new element $p_{(n+1)}$ comes, the position of P_i for each categorical uncertain attribute can be updated incrementally by Equation (6.6).

$$\begin{aligned} P_i^{n+1} &= \frac{\sum_{i=1}^{n+1} p_i}{n+1} = \frac{\frac{n}{n+1} * \sum_{i=1}^{n} p_i}{n} + \frac{p_{(n+1)}}{n+1} \\ &= \frac{n}{n+1} P_i^n + \frac{p_{(n+1)}}{n+1} \end{aligned} \tag{6.6}$$

We substitute Equation (6.6) into Equation (6.2) to induce our distance-based naïve Bayesian classifier for uncertain data streams.

The distance-based approach is simple and fast. We only need to maintain a center point for each attribute in every class, and it cost no additional memory to handle endless incoming data streams. The trade-off of this simplicity is that we assume the density distribution has only one mode, which locates at the center point. However, in most cases, the density distribution is much more complex, and cannot be represented by a single point. Meanwhile, the center point position is sensitive to outliers.

6.4.2 A Density Based Approach

In this section, we introduce a density-based approach to induce naïve Bayesian classifier in uncertain streams. In each class, we employ a multi-variable kernel density estimator to estimate the density distribution, which is shown in Equation (8).

$$\hat{f}_h(\boldsymbol{x}) = \frac{1}{n}\frac{1}{h}\sum_{i=1}^{n} K(\frac{x_1 - X_{i_1}}{h_1}, \ldots, \frac{x_d - X_{i_d}}{h_d}) \tag{6.7}$$

Where, $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ is a data point in the d-dimensional space, $X_i = (X_{i_1}, X_{i_2}, \ldots, X_{i_d})$ are n training points, $h = (h_1, h_2, \ldots, h_d)^T$ is the bandwidth matrix and $K(\mu) = K(\mu_1, , \mu_d)$ is the multidimensional kernel function. $K(\mu)$is usually approximated by the multiplicative kernel[14], as shown in Equation (6.8).

$$K(\mu) = \prod_{i=1}^{d} k(\mu_i) \tag{6.8}$$

Where k is a uni-variable kernel function, and then the density at $x = (x_1, \ldots, x_d)$ in Equation (6.8) can be approximated by Equation (6.9).

$$\hat{f}_h(x_1, \ldots, x_d) = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{d} k(\frac{x_j - X_{i_j}}{h_j}) \tag{6.9}$$

Here, k is the 1-dimensional kernel function, and the bandwidth h_j is estimated from the observations in each dimension. However, the kernel density estimator in Equation (6.9) is not efficient enough for mining data streams, because its computation and memory cost is growing with the incoming data. Therefore, we design a pre-binning approach to improve the efficiency of kernel estimation in data streams.

As the Euclidean data points are mapped from vectored pdfs in our model, the cooperate values of all the data points are bounded in the range of $[0, 1]$. Thus, it is practical and reasonable to discrete them into equal-width bins. For example, if the precision of pdf is measured in unit 0.01, then there is no information lost if we divide the partition [0, 1] into 100 bins equally. By pre-binning the incoming pdfs, we significantly reduce the computation and memory consumption in kernel density estimation. Suppose the range $[0, 1]$ in each dimension of the space is equally divided into k bins, then the entire space is divided into k^d cubes with unit size $1/k$. Here, each cube is represented by its center point. We maintain a kernel table in memory to record historical kernel points. For example, we first discrete the probability values in Table 6.1 into five bins, and then insert the pre-binned vectored pdfs into the kernel table, which is shown in Table 6.2.

Here the partition [0,1] is divided into five bins: $b_1 = [0, 0.2]$, $b_2 = (0.2, 0.4]$, $b_3 = (0.4, 0.6]$, $b_4 = (0.6, 0.8]$ and $b_5 = (0.8, 1.0]$. The value of a bin is represented by its center point. For example, the value of b_1 is 0.1. We can see that the data instances with $Id = 7$ and $Id = 8$ in Table 6.1 are grouped to one kernel entry in Table 6.2, because they are identical after binning. The size of the kernel table is constant with the number of incoming data, which is at most 5^3 in this example.

The kernel table is updated, when comes a new training instance. Suppose A_t^U is a pre-binned uncertain attribute belonging to class C_i, then if there exists an entry in the kernel table which is identical with A_t^U, we increase the count of that kernel in class C_i by one; otherwise, we add a new entry of the kenerl A_t^U, and initialize its number in C_i as 1.

Now we can revise the kernel density estimator in Equation (6.10) by using the pre-binning technique.

$$\hat{f}_h(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{S} n_i * \prod_{j=1}^{d} \frac{1}{h_j} k(\frac{x_j - B_{i_j}}{h_j}) \tag{6.10}$$

Where N is the number of training instances, S is the number of entries in kernel table, and n_i is the number of i^{th} kernel in the kernel table in this class. d is the number of possible values for the attribute, h_j is the bandwidth of the j^{th} kernel function, and B_{i_j} is the pre-binned value of i^{th} kernel in j^{th} dimension.

Table 6.2.
An example of a kernel table

Red	Green	Blue	Class Count (N_{C_1}, N_{C_2})
0.1	0.5	0.1	$(1,0)$
0.5	0.1	0.1	$(0,1)$
0.1	0.5	0.3	$(1,0)$
0.5	0.3	0.3	$(0,1)$
0.7	0.1	0.1	$(0,1)$
0.3	0.5	0.1	$(1,1)$
0.5	0.1	0.3	$(0,2)$
0.5	0.1	0.5	$(0,1)$

The bandwidth h_j can also be estimated from the kernel table by the plug-in method in Equation (6.11).

$$h_j = 0.9 * A * N^{-\frac{1}{5}} \tag{6.11}$$

Where we have

$$A = min(\frac{IQR}{1.34}, \sigma)$$

Here IQR is the inter-quartile and σ is the standard deviation in one dimension. Suppose there are totally N_j entries in j^{th} dimension and each bin has n_i duplicated kernels, then the mean μ_j and standard deviation σ can be estimated by Equation (6.12).

$$\begin{aligned}\mu_j &= \frac{\sum_{i=1}^{N_j} n_i * b_{ij}}{\sum_{i=1}^{N_j} n_i} \\ \sigma^2 &= \frac{1}{\sum_{i=1}^{N_j} n_i - 1} * \sum_{i=1}^{N_j} (b_{ij} - \mu_j)^2\end{aligned} \tag{6.12}$$

Algorithm 6.1 shows the method to estimate parameter σ in one class C from kernel table T. We first estimate the mean μ by one scan of the entries in the kernel table, and then use μ to compute the standard deviation σ.

Similarly, we can estimate $IQR = Q_3 - Q_1$ from the kernel table. Q_3 is the 75% percentile quartile so that 75% of the values are smaller than Q_3; and Q_1 is the 25% percentile quartile. In each Euclidean dimension of an uncertain attribute, we first select out the entries belonging to class C and sort it by the entries' values. Then, we can directly select the Q_3 and Q by one scanning of the sorted entries. The details of IQR estimation are shown in Algorithm 6.2.

Now we can incrementally estimate the density distribution in uncertain data streams. When new training data come, we update the kernel table to estimate the density and compute $P(X|C_i)$, which is used to calculate the membership of class C_i for any test point X. X is classified to the class with the maximal membership. Algorithm 6.3 shows our density based algorithm to induce naïve Bayesian classifier in categorical uncertain data streams.

6.5 Experiments

6.5.1 Setup

We use five real datasets, which are listed in Table 6.3, from UCI repository in our experiments. For all datasets except LED, we add synthetic data uncertainty to raw data by the approach in [8]. Suppose a categorical attribute A in original dataset D has n possible values, then the generated uncertain attribute A^U has the discrete pdf $\langle a_1 : p_1, a_2 : p_2, \ldots, a_n : p_n \rangle$. The original attribute value a_k is defined as

Algorithm 6.1: Estimating σ in class C from kernel table

```
Input: $T$: kernel Table, $N$: number of entries in $T$
Output: $\sigma$
$\boldsymbol{\mu} \leftarrow 0$
$s \leftarrow 0$
foreach kernel entry $\boldsymbol{e} \in T$ do
    if $\boldsymbol{e}.N_c > 0$ then
        $\boldsymbol{e.k} = \langle \boldsymbol{e}.k_1, \ldots, \boldsymbol{e}.k_d \rangle$
        $\boldsymbol{\mu} = (\boldsymbol{\mu} * s + \boldsymbol{e.k} * e.N_c)/(s + \boldsymbol{e}.N_c)$
        $s = s + \boldsymbol{e}.N_c$
    end
end
$\boldsymbol{\sigma}^2 \leftarrow 0$
foreach kernel entry $\boldsymbol{e} \in T$ do
    if $\boldsymbol{e.N_c} > 0$ then
        $\boldsymbol{e.k} = \langle \boldsymbol{e}.k_1, \ldots, \boldsymbol{e}.k_d \rangle$
        $\boldsymbol{\sigma}^2 = \boldsymbol{\sigma}^2 + (\boldsymbol{e.k} - \boldsymbol{\mu})^2$
    end
end
return $\sqrt{\boldsymbol{\sigma}^2/(s-1)}$
```

the *Main Value*, which is associated with the probability p_m. Here p_m is drawn from a normal distribution $N \sim (\mu, 0.1)$, where μ is a parameter to control the uncertain level and its value is selected randomly from $\{0.6, 0.7, 0.8\}$. All other possible values of A^U except the Main value a_k are assigned probabilities p_i so that they satisfy the constraint $\sum_{i=1}^{n} p_i = 1$. For missing attribute values, it is reasonable to assign an equal probability to every possible value in its discrete pdf. Meanwhile, we construct a noisy dataset for comparison purpose by drawing one sample from each pdf-represented uncertain attribute value.

Algorithm 6.2: Estimating IQR in class C from kernel table

```
Input: $T$: kernel Table, $N$: number of entries in $T$
Output: $IQR$
$q_1 \leftarrow 0.25 * N$
$q_3 \leftarrow 0.75 * N$
foreach dimension $d_i$ do
    $v = \phi$
    foreach kernel entry $\boldsymbol{e} \in T$ do
        if $\boldsymbol{e}.N_C > 0$ then
            $v = v \cup \langle \boldsymbol{e}.k_i, \boldsymbol{e}.N_C \rangle$
        end
    end
    sort($v$) by $v.k_i$
    $i \leftarrow 0$, count $\leftarrow 0$
    while $i < v.size$ do
        count $=$ count $+ v.N_C$
        if $count \geq q_1$ then
            $Q_1 = v.k_i$
        end
        if $count \geq q_3$ then
            $Q_3 = v.k_i$
            break;
        end
        $i = i + 1$
    end
    $IQR_i = Q_3 - Q_1$
end
return $IQR = \langle IQR_1, \ldots, IQR_d \rangle$
```

The original dataset LED is inherently imprecise [73]. Thus, we generate the uncertain data stream by aggregating the original data. Every 100 instances of the

Algorithm 6.3: The density based naïve Bayesian classifier for categorical uncertain streams

```
Input: A chunk of buffered data $D = D_1 \cup D_2$
       $D_1$:training set, $D_2$: testing set
Output: predicted labels of $D_2$
$B \leftarrow$ binning $D_1$ into $k$ bins
update Kernel table $T$
estimate parameters $\sigma$ and $IQR$ to select bandwidth matrix $h$
$L \leftarrow \phi$
foreach instance $t \in D_2$ do
    foreach attribute $a^u \in t$, $a^u = \{a_1 : p_1, \ldots, a_n : p_n\}$ do
        $t \leftarrow \text{bin}(p_1, \ldots, p_n)$
        foreach class $C_i$ do
            estimate $P(C_i|t.a^u)$ by the density $P(t.a^u|C_i)$
        end
    end
    compute the posterior probability $P(C|t) = \prod_{a^u} P(C_i|t.a^u)$
    $t.C = \arg\max_C P(C|t)$
    $L = L \cup \{t.C\}$
end
return $L$
```

original dataset are grouped as one uncertain instance. and the probabilities of the uncertain attributes are measured by their frequencies. For example, suppose one attribute A in class C_i has two possible values a_1, a_2 and there are t instances in class C_i among all 100 instances. If the frequency of a_1 in these t instances is x, and the frequency of a_2 is y, then we estimate the probability distribution of A as $\{a_1 : x/t, a_2 : y/t\}$. An uncertain instance is generated by repeating this process in all attributes.

Table 6.3.
Datasets used in experiments

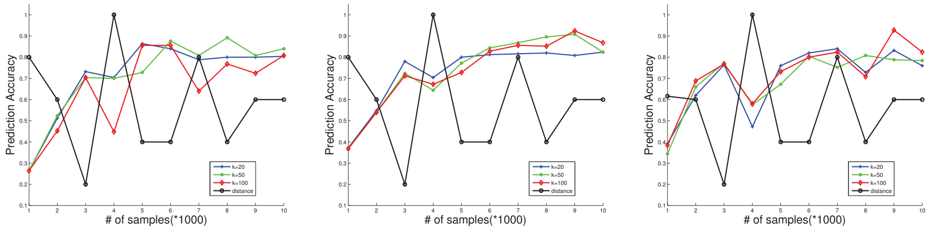
DataSet	# of instance	# of categorical attribute	missing values
Credit	1000	13	N/A
Chess	3196	36	N/A
Voting	435	16	Yes
Mushroom	8124	22	Yes
Led	1 000 000	7	N/A

We repeatedly load the categorical uncertain datasets to simulate the uncertain data streams. In our experiment system, an input buffer is used to save 200 incoming data instances. The buffered data are equally divided into four folds. one is randomly selected for testing; while other folds are used as new training data samples.

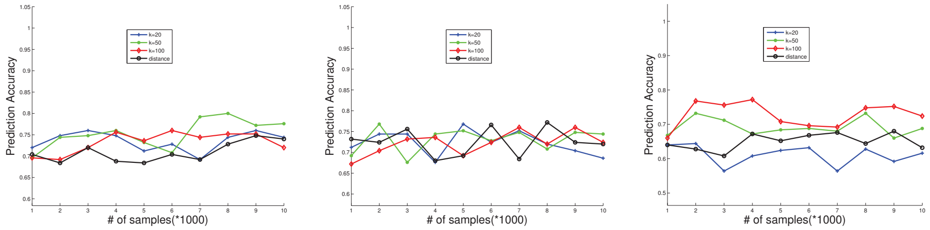
We implement both the distance-based and the density-based classifiers in our experiments of classifying categorical uncertain data streams. We also implement a batch-mode classical naïve Bayesian classifier in the sampled noisy datasets, for the purpose of comparison. For each dataset except LED, we set different uncertain levels by varying the value of μ. In density-based approach, we select different number of bins from $\{20, 50, 100\}$ in pre-binning process. We analyze the performances of our algorithms to verify the following advantages: (1) prediction accuracy; (2) effect of pre-binning technique; (3) memory efficiency.

6.5.2 Results

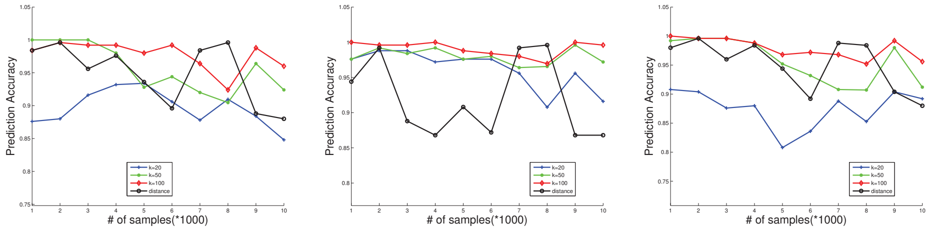
Table 6.4 compares the averaged prediction accuracies in the five uncertain data streams. Our density-based naïve Bayesian classifier outperforms other two methods in most cases. The prediction accuracy is improved when we select a proper value of



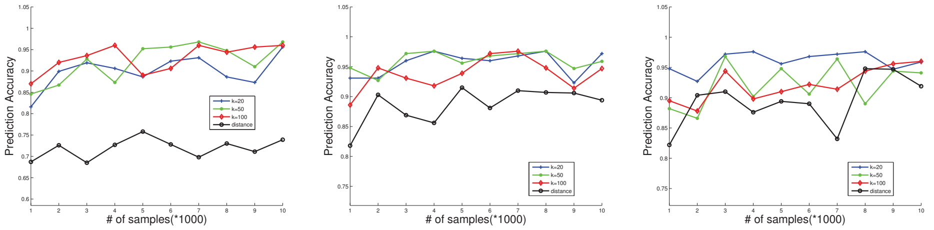
(a) chess, $\mu = 0.6$ (b) chess, $\mu = 0.7$ (c) chess,$\mu = 0.8$



(d) credit,$\mu = 0.6$ (e) credit,$\mu = 0.7$ (f) credit,$\mu = 0.8$



(g) mushroom,$\mu = 0.6$ (h) mushroom,$\mu = 0.7$ (i) mushroom,$\mu = 0.8$



(j) voting,$\mu = 0.6$ (k) voting,$\mu = 0.7$ (l) voting,$\mu = 0.8$

Figure 6.2. Comparing prediction accuracy under different uncertain levels in four data streams

k, which is the number of bins in pre-binning technique. k corresponds to the round-off error in discretization, and reflects the trade-off between efficiency and accuracy.

Table 6.4.
Classification accuracies in uncertain data streams

Data-streams	μ	Density based			Distance-based	Traditional-NB
		$k = 20$	$k = 50$	$k = 100$		
chess	0.6	0.806 ± 0.017	0.845 ± 0.038	0.760 ± 0.072	0.58 ± 0.220	0.597
	0.7	0.816 ± 0.006	0.868 ± 0.031	0.866 ± 0.032	0.60 ± 0.140	0.551
	0.8	0.796 ± 0.044	0.787 ± 0.019	0.817 ± 0.070	0.56 ± 0.150	0.482
credit	0.6	0.733 ± 0.023	0.770 ± 0.032	0.746 ± 0.014	0.723 ± 0.022	0.682
	0.7	0.711 ± 0.032	0.735 ± 0.015	0.738 ± 0.018	0.701 ± 0.040	0.573
	0.8	$0.606 \pm 0.025^*$	0.689 ± 0.023	0.722 ± 0.025	0.660 ± 0.018	0.431
mushroom	0.6	0.902 ± 0.019	0.932 ± 0.020	0.969 ± 0.024	0.910 ± 0.044	0.743
	0.7	0.954 ± 0.024	0.976 ± 0.011	0.984 ± 0.009	0.927 ± 0.056	0.695
	0.8	$0.874 \pm 0.026^*$	0.927 ± 0.028	0.968 ± 0.014	0.926 ± 0.046	0.477
voting	0.6	0.914 ± 0.030	0.950 ± 0.021	0.945 ± 0.020	0.721 ± 0.014	0.773
	0.7	0.959 ± 0.019	0.964 ± 0.010	0.951 ± 0.022	0.899 ± 0.010	0.682
	0.8	0.964 ± 0.010	0.929 ± 0.026	0.939 ± 0.018	0.907 ± 0.043	0.429
LED	-	0.956 ± 0.027	0.995 ± 0.011	0.996 ± 0.018	0.901 ± 0.051	0.601

For example, in data stream *mushroom*, the density-based classifier has the highest accuracy when $k = 100$. The resolution is higher when k is set to a large value, which helps distinguish clustered Euclidean points. This also explains the reason why the density-based classifier has the slightly lower accuracy in *credit* and *mushroom* than other two approaches, when $k = 20$. However, in the data stream *chess*, it achieves the best performance when $k = 20$. The reason is that the smaller k value helps reduce the influence of outliers.

The original LED dataset contains inherent error, and its optimal classification accuracy is proved to be 74% [73]. However, by pre-aggregating the original LED dataset into the categorical uncertain data stream, our density-based classifier achieves the accuracy 95.6% $\sim$ 99.6%, which outperforms the traditional naïve Bayesian classifier and the distance-based approach. This proves that our uncertain management of mapping pdfs into Euclidean points is effective in classifying uncertain data streams.

By comparing to the performance of traditional naïve Bayesian algorithm in Table 6.4, we can see that our density-based classifier is robust to data uncertainty. The accuracy of traditional naïve Bayesian classifier drops dramatically with the increment of noise; while our density-based approach has relatively similar performance under different uncertain levels.

Fig. 6.2 compares the performance between our density-based classifier and distance-based algorithm under different uncertain levels in the four data streams. Here each marked point is the prediction accuracy in one iteration, which processes five frames of buffed data. We can see that the density-based approach has a more smooth performance than the distance-based method. The first reason is that the distance-based approach is more sensitive to the influence of outliers. Second, the distance-based approach can only work in the single mode data streams; while the density-based method does not have this constraint and can effectively classify data streams in arbitrary shapes with accumulative training instances. This is also the reason why the density based classifier usually has smaller standard deviation of its prediction accuracy.

Fig. 6.3 shows the kernel table size in classifying data stream LED. The data stream contains 200 000 uncertain instances in total. In Fig. 6.3, the kernel table size quickly arrives to almost a constant in every setting of the bin number. Because of the pre-binning technique, we consume bounded memory to build the kernel table for the incoming data. In practice, if we set $k = 100$, then it only needs less than 1MB memory to classify the uncertain stream.

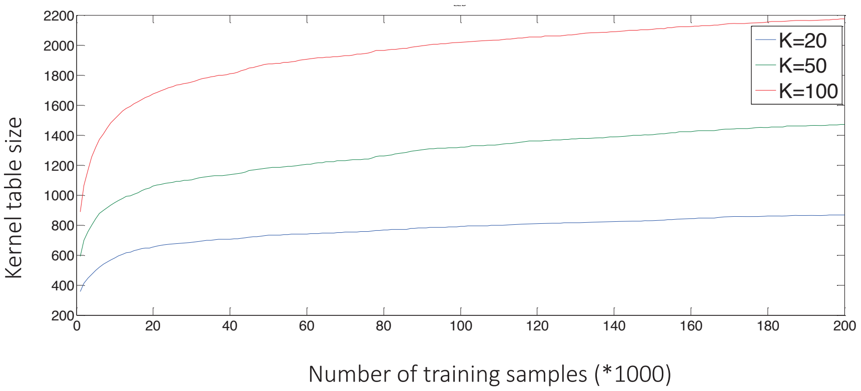


Figure 6.3. Size of kernel tables in classifying LED data stream

6.6 Conclusion

In this paper, we propose a new approach to construct naïve Bayesian classifier for uncertain categorical data streams. We map the vectored pdfs of uncertain categorical attribute into points in the multi-dimensional Euclidean space to estimate the distribution of pdf inputs,which is used to induce naïve Bayesian classifier. We prebin the discrete pdf to guarantee the bounded computation and memory efficiency in classifying uncertain data streams.Experiments proved the outstanding performance of our methods. In the future, we will continue to develop data mining applications in uncertain stream minings.

7 DISCRETIZATION IN UNCERTAIN DATABASES

7.1 Introduction

Data discretization is a commonly used data pre-processing technique in data mining. Data discretization reduces the number of values for a given continues attribute by dividing the range of the attribute value into several intervals. Interval labels are then used to replace the actual data values. Replacing numerous values of a continues attribute by a small number of interval labels thereby simplifies the original data. This leads to a concise, easy-to-use, knowledge-level representation of mining results. Discretization is often performed prior to the data mining process and usually considered as the pre-processing step in data mining and knowledge discovering. For example, many classification algorithms as AQ [74], CLIP [75], and CN2 [76] are only designed for category data, and numerical data are usually first discretized before being processed by these classification algorithms. Assume A is one of the continues attributes of a dataset, A can be discretized into n intervals as $D = \{[d_0, d_1), [d_1, d_2), ..., [d_{n-1}, d_n]\}$, where d_i is the value of the endpoints for each interval. Then, D is called as a discretization scheme on attribute A. A good discretization algorithm not only produces a concise view of continues attributes so that experts and users can have a better understanding of the data, but also helps machining learning and data mining applications to be more effective and efficient.

A number of discretization algorithms have been proposed in literature these year [77], most of them works only on traditional certain data. However, data tends to be uncertain in many applications [43]. Data uncertainty can downgrade the performance of various data mining algorithms if it is not well processed. It is widely accepted that we use a random variable with probability distribution to model data uncertainty.

Thus, uncertain attribute value is often represented by an interval associated with a pdf over it [43, 67].

In this chapter, we propose a data discretization technique call Uncertain Class-Attribute Interdependency Maximization (UCAIM) for uncertain data. It is based on traditional CAIM discretization algorithm, and we extend it with a new mechanism to process uncertainty. Probability distribution function is commonly used to mired data uncertainty and pdf can be represented in a formula based or sample based form. We adopt the concept of probability cardinality to build the quanta matrix for uncertain data. Based on the quanta matrix, we define a new criterion value *ucaim* to measure the interdependency between uncertain attributes and uncertain class memberships. The optimal discretization scheme is determined by search the one with the largest *ucaim* value.

7.2 Related Works

Discretization algorithms can be divided into top-down and bottom-up methods according to how the algorithms generate discrete schemes [78]. Both top-down and bottom-up discretization algorithms can be further subdivided into unsupervised and supervised methods [79]. Equal width and equal frequency are the well-known unsupervised top-down algorithms, while the supervised top-down algorithms include MDLP [80], CADD(class-attribute dependent discretize algorithm) [81], information entropy maximization [82], CAIM (class-attribute interdependent maximization algorithm) [83] and FCAIM (fast class-attribute interdependent maximization algorithm) [84]. Since CAIM selects the optimal discretization scheme that has the highest interdependence between target class and discretized attributes, it is proved to be superior to other top-down discretization algorithms in helping the classifiers to achieve high classification accuracy. FCAIM extends CAIM by using a different strategy to select fewer boundary points during the initialization, which speeds up the process of finding the optimal discretization scheme.

In the bottom-up category, there are widely used algorithms such as ChiMerge [85], Chi [86], Modified Chi2 [87] and Extended Chi2 [88]. Bottom-up method starts with the complete list of all continues values of the attribute as cut-points, so its computational complexity is usually higher than the top-down method. Algorithms like ChiMerge requires users to provide some parameters such as significant level and minimal/maximal interval numbers during the discretization process. [89] illustrates that all these different supervised discretization algorithms can be viewed as assigning different parameters to a unified goodness function, which can be used to evaluate the quality of discretization algorithms. There also exist some dynamic discretization algorithms [64] which are designed for particular machine learning algorithms such as decision tree and naïve Bayesian classifier.

All the algorithms mentioned above are based on traditional certain datasets. To the best of our knowledge, no discretization algorithm has been proposed for uncertain data that are represented by pdfs. In the recent years, there have been growing interests in uncertain data mining. And a number of classic classification algorithms are extended to process uncertain datasets, as uncertain decision tree [63], uncertain naïve Bayesian classifier [61] and so on. Therefore, it is extremely important that data preprocessing techniques like discretization properly handle this kind of uncertainty as well.

7.3 Problem Statement

In our uncertain data model, we first define the uncertain attribute, denoted as A^{u_n}, as the probabilistic value of a numerical attribute A. In uncertain dataset D, each tuple t_i is associated with a feature vector $V_i = (f_{i,1}, f_{i,2}, ..., f_{i,k})$ to model its uncertain attributes. Here, $f_{i,j}$ is the pdf represent the uncertainty attribute $A^{u_n}_{i,j}$ in tuple t_i. Meanwhile, a pdf C_i is assigned as tuple t_i's uncertain class label as class membership.

Table 7.1.
An example of uncertain dataset

ID	Class type	Attribute 1	Attribute 2
1	T:0.3, F:0.7	(105, 5)	(100:0.3, 104:0.6, 110:0.1)
2	T:0.4, F:0.6	(110, 10)	(102:0.2, 109:0.8)
3	T:0.1, F:0.9	(70, 10)	(66:0.4, 72:0.4, 88:0.2)

In practical applications, uncertainties are usually modeled in forms of Gaussian distributions, and parameters such as mean μ and standard deviation σ are used to model the Gaussian distributed uncertainty. In such a case, uncertain attribute $A_{i,j}^{u_n}$ has a formula based probability as $A_{i,j}^{u_n} \sim N(\mu, \sigma)$. In case that the pdf of data uncertainty has no closed form, a sample based method is used to represent the pdf as $\{A_{i,j}^{u_n} | (x_1 : p_1), (x_2 : p_2), ..., (x_n : p_n)\}$, where $X = \{x_1, x_2, ..., x_n\}$ is the set of all possible values of attribute $A_{i,j}^{u_n}$, and p_i is the probability that $A_{i,j}^{u_n} = x_i$.

Not only can the attributes be uncertain, class labels may also contain uncertainty. Instead of having the accurate class label, a class membership may be a pdf as $C_i = \{c | (c_1 : p_1), (c_2, p_2), ..., (c_n, p_n)\}$, where $C = \{c_1, c_2, ..., c_n\}$ is the set containing all possible class labels, and p_i is the probability that this instance t_i belongs to class c_i. Table 7.1 shows an example of an uncertain dataset. Both attributes and class labels of the dataset are uncertain. Their precise values are unavailable and we only have knowledge about the pdf. For attribute 1, its uncertainty is represented buy a Gaussian distribution with parameters (μ, σ). For attribute 2, it lists all possible values with their corresponding probabilities for each instance. Note that the uncertainty of class label in our model is always represented in the sample based format as the values are discrete.

7.4 UCAIM Algorithm

7.4.1 Cardinality Count For Uncertain Data

According to the uncertain model, an uncertain attribute $A_{i,j}^{u_n}$ is associated with a pdf either in a formula based or sample based format. If it is in a formula based format, then we can compute the probability that the value of $A_{i,j}^{u_n}$ falls in range $[left, right]$ by equation 7.1; otherwise, we compute this probability by equation (7.2).

$$P(A_{i,j}^{u_n} \in [left, right]) = \int_{left}^{right} f_A(x)dx \tag{7.1}$$

where, $f_A(x)$ is the probability density function of uncertain attribute $A_{i,j}^{u_n}$.

$$P(A_{i,j}^{u_n} \in [left, right]) = \sum_{x_k \in [left, right]} p_k \tag{7.2}$$

where,x_k is the possible value of $A_{i,j}^{u_n}$, and p_k is the probability that $A_{i,j}^{u_n} = x_k$.

If the uncertain class label's distribution is independent with the pdf of uncertain attribute value, then, given a tuple t_i, the joint probability that its value falls in range $[left, right]$ and it is assign a class label c_i can be calculated in equation (7.3).

$$P(A_{i,j}^{u_n} \in [left, right], C_t = c_i) = P(A_{i,j}^{u_n} \in [left, right]) * P(C_t = c_i) \tag{7.3}$$

Given an partition as $[left, right]$, for all the tuples in class c_i, we compute the sum of the probabilities that the value of its uncertain attribute $A_{i,j}^{u_n}$ is in the range $[left, right]$. This summation is called probability cardinality. For example, the probability cardinality of partition $P = [a, b)$ for class c_i is calculated in equation (7.4).

$$P_{c_i} = \sum_{i=1}^{n} P(A_{i,j}^{u_n} \in [a, b)) * P(C = c_i) \tag{7.4}$$

Probability cardinalities provide us valuable insight during the discretization process and it is used to build quanta matrix for uncertain data.

Table 7.2.
Quanta matrix and discretization scheme

class	Intervals					Class Total
	$[d_0, d_1)$	...	$[d_{r-1}, d_r)$	...	$[d_{n-1}, d_n]$	
c_1	q_{11}	...	q_{1r}	...	q_{1n}	M_{1+}
...	...	...	...	...	...	
c_i	q_{i1}	...	q_{ir}	...	q_{in}	M_{i+}
...	...	...	...	...	...	
c_s	q_{s1}	...	q_{sr}	...	q_{sn}	M_{s+}
Interval Total	M_{+1}		M_{+r}		M_{+n}	M

7.4.2 Quanta Matrix for Uncertain Data

The discretization algorithm is to find the minimal number of discrete intervals while minimizing the loss of class-attribute interdependency. Suppose F is a continues numeric attribute, and there exists a discretization scheme D on F, which divides the whole continues domain of attribute F into n discrete intervals bounded by the endpoints as $D : \{[d_0, d_1), [d_1, d_2), ..., [d_{n-1}, d_n]\}$, where d_0 is the minimal value and d_n is the maximal value of attribute F; $d_1, d_2, ..., d_{n-1}$ are cutting points arranged in ascending order.

In certain dataset, every value of attribute F is precise; therefore its value will be in only one of the n intervals. However, the value of an uncertain attribute can vary in a range, and the interval it belongs to is probabilistic. We use the probability that the value of an uncertain attribute belongs to an interval to model its membership. Thus, the class membership for a specify interval varies with different discretization scheme D. The class variable and the discretization variable of attribute F are treated as two random variables defining a 2-D quanta matrix, and Table 7.2 shows an example of quanta matrix.

In table 7.2, q_{ir} is the probability cardinality of the uncertain attribute $A_F^{u_n}$ which belongs to the i^th class and has its value within the interval $[d_{r-1}, d_r]$. Thus, according to equation 7.4, q_{ir} can be calculated as $q_{ir} = P_{c_i}(C = c_i, A_F^{u_n} \in [d_{r-1}, d_r))$. M_{i+} is the sum of the probability cardinality for objects belonging to the i^{th} class, and M_{+r} is the total probability cardinality of $A_F^{u_n}$ that are within the interval $[d_{r-1}, d_r)$, for $i = 1, 2, ..., s$, and $r = 1, 2, ..., n$.Then, the estimated joint probability that one uncertain attribute values $A_F^{u_n}$ of a tuple t is within the interval $D_r = [d_{r-1}, d_r)$ and t belongs to class c_i is calculated in equation (7.5).

$$p_{ir} = p(C = c_i, D_r | A_F^{u_n}) = \frac{q_{ir}}{M} \tag{7.5}$$

7.4.3 Uncertain Class-Attribute Interdependent Discretization

First, we briefly introduce the traditional Class-Attribute Interdependency Maximization (CAIM) discretization approach. CAIM is one of the classical discretization algorithms. It generates the optimal discretization scheme by quantifying the interdependence between classes and discretized attributes, and its criterion is defined in equation 7.6

$$CAIM(c, D | A_F^{u_n}) = \frac{\sum_{r=1}^{n} \frac{max_r^2}{M_{+r}}}{n} \tag{7.6}$$

where, n is the number of intervals, and r iterates through all inter values, *i.e.* $r = 1, 2, ..., n$. max_r is the maximum value in the r^{th} column of the quanta matrix, $i = 1, 2, ..., s$, M_{+r} is the sum of probabilities that the values of attribute F for each instance are in the interval $D_r = [d_{r-1}, d_r)$.

From the definition in equation (7.6), we can see that the *caim* value increases when the values of max_i grow, which indicates the increase of interdependence between the class labels and the discrete intervals. Thus, CAIM algorithm finds the optimal discretization scheme by searching the scheme with the highest *caim* value. Since the maximal value max_r dominants the value of CAIM criterion, the class

Table 7.3.
An example of uncertain dataset

Attribute($x : p_x$)	class($c_i : p_i$)
(0.1:0.3),(0.9:0.7)	(0:0.9),(1:0.1)
(0.1:0.2),(0.9:0.8)	(0:0.9),(1:0.1)
(0.9:1.0)	(0:1.0)
(0.2:0.7),(0.8:0.3)	(0:0.1),(1:0.9)
(0.1:0.7),(0.8:0.2)	(0:0.1),(1:0.9)

which max_r corresponds to is called *main class*. And a larger max_r indicates the more interdependency between main class and the interval D_r.

Although caim performs well in traditional certain cases, it encounters new challenges in uncertain datasets. For each interval, CAIM algorithm only takes the main class into account, but does not consider the distribution over all other classes, which leads to problems in its uncertain version. In an uncertain dataset, each instance no longer has a deterministic class label, but it has a discrete pdf over all possible classes, which reduces the interdependency between attributes and classes. We use the probability cardinality to build the quanta matrix for uncertain attributes, and then observe that the original *caim* criterion cannot well manage probabilistic data. Here we use an example in table 7.3 to show the potential drawbacks.

From table 7.3, we calculate the probability cardinality of attribute x in each class as following.

$$P(x = 0.1, c = 0) = 0.3 * 0.9 + 0.2 * 0.9 + 0.7 * 0.1 = 0.52$$

$$P(x = 0.1, c = 1) = 0.3 * 0.1 + 0.2 * 0.1 + 0.7 * 0.9 = 0.68$$

$$P(x = 0.2, c = 0) = 0.7 * 0.1 = 0.07$$

$$P(x = 0.2, c = 1) = 0.7 * 0.9 = 0.63$$

$$P(x = 0.8, c = 0) = 0.3 * 0.1 + 0.2 * 0.1 = 0.05$$

$$P(x = 0.8, c = 1) = 0.3 * 0.9 + 0.2 * 0.9 = 0.45$$

$$P(x = 0.9, c = 0) = 0.7 * 0.9 + 0.8 * 0.9 + 0.1 * 0.1 + 1.0 * 1.0 = 2.36$$

$$P(x = 0.9, c = 1) = 0.7 * 0.1 + 0.8 * 0.1 + 0.1 * 0.9 = 0.24$$

Referring to the definition of *caim* criterion, the *caim* value for the original quanta matrix in Table 7.4 is $caim = 3^2/(3 + 2) = 1.8$. From the distribution of attribute values in each class, we can see that the attribute values of instances in class 0 have a high probability cardinality at $x = 0.9$; and those instance in class 1 mainly locate in another end around $x = 0.1$, and $x = 0.2$. Obviously, $x = 0.5$ is a reasonable cutting point to generate one discretization scheme as $\{[0, 0.5), [0.5, 1.0]\}$. After splitting, the quanta matrix turns to be as in table 7.5, and its *caim* value is

$$caim = \frac{\frac{1.31^2}{1.31+0.59} + \frac{2.41^2}{2.41+0.69}}{2} = 1.38 \tag{7.7}$$

The goal of the CAIM algorithm is to find the discretization scheme with the highest *caim* value, so $[0, 0.5)[0.5, 1.0]$ will not be accepted as a better discretization scheme, because *caim* value decrease from 1.8 to 1.38 after splitting at $x = 0.5$. From this example, we see that data uncertainty obscure the interdependency between classes and attribute values by flatting the probability distributions. Therefore, when the original CAIM criterion is applied to uncertain data, it results in two new problems. First, it usually does not create enough intervals in the discretization scheme

Table 7.4.
Quanta matrix for the original uncertain dataset

class	Interval:[0,1]
0	3
1	2

Table 7.5.
Quanta matrix for splitting datasets

class	Interval	
	[0,0.5)	[0.5,1.0]
0	0.59	2.41
1	1.31	0.69

or it stops splitting too early, which causes the loss of class-attribute interdependence. Second, in order to increase the *caim* value, it is possible that the algorithm generates intervals with very small probability cardinalities, which reduces the robustness of the algorithm. For uncertain data, the attribute-class interdependence is probabilistic and is also modeled by a probability distribution. The original *caim* definition ignore this distribution and only considers the main class. Therefore, we revise the original definition of discretization criterion to discrete uncertain numerous data. Now that uncertainty blurs the attribute-class interdependency and reduces the difference between the main class and rest of the classes, we make the new criterion more sensitive to the change of values in quanta matrix. We propose the uncertain *caim* criterion called *ucaim*, which is defined in equation (7.8).

$$UCAIM(c, D|A_F^{u_n}) = \frac{\sum_{r=1}^{n} \frac{max_r^2 * \delta_r}{M_{+r}}}{n} \tag{7.8a}$$

$$\delta_r = \frac{\sum_{i=1, q_{ir} \neq max_r}^{s} max_r - q_{ir}}{s-1} \tag{7.8b}$$

In equation (7.8a), max_r is the maximum value among all q_{ir} values, which is the maxim value within the r^{th} column of the quanta matrix, and M_{+r} is the total probability of continues values of attribute F that are within the interval $D_r = [d_{r-1}, d_r)$. δ_r in equation (7.8b) is the average offset or difference for all other q_{ir} values to max_r.

The lager the attribute-class interdependence, the larger the value max_r/M_{+r} is. And CAIM then use it to identify splitting points. In the UCAIM algorithm, the δ_r shows how significant the main class is, compared to other classes. When δ_r is large, it means that within interval r, the probability that an instance belongs to the main class is much higher than the other classes, so the interdependence between interval r and the main class become high. In sum, we propose the *ucaim* criterion for two main reasons: (1) Compared with max_r/M_{+r}, we multiply it with the factor δ_r to make the value $\delta_r * max_r/M_{+r}$ more sensitive to interdependence changes, which usually are not that significant for uncertain data; (2) The value max_r/M_{+r} may be large merely because M_{+r} is small, which happens when there are not many instances falling into interval r. However δ_r does not have such problem, because it measures the relative relationship between main class and other classes.

Now we apply the new definition to the sample uncertain data in table 7.3. For the original quanta matrix in table 7.4, the *ucaim* value is

$$ucaim = \frac{3^2 * (3-2)}{5} = 1.8$$

And in the quanta matrix after splitting as in table 7.5, we have

$$\delta_1 = 1.31 - 0.59 = 0.72$$

$$\delta_2 = 2.41 - 0.69 = 1.72$$

$$ucaim = \frac{\frac{1.31^2 * 0.72}{1.31+0.59} + \frac{2.41^2 * 1.72}{2.41+0.69}}{2} = 1.98$$

Since *ucaim* value increases after splitting, the cutting point $x = 0.5$ will be accepted in the discretization scheme. From this example, we can see that *ucaim* has the potential to be more effective in finding the interdependency between attribute values and classes, compared to original approach.

7.4.4 Uncertain Discretization Algorithm

Algorithm 7.1 shows the detail UCAIM uncertain discretization algorithm. It consists of two main steps: (1) initialization of candidate interval endpoints and the initial discretization scheme; (2) iterative additions of new splitting points to achieve the highest value of *ucaim* criterion.

The time complexity of our *ucaim* algorithm is similar to the classic *caim* algorithm. For a single attribute, in the worst case, the running time of *caim* is $O(Mlog(M))$ [83], and M is the number of distinct values of the discretized attribute. In *ucaim* algorithm, the additional computation is to calculate S_r, which costs $O(C \cdot M)$ time. Because C is the number of class labels and usually is much smaller than M, the additional time cost is $O(M)$, and the final running time is still $O(Mlog(M))$.

7.5 Experimental Results

In this section, we present the experimental results of UCAIM discretization algorithm on eight datasets. We compare our technique with the traditional CAIM discretization algorithm to show the effectiveness of *ucaim* for uncertain data.

7.5.1 Setup

The datasets selected to test the UCAIM algorithm are: Iris plants dataset (Iris), Johns Hopkins University Ionosphere dataset (Ionosphere), Pima Indians Diabetes dataset (Pima), Glass Identification dataset (Glass), Wine dataset (Wine), Breast Cancer Wisconsin Original dataset (Breast), Vehicle Silhouettes dataset (Vehicle) and Statlog Heart dataset (Heart). All these datasets are drawn from UCI machine learning repository [59], and their detail information is shown in table 7.6.

These datasets are made uncertain by adding Gaussian distributed noises as in [70, 90]. The Gaussian noise added to each numeric attribute value has the zero

Algorithm 7.1: uncertain discretization Algorithm: UCAIM

```
Input: A database with continues uncertain attribute $A_F^{u_n}$ and its labels
       $C = c_1, ...c_s$
Output: Optimal discretization schema: $D$
Find the maximal and minimal possible values of $A_F^{u_n}$, recorded as $d_0$, $d_1$
Create a set $B$ of all potential endpoints. For uncertain attribute modeled by
sample based pdf, we sort all distinct possible values and use them to form the
set $B$; for formula based pdf, we use the mean of each distribution to form $B$.
Set the initial discretization scheme $D : [d_0, d_1]$, set $GlobalUcaim = 0$.
Let $k = 1$
forall the endpoints i, $i \in B$ do
    Initial new UCAIM value as $ucaim = 0$, and new scheme $newD = null$
    if $i \notin D$ then
        tentatively add $i$ to $D$ to generate a candidate discretization scheme
        $CD_i$
        compute the ucaim value of $CD_i$ as $ucaim_i$
        if $ucaim_i > ucaim$ then
            $ucaim = ucaim_i$
            $newD = CD_i$
        end
    end
end
if $ucaim > GlobalUcaim$ or $k < s$ then
    $GlobalUcaim = ucaim$
    $D = newD$
    $k = k + 1$
    goto
end
return D
```

Table 7.6.
Properties of experimental datasets

Dataset	#of classes	# of instances	# of attributes	# of continues
Iris	3	1	150	4
Ionosphere	2	351	34	34
Pima	2	768	8	8
Glass	7	214	10	10
Wine	3	178	13	13
Breast	2	699	10	10
Vehicle	4	846	18	18

mean, and its standard deviation is drawn from a uniform distribution in $[0, 2 * f * \sigma]$. Here, σ is the standard deviation of the original values of the attribute, and f is a parameter used to define different uncertain level. The value of f is selected from the set $\{1, 2, 3\}$. For the uncertainty in class labels, we assume the original class for each instance is the *main class*, and assign it a probability p_{mc}, and all other classes have the same probability $p = (1 - p_{mc})/(n - 1)$, where n is the number of all classes. Then, we drawn a sample for each uncertain attribute value in the uncertain dataset, and generate a new dataset as a comparison.

We use the accuracy of uncertain naïve Bayesian classifier to evaluate the quality of discretization algorithms. As the purpose of our experiment is to compare discretization algorithms, we ignore nominal attributes when we train the classifier. In the experiments, we first compare our UCAIM algorithm with the original CAIM algorithm (CAIM-O) which does not take data uncertainty into account. Next, we compare the UCAIM with the discretization algorithm named CAIM-M, which directly use caim criterion on uncertain quanta matrix.

Table 7.7.

Average classification accuracies with different discretization algorithms under different uncertain levels

Uncertain level	UCAIM	CAIM-M	CAIM-O
$f = 1$, $p_{mc} = 0.9$	79.53%	77.44%	72.51%
$f = 2$, $p_{mc} = 0.8$	78.19%	72.16%	70.15%
$f = 3$, $p_{mc} = 0.7$	71.79%	66.26%	61.96%

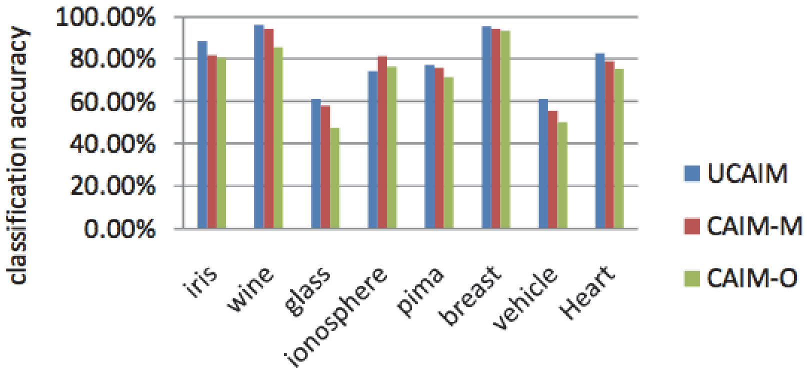
7.5.2 Results

The accuracy of uncertain naïve Bayesian classifier on these eight datasets is shown in table 7.8. The average classification accuracy under different uncertain levels for all three algorithms is shown in table 7.7. And figure 7.1 shows the detail performance comparison of these algorithms at each uncertain level.

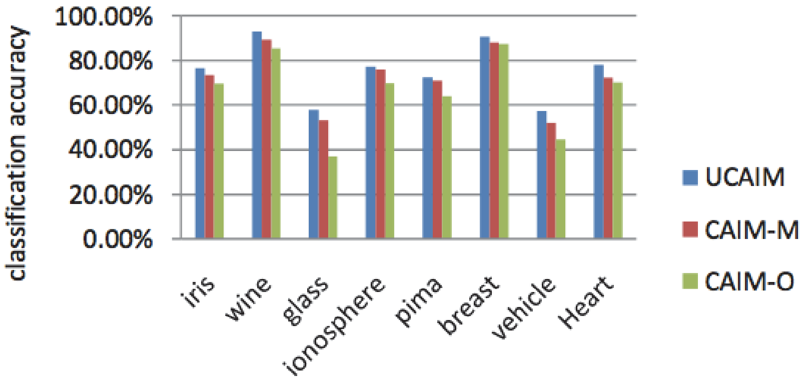
From table 7.7, 7.8 and figure 7.1, we can see that UCAIM outperforms the other two algorithms in most cases. Particularly, UCAIM has a more significant performance improvement for datasets with higher uncertainty. That is because UCAIM utilizes extra information such as pdfs of uncertain data and uses the new criterion to retrieve the class-attribute interdependency which is not that obvious when data are uncertain. Therefore, UCAIM is more powerful in helping improve the performance of naïve Bayesian classifier in uncertain data.

7.6 Conclusion

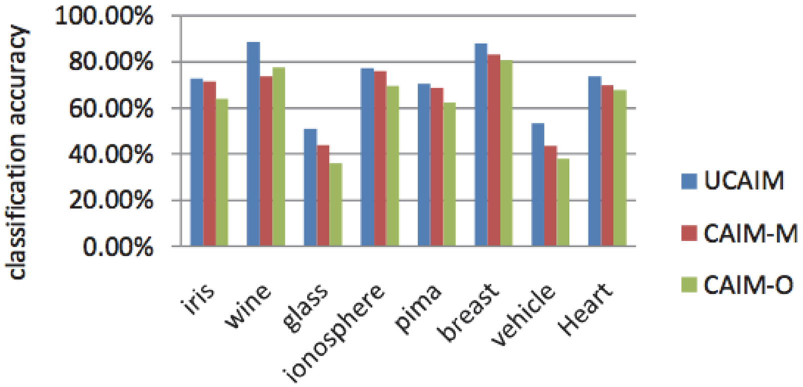
In this section, we propose a new discretization algorithm for uncertain data. We employ both formula-based and sample-based pdf to model numeric uncertain attributes, and define a new criterion to discover the class-attribute interdependency in the uncertain dataset. Experiments shows that our UCAIM algorithm can significantly help naïve Bayesian classifier to achieve a higher classifying accuracy.



(a) $f = 1, p_{mc} = 0.9$



(b) $f = 2, p_{mc} = 0.8$



(c) $f = 3, p_{mc} = 0.7$

Figure 7.1. Classification accuracies with different discretization algorithms under different uncertain levels

Table 7.8.

Accuracies of the uncertain Naïve Bayesian classifier with different discretization algorithms

Dataset	Uncertain level	UCAIM	CAIM-M	CAIM-O
Iris	$f = 1$, $p_{mc} = 0.9$	88.67%	81.67%	80.56%
	$f = 2$, $p_{mc} = 0.8$	76.67%	73.33%	69.56%
	$f = 3$, $p_{mc} = 0.7$	72.66%	71.33%	63.85%
wine	$f = 1$, $p_{mc} = 0.9$	96.07%	94.38%	63.85%
	$f = 2$, $p_{mc} = 0.8$	93.09%	89.32%	85.39%
	$f = 3$, $p_{mc} = 0.7$	88.44%	73.59%	77.53%
glass	$f = 1$, $p_{mc} = 0.9$	61.07%	57.94%	47.66%
	$f = 2$, $p_{mc} = 0.8$	57.94%	53.27%	37.07%
	$f = 3$, $p_{mc} = 0.7$	50.93%	43.92%	35.98%
Ionosphere	$f = 1$, $p_{mc} = 0.9$	74.09%	81.26%	76.31%
	$f = 2$, $p_{mc} = 0.8$	78.34%	77.13%	72.17%
	$f = 3$, $p_{mc} = 0.7$	77.20%	75.88%	69.66%
pima	$f = 1$, $p_{mc} = 0.9$	77.13%	75.74%	71.35%
	$f = 2$, $p_{mc} = 0.8$	72.32%	70.89%	63.97%
	$f = 3$, $p_{mc} = 0.7$	70.45%	68.66%	62.33%
breast	$f = 1$, $p_{mc} = 0.9$	95.42%	94.27%	93.36%
	$f = 2$, $p_{mc} = 0.8$	90.70%	87.83%	87.14%
	$f = 3$, $p_{mc} = 0.7$	87.83%	83.12%	80.68%
Vehicle	$f = 1$, $p_{mc} = 0.9$	61.22%	55.39%	50.13%
	$f = 2$, $p_{mc} = 0.8$	57.44%	52.12%	44.72%
	$f = 3$, $p_{mc} = 0.7$	53.19%	43.61%	37.87%
Heart	$f = 1$, $p_{mc} = 0.9$	82.59%	78.88%	75.33%
	$f = 2$, $p_{mc} = 0.8$	78.19%	72.16%	70.15%
	$f = 3$, $p_{mc} = 0.7$	73.63%	69.95%	67.76%

8 SUMMARY

This dissertation studies how to handle uncertainty information in data mining. It models two typical types of uncertainty in uncertain databases: tuple-level uncertainty and attribute-level uncertainty. And it aims to mine accurate results by incorporating uncertainty information in the process of data mining.

In particular, we investigate sequential pattern mining with either existential uncertainty or temporal uncertainty as a motivating example of how to incorporate uncertain information in traditional data mining algorithms. For different sequential pattern mining frameworks, we have developed various ways to handle uncertainty and highlighted that the result of uncertain sequential pattern mining strongly depends on both the domain and the user. Distributed computing platforms, such as MapReduce and Spark, are also utilized to mine sequential patterns in large scale uncertain databases, which make our algorithms more practical for real applications.

We also study uncertainty management in supervised machine learning processes. For example, we develop an artificial neural network to classify numeric uncertain data, in which we track the linear propagation of data uncertainty from input to output. Meanwhile, we also develop a Naïve Bayesian classifier to classify categorical uncertain data streams. We map the vector-like distribution of uncertain attributes to certain points in an Euclidean space, from which we learn the model for classifying uncertain streams. In addition, we design a discretization algorithm for uncertain data, since data pre-processing is an important step in uncertain data mining.

Mining uncertain data is challenging in both quality and performance. However, we explore various ways of uncertainty management and integrate them with different types of data mining application to mine satisfiable results from uncertain data. And our approaches are also proved to be very practical in real world applications.

REFERENCES

REFERENCES

[1] Kugatsu Sadamitsu, Satoshi Sekine, and Mikio Yamamoto. Sentiment analysis based on probabilistic models using inter-sentence information. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, 2008.

[2] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of ACM International Conference on Management of Data*, SIGMOD '03, pages 313–324, 2003.

[3] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd International Conference on Very Large Databases*, VLDB '06, pages 9–16. VLDB Endowment, 2006.

[4] Mauricio A. Hernández and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[5] Vladimir Ryabov and Seppo Puuronen. Estimation of uncertain relations between indeterminate temporal points. In *Advances in Information Systems*, volume 1909, pages 108–116, 2000.

[6] C.C. Aggarwal. On unifying privacy and uncertain data models. In *the 24th IEEE International Conference on Data Engineering*, ICDE '08, pages 386–395, 2008.

[7] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Querying imprecise data in moving object environments. In *IEEE Transactions on Knowledge and Data Engineering*, volume 16, pages 1112–1127, 2004.

[8] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *Proceedings of ACM International Conference on Management of Data*, volume 16, pages 34–48, 1987.

[9] F. Sadri. Modeling uncertainty in databases. In *IEEE International Conference on Data Engineering*, pages 122–131, 1991.

[10] T. Green and V. Tannen. Models for incomplete and probabilistic information. In *Proceedings of the 9th International Conference on Extending Database Technology*, pages 278–296, 2006.

[11] Charu C. Aggarwal, Yan Li, Jianyong Wang, and Jing Wang. Frequent pattern mining with uncertain data. In *International Conference on Knowledge Discovery and Data Mining*, SIGKDD '09, pages 29–38, 2009.

[12] Jianzhong Li, Zhaonian Zou, and Hong Gao. Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. *The VLDB Journal*, 21(6):753–777, 2012.

[13] Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[14] Michael Chau, Reynold Cheng, Ben Kao, and Jackey Ng. Uncertain data mining: An example in clustering location data. In *Proceedings of the 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, volume 3918, pages 199–204, 2006.

[15] Zhou Zhao, Da Yan, and Wilfred Ng. Mining probabilistically frequent sequential patterns in uncertain databases. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 74–85, 2012.

[16] Zhou Zhao, Da Yan, and Wilfred Ng. Mining probabilistically frequent sequential patterns in large uncertain databases. In *IEEE Transactions on Knowledge and Data Engineering*, volume 26, pages 1171–1184, 2013.

[17] Muhammad Muzammal and Rajeev Raman. Mining sequential patterns from probabilistic databases. In *Proceedings of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 210–221, 2011.

[18] Reza Sadri, Carlo Zaniolo, Amir Zarkesh, and Jafar Adibi. Expressing and optimizing sequence queries in database systems. In *ACM Transactions on Database Systems*, pages 282–318, 2004.

[19] Liwen Sun, Reynold Cheng, David W. Cheung, and Jiefeng Cheng. Mining uncertain data with probabilistic guarantees. In *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining*, pages 273–282, 2010.

[20] Xingzhi Sun, M.E. Orlowska, and Xue Li. Introducing uncertainty into pattern discovery in temporal event sequences. In *IEEE International Conference on Data Mining*, pages 299–306, 2003.

[21] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[22] Frank Höppner. Discovery of temporal patterns: Learning rules about the qualitative behaviour of time series. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 192–203, 2001.

[23] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. Discovering frequent arrangements of temporal intervals. In *Proceedings of the 5th IEEE International Conference on Data Mining*, ICDM '05, pages 354–361, 2005.

[24] Edi Winarko and John F. Roddick. Armada: An algorithm for discovering richer relative temporal association rules from interval-based data. *Data and Knowledge Engineering*, 63(1):76–90, 2007.

[25] Yongluan Zhou, Chunyang Ma, Qingsong Guo, Lidan Shou, and Gang Chen. Sequence pattern matching over time-series data with temporal uncertainty. In *Proceedings of 17th International Conference on Extending Database Technology*, pages 205–216, 2014.

[26] C.E.Dyreson and R.T.Snodgrass. Supporting valid-time indeterminacy. In *ACM Transactions on Database Systems*, volume 23, pages 1–57, 1998.

[27] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of ACM International Conference on Management of Data*, SIGMOD '03, pages 551–562, 2003.

[28] Jeffrey Jestes, Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data. In *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1903–1917, 2011.

[29] Thomas Bernecker, Hans-Peter Kriegel, Matthias Renz, Florian Verhein, and Andreas Zuefle. Probabilistic frequent itemset mining in uncertain databases. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining*, pages 119–128, 2009.

[30] Jian Pei, Jiawei Han, and Wei Wang. Mining sequential patterns with constraints in large databases. In *Proceedings of the 18th International Conference on Information and Knowledge Management*, pages 18–25, 2002.

[31] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, 1995.

[32] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17, 1996.

[33] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[34] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th ACM International Conference on Knowledge discovery and Data Mining*, pages 355–359, 2000.

[35] Jian Pei, Jiawei Han, Behzad Mortazavi-asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th IEEE International Conference on Data Engineering*, pages 215–224, 2001.

[36] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining*, pages 429–435, 2002.

[37] Jiong Yang, Wei Wang, Philip S. Yu, and Jiawei Han. Mining long sequential patterns in a noisy environment. In *Proceedings of the ACM International Conference on Management of data*, SIGMOD '02, pages 406–417, 2002.

[38] Ding-Ying Chiu, Yi-Hung Wu, and Arbee L. P. Chen. An efficient algorithm for mining frequent sequences by a new strategy without support counting. In *Proceedings of the 20th IEEE International Conference on Data Engineering*, pages 275–286, 2004.

[39] Li Wan, Ling Chen, and Chengqi Zhang. Mining frequent serial episodes over uncertain sequence data. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 215–226, 2013.

[40] Yuxuan Li, James Bailey, Lars Kulik, and Jian Pei. Mining probabilistic frequent spatio-temporal sequential patterns with gap constraints from uncertain databases. In *IEEE International Conference on Data Mining*, pages 448–457, 2013.

[41] Haopeng Zhang, Yanlei Diao, and Neil Immerman. Recognizing patterns in streams with imprecise timestamps. *The VLDB Endowment*, 3(1-2):244–255, 2010.

[42] Jiaqi Ge, Yuni Xia, and Jian Wang. Towards efficient sequential pattern mining in temporal uncertain databases. In *Proceedings of the 19th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 268–279, 2015.

[43] Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data algorithms and applications. In *IEEE Transactions on Knowledge and Data Engineering*, 21(5):609–623, May 2009.

[44] Chun-Kit Chui and Ben Kao. A decremental approach for mining frequent itemsets from uncertain data. In *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 64–75, 2008.

[45] Chun-Kit Chui, Ben Kao, and Edward Hung. Mining frequent itemsets from uncertain data. In *Proceedings of the 11th Pacific-Asia conference on Advances in knowledge discovery and data mining*, pages 47–58, 2007.

[46] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499, 1994.

[47] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Probabilistic event extraction from rfid data. In *In Proceedings of the 24th IEEE International Conference on Data Engineering*, pages 1480–1482, 2008.

[48] Yongxin Tong, Lei Chen, Yurong Cheng, and Philip S. Yu. Mining frequent itemsets over uncertain databases. In *Proceeding of the VLDB Endowment*, volume 5, pages 1650–1661, 2012.

[49] Byeong-Soo. Jeong, Ho-Jin. Choi, Md. Azam. Hossain, Md. Mamunur. Rashid, and Md. Rezaul. Karim. A mapreduce framework for mining maximal contiguous frequent patterns in large dna sequence datasets. In *IETE Technical Review*, volume 29, pages 162–168, 2012.

[50] Ming-Syan Chen Chun-Chieh Chen, CHi-Yao Tseng. Highly scalable sequential pattern mining based on mapreduce model on the cloud. In *Proceedings of IEEE International Congress on Big Data*, pages 310–317, 2013.

[51] Iris Miliaraki, Klaus Berberich, Rainer Gemulla, and Spyros Zoupanos. Mind the gap: Large-scale frequent sequence mining. In *Proceedings of ACM International Conference on Management of Data*, pages 797–808, 2013.

[52] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 165–172, 2013.

[53] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 2–2, 2012.

[54] Muhammad Muzammal and Rajeev Raman. On probabilistic models for uncertain sequential pattern mining. *Proceedings of the 6th International Conference on Advanced Data Mining and Applications*, pages 60–72, 2010.

[55] M.S.Chen J. W. Huang, S.C.Lin. Dpsp: Distributed progressive sequential pattern mining on the cloud. In *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 27–34, 2010.

[56] Deng Jie, Qu Zhiguo, Zhu Yongxu, Gabriel-Miro Muntean, and Wang Xiaojun. Towards efficient and scalable data mining using spark. In *International Conference on Information and Communications Technologies*, pages 1–6, 2014.

[57] Jiaqi Ge, Yuni Xia, and Jian Wang. Mining uncertain sequential patterns in iterative mapreduce. In *Proceedings of the 19th Pacific-Asia Conference on Knowledge Discover and Data Mining*, pages 243–254, 2015.

[58] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. In *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[59] K. Bache and M. Lichman. UCI machine leaning repository, 2013.

[60] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y. Yip. Efficient clustering of uncertain data. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 436–445, 2006.

[61] Jiangtao Ren, Sau Dan Lee, Xianlu Chen, Ben Kao, Reynold Cheng, and David Cheung. Naive bayes classification of uncertain data. In *Proceedings of the 9th IEEE International Conference on Data Mining*, pages 944–949, 2009.

[62] Chuancong Gao and Jianyong Wang. Direct mining of discriminative patterns for classifying uncertain data. In *Proceedings of the 16th ACM International Conference on Knowledge Discovery and Data Mining*, pages 861–870, 2010.

[63] Biao Qin, Yuni Xia, and Fang Li. Dtu: A decision tree for uncertain data. In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 4–15, 2009.

[64] Fernando Berzal, Juan-Carlos Cubero, Nicolás Marín, and Daniel Sánchez. Building multi-way decision trees with numerical attributes. *Information Science*, 165(1-2):73–90, September 2004.

[65] J. Bi and T. Zhang. Support vector machine with input data uncertainty. In *Proceedings of Advances in Neural Information Processing Systems*, pages 369–374, 2004.

[66] C.C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining*, pages 503–508, 2004.

[67] C.C. Aggarwal. On density based transforms for uncertain data mining. In *IEEE 23rd International Conference on Data Engineering*, pages 866–875, 2007.

[68] Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah, and Susanne Hambrusch. Indexing uncertain categorical data. In *Proceedings of the 23rd IEEE International Conference on Data Engineering*, pages 616–625, 2007.

[69] C.C. Aggarwal. On high dimensional projected clustering of uncertain data streams. In *IEEE International Conference on Data Engineering*, pages 1152 – 1154, 2009.

[70] Charu C. Aggarwal and Philip S. Yu. A framework for clustering uncertain data streams. In *Proceedings of IEEE International Conference on Data Engineering*, pages 150–159, 2008.

[71] Thanh T.L, Liping Peng, Boduo Li, Yanlei Diao, and Liu Anna. Pods: A new model and processing algorithm for uncertain data streams. In *Proceedings of the ACM International Conference on Management of Data*, pages 159–170, 2010.

[72] Haixun Wang, Wei Fan, and Philip S. Yu. Mining concept-drifting data stream using ensemble classifiers. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[73] M. Tan and L. Eshelman. Using weighted networks to represent classification knowledge in noisy domains. In *Machine Learning*, 1988.

[74] Kenneth A. Kaufman and Ryszard S. Michalski. Learning from inconsistent and noisy data. In *Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, ISMIS '99, pages 411–419, 1999.

[75] Krzysztof J. Cios and Lukasz A. Kurgan. *New learning paradigms in soft computing*. Physica-Verlag GmbH, 2002.

[76] Peter Clark and Tim Niblett. The cn2 induction algorithm. *Machine Learning*, 3(4):261–283, March 1989.

[77] Sotiris Kotsiantis and Dimitris Kanellopoulos. Discretization techniques: A recent survery. In *International Transactions on Computer Science and Engineering*, volume 32, pages 47–58, 2006.

[78] Usama M. Fayyad and Keki B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027, 1993.

[79] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202, 1995.

[80] Mark H. Hansen and Bin Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96:746–774, 1998.

[81] Y. Linde, A. Buzo, and R. Gray. An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*, 28(1):84–95, January 2003.

[82] Andrew K. C. Wong and David K. Y. Chiu. Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):796–805, June 1987.

[83] Lukasz A. Kurgan and Krzysztof J. Cios. Caim discretization algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):145–153, February 2004.

[84] Lukasz A. Kurgan and Krzysztof J. Cios. Fast class-attribute interdependence maximization (caim) discretization algorithm. In *Proceedings of International Conference on Machine Learning and Applications*, pages 30–36, 2003.

[85] Randy Kerber. Chimerge: discretization of numeric attributes. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 123–128, 1992.

[86] Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *In Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, pages 388–391, 1995.

[87] F. E. H. Tay and L. Shen. A modified chi2 algorithm for discretization. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):666–670, May 2002.

[88] Chao-Ton Su and Jyh-Hwa Hsu. An extended chi2 algorithm for discretization of real value attributes. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):437–441, March 2005.

[89] Ruoming Jin, Yuri Breitbart, and Chibuike Muoh. Data discretization unification. *Knowledge and Information Systems*, 19(1):1–29, March 2009.

[90] Charu C. Aggarwal and Philip S. Yu. Outlier detection with uncertain data. In *SIAM International Conference on Data Mining*, pages 483–493, 2008.

VITA

VITA

Jiaqi Ge received his BS and MSEE from Nanjing University in 2005 and 2008, respectively. Upon receiving his PhD from Purdue University in May 2016, he joined Expedia Inc as a data scientist.