Efficient Machine Learning on Relational Data

March 2018

Masafumi Oyamada

Efficient Machine Learning on Relational Data

Graduate School of Systems and Information Engineering

University of Tsukuba

March 2018

Masafumi Oyamada

Acknowledgements

I would like to thank my Ph.D advisor, Prof. Hiroyuki Kitagawa, for all his help and encouragement throughout my dissertation research. He told me everything about research in my bachelor and master days. His lesson, making efforts to telling research to others with a motivating story is as important as making efforts to conducting good research, is my principle as a researcher.

I also thank members of my committee Prof. Kazuhiko Kato, Prof. Toshiyuki Amagasa, Prof. Tetsuji Satoh, and Prof. Jun Sakuma for providing insightful comments and suggests on my work. I greatly appreciate the time and effort they spent on my behalf.

The work in this dissertation was mostly carried out at NEC Corporation where I was surrounded by smart, passionate, and encouraging colleagues. Discussions with those colleagues were always exciting and insightful. I am very grateful to Shinji Nakadai, who was my boss and a collaborator. He was always full of ideas, passionate, serious, and critical. I was very stimulated by his words. I thank co-authors who helped me to conduct research and publish papers: Jianquan Liu, Shinji Ito, Kazuyo Narita, and Takuya Araki.

Many thanks to all my friends who helped make my time fun and exciting. It was their support that helped me through times of struggle and stress.

Last, but by no means least I would like to thank my parents Hitomi and Shusuke for their love and nurture in all my years. They had a profound influence in making me who I am today.

Abstract

The rapid development of machine learning frameworks has widened the use of machine learning in enterprise businesses. In businesses, machine learning cannot be a single shot process but a repetitive trial-and-error process that heavily involves data analysts. Since the trial-and-error process has three bottleneck phases, “feature-engineering” phase, “model training” phase, and “model evaluation” phase in its loop, the whole process lasts for a long time, over a day, a week, or a month. This dissertation explores the ways to reduce the time of such machine learning process by tackling those three bottleneck phases. (1) First, we propose a query processing technique for accelerating feature-engineering in relational database systems. Concretely, our query processing technology accelerates repetitive aggregation operations, which often appear in the feature-engineering. (2) Second, we propose a machine learning model for predicting attributes of entity sets in relational database systems, which does not require feature-engineering phase. (3) Third, we propose a method for accelerating model training phase and model evaluation phase based on data compression technique.

Table of contents

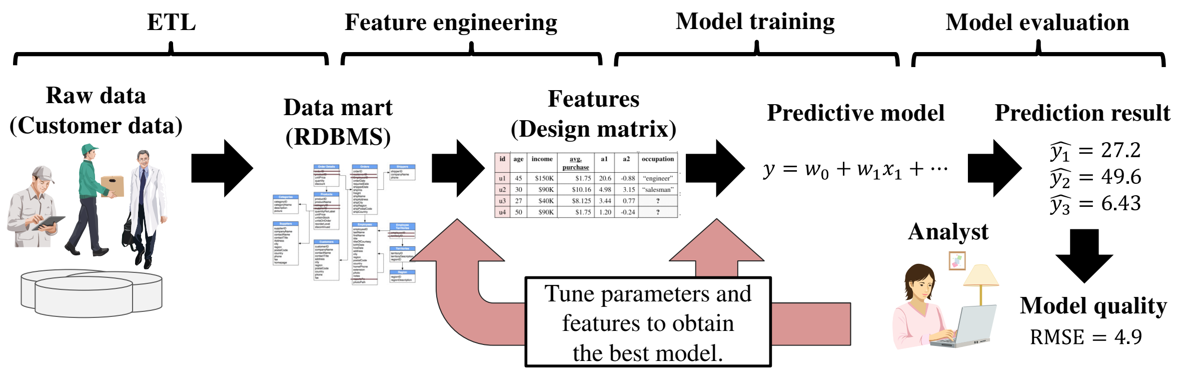
List of figures

List of tables

Chapter 1

Introduction

1.1 Background



id	age	income	avg. purchase	a1	a2	occupation
u1	45	$150K	$1.75	20.6	-0.88	"engineer"
u2	30	$90K	$10.16	4.98	3.15	"salesman"
u3	27	$40K	$8.125	3.44	0.77	?
u4	50	$90K	$1.75	1.20	-0.24	?

Fig. 1.1 Machine learning on relational data.

The rapid development of machine learning frameworks has widened the use of machine learning in various fields. One of the relatively new domains for machine learning is enterprise field where the data to be analyzed are stored and organized as relational data. In such enterprise relational data analysis, machine learning cannot be a simple single shot process but a repetitive trial-and-error process that heavily involves data analysts. Figure 1.1 shows the typical machine learning workflow on relational data [102, 64, 94]:

1. **(ETL)** Data to be analyzed (e.g., relational tables in other database systems and flat raw files such as CSV) are loaded into a *database management system* (DBMS) through *ETL* (Extract, Transform, and Load) process to construct a single database for analysis called a *data mart*. Since DBMS converts the data into its internal

data structure (e.g., row-based record structure or column-based record structure) and construct indices such as B+-Tree on the data for efficient query processing, this ETL process takes a long time. For this reason, typically this ETL process is done in batch processing manner, typically at night, after the close of business.

2. **(Feature engineering)** Data analysts try to design features that would yield good predictive performance in the subsequent machine learning methods by transforming tables in the data mart. Typically, in this phase, data analysts write tons of SQL queries that construct features from tables in the data mart based on his domain knowledge. For example, in customers' income prediction, aggregating the customers' past purchase histories may produce good features for income prediction, such as the average amount of money they use in the shopping. This phase involves the tremendous amount of aggregation query processing in DBMS and takes a long time.

3. **(Model training)** Data analysts pick up machine learning models to try considering the task (such as linear-regression for regression task, support vector machine for classification task) and train the models on the data by specifying hyper-parameter values of the models. Since a model is trained for each combination of algorithm and hyper-parameters, tons of models are trained in this phase, and it takes a long time (# of algorithms $\times$ # of candidate values of the first hyper-parameter $\times$ # of second hyper-parameter $\times$...).

4. **(Model evaluation)** Data analysts check the quality of the models by examining its predictive performance. Since the number of models to be evaluated could be large and each model evaluation includes the prediction on the training dataset, which is costly for large datasets, this phase also takes a long time.

If the predictive performance is not satisfactory in "model evaluation" phase (4), data analysts back to "feature-engineering" phase (2) or "model training" phase (3), and repeat the process by changing features to construct, algorithms to learn, and hyper-parameters to be used. Since this trial-and-error process will be repeated many times until data scientists get the satisfactory results, the whole machine learning process takes considerable time.

1.2 Overview

Motivated by these issues in machine learning on relational data, this dissertation explores methods to reduce the time of processes inside the trial-and-errors in the machine learning on relational data.

(Chapter 3) Fast Feature Engineering by Adaptive Partial Aggregation: First, we propose *Adaptive Partial Aggregation Tree (APA-Tree)*, a query processing technology for accelerating repetitive data aggregation operations. APA-Tree is based on partial aggregation method [73, 34] that pre-computes and cache aggregation values on a subset of the data and reuse pre-computed aggregation values in further aggregation queries. While conventional partial aggregation method computes aggregation values on pre-determined sized subsets of data (typically a page or a block in storage), APA-Tree computes aggregation values on dynamically-decided sized subsets. Since aggregation operations in feature engineering tasks have locality of reference, APA-Tree finely computes aggregation values on frequently accessed data and coarsely on rarely accessed data. As a result, APA-Tree accelerates repetitive aggregation queries with a small number of pre-computed aggregation values. Experimental results on synthetic workloads confirm APA-Tree's efficiently compared to the previous partial aggregation methods [73, 34].

(Chapter 4) Machine Learning without Feature Construction: Second, we propose a machine learning model for predicting entity attributes in relational database systems, which does not require conventional time-consuming feature-engineering process. As a motivating application of entity attribute prediction in relational data, we tackle the problem of customers' demographics prediction based on their behavioral data. This demographics prediction problem is modeled as a classification task in which a customer's sensitive demographic y is predicted from his feature vector $\boldsymbol{x}$. So far, two lines of work have tried to produce a "good" feature vector $\boldsymbol{x}$ from the customer's behavioral data: (1) application-specific feature engineering using behavioral data and (2) representation learning (such as singular value decomposition or neural embedding) on behavioral data. Although these approaches successfully improve the predictive performance, (1) designing a good feature requires domain experts to make a great effort and (2) features obtained from representation learning are hard to interpret. To overcome these problems, we present a *Relational Infinite Support Vector Machine (R-iSVM)*, a mixture-of-experts model that can leverage behavioral data. Instead of augmenting the feature vectors of customers, R-iSVM uses behavioral data to find out behaviorally similar customer clusters and constructs a local prediction model at each customer cluster. In doing so, R-iSVM successfully improves the predictive performance without requiring application-specific feature designing and hard-to-interpret representations. Experimental results on three real-world datasets demonstrate the predictive performance and interpretability of R-iSVM. Furthermore, R-iSVM can co-exist with previous demographics prediction methods to further improve their predictive performance.

(Chapter 5) Fast and Space-Efficient Machine Learning with Data Compression: Third, we propose a method to make training and evaluation of various machine learning models fast and space-efficient, which is based on data compression technique (Chapter 5). Our proposed *compressed vector set (CVS)* runs machine learning algorithms in a fast and space-efficient manner on both sparse and dense datasets. CVS holds a set of vectors in a compressed format and conducts primitive vector operations, such as ℓ_p-norm and dot product, without decompression. By combining these primitive operations, CVS accelerates prominent data mining or machine learning algorithms including k-nearest neighbor algorithm, stochastic gradient descent algorithm on logistic regression, and kernel methods. In contrast to the commonly used sparse matrix/vector representation, which is not effective for dense datasets, CVS efficiently handles sparse datasets and dense datasets in a unified manner. Our experimental results demonstrate that CVS could process both dense datasets and sparse datasets faster than conventional sparse vector representation with smaller memory usage.

1.2.1 Summary

This dissertation tackles three bottlenecks in a trial-and-error process in machine learning on relational data: feature engineering, model training, and model evaluation. We summarize how the time of those bottleneck steps will be reduced by our proposed technologies.

1. **(Feature engineering)** With APA-Tree (Chapter 3), the execution time of repetitive feature engineering queries will be drastically reduced. Moreover, for a specific task (entity attribute prediction of a relational data), R-iSVM (Chapter 4) can omit this time-consuming feature engineering step.

2. **(Model training)** With CVS (Chapter 5), machine learning models can be quickly trained.

3. **(Model evaluation)** CVS (Chapter 5) drastically reduces the machine learning models' prediction time, reducing computational time for model evaluation. Further, R-iSVM (Chapter 4) assists interpretation of trained models by revealing hidden structures in data, which reduces laborious model evaluation by "human."

Chapter 2

Related Work

In this chapter, we review studies on improving the performance of each step in machine learning on relational data (ETL, feature engineering, model training, model evaluation).

2.1 ETL

In ETL process, data are gathered from various places such as other database systems (e.g., OLTP database) or flat raw files such as CSV, and loaded into a single DBMS to construct a data mart. Since the loading process involves costly operations such as converting data into the DBMS's internal data structure and constructing indices such as B+-Tree for efficient query processing, this ETL process generally takes a long time. So far, several studies have tried to reduce the ETL time in different ways.

2.1.1 Fast Loading and Indexing

One important aspect in ETL is the trade-off between loading time and query processing time (such as feature engineering) after ETL. For example, when we avoid B+-Tree index construction in ETL process, ETL process itself could be instantly finished while the subsequent query processing will be unreasonably slow because of the lack of indices. So far, a variety of studies tried to both achieve fast loading and indexing. We review several studies in this direction.

Lightweight Index One direction is *Lightweight index*, which has the drastically lower construction cost compared to conventional indices such as B+-Tree. Moerkotte [73] proposed *Small Materialized Aggregates*, which horizontally splits a table into sub tables, computes statistics such as minimum and maximum values of each column in sub table, and stores those statistics into file header. These simple statistics work well as indices by allowing DBMS to evaluate range predicates only by checking the statistics. Similar structures are used in several commercial systems such as Brighthouse [88], Netezza [24], and DB2 [82]. Graefe *et al.* [34] extended the idea of the lightweight index to support not only range predicates but other predicates such as containment using the hashing technique.

Adaptive Index Another direction is *Adaptive index*, an indexing strategy that gradually constructs index typically in response to actual queries. Since no index construction is required in loading time, ETL itself can be finished instantly with this adaptive indexing strategy. *Database cracking* proposed by Idreos *et al.* [46] is a prominent work of this direction, which construct indices in response to range queries. Database cracking is then extended to support updates [47] and tuple reconstruction [48]. Graefe *et al.* considered adaptive indexing based on conventional B-tree structure [33]. Similarly, Abouzied *et al.* proposed a data loading and indexing scheme called *Invisible loading*, which loads data onto MapReduce system as a side effect of actual query processing [2].

In-situ Processing Another line of work is *In-situ processing*, which allows analysts to query raw data directly without loading the data into database systems enabling instant data analysis. NoDB [45] is a prominent work in this direction, which enables instant data analysis while achieving high query processing performance constructing indices called "positional maps" on raw data. Karpathiotakis *et al.* [59] also proposed high-performance in-situ processing system called "RAW", which constructs the system's internal data structure for tables in a just-in-time fashion. Given a raw data, RAW generates its internal data structure that is optimized for the data. With this technique, RAW drastically reduce the cost of converting raw data into system's internal data structure and further query processing time.

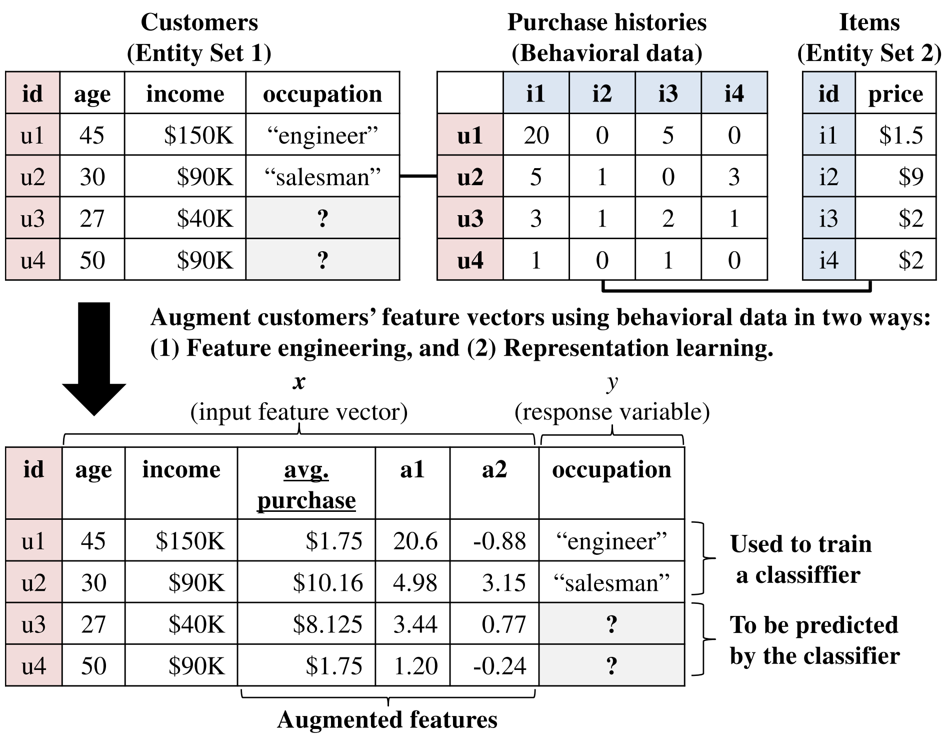
2.1.2 Other Acceleration Approaches

Besides indexing technologies, several studies have tried to improve the performance of ETL process.

Data Elimination Data elimination approaches reduce the amount of data to be loaded using meta information. Kargin *et al.* proposed *Lazy ETL*, which eliminates the amount of data to be loaded by using metadata attached to the flat files [57, 58]. They modified DBMS to only load metadata in ETL phase, used query optimization rules that reduce the amount data to be loaded by applying selection or projection operation using metadata.

Modern Hardware Another direction is to use modern hardware technologies to accelerate ETL process. Tobias *et al.* [93] proposed *Instant Loading*, a fast way to load flat files into database utilizing modern hardware instructions such as string SIMD operations (SSD 4.2's string and text instructions) and hardware optimized index structure, namely, adaptive radix tree [67].

2.2 Feature Engineering on Relational Data



Customers (Entity Set 1)
Purchase histories (Behavioral data)
Items (Entity Set 2)
Augment customers' feature vectors using behavioral data in two ways: (1) Feature engineering, and (2) Representation learning.
x (input feature vector)
y (response variable)
Used to train a classiffier
To be predicted by the classifier
Augmented features

id	age	income	occupation
u1	45	$150K	"engineer"
u2	30	$90K	"salesman"
u3	27	$40K	?
u4	50	$90K	?

	i1	i2	i3	i4
u1	20	0	5	0
u2	5	1	0	3
u3	3	1	2	1
u4	1	0	1	0

id	price
i1	$1.5
i2	$9
i3	$2
i4	$2

id	age	income	avg. purchase	a1	a2	occupation
u1	45	$150K	$1.75	20.6	-0.88	"engineer"
u2	30	$90K	$10.16	4.98	3.15	"salesman"
u3	27	$40K	$8.125	3.44	0.77	?
u4	50	$90K	$1.75	1.20	-0.24	?

Fig. 2.1 Feature engineering on relational data.

In feature engineering phase, data analysts try to design features that would yield good predictive performance in the subsequent machine learning methods by transforming tables in the data mart. Typically, in this phase, data analysts write tons of SQL queries that construct features from tables in the data mart based on his domain knowledge.

For example, as shown in Figure 2.1, aggregating the customers' past purchase histories and compute the average amount of money they use in the shopping can produce a good indicator of their occupation. This feature construction can be done by the following SQL query:

```
SELECT Customers.id, AVG(Items.price)
FROM   Customers, PurchaseHistory, Items
WHERE  Customers.id = PurchaseHistory.uid AND
       Items.id = PurchaseHistory.iid
GROUP BY Customers.id
```

Since such features are repeatedly generated, this phase involves the tremendous amount of aggregation query processing in DBMS and takes a long time.

Though not directly motivated by the feature engineering issue, there exist several database technologies that can accelerate such heavy feature engineering operations. In the rest of this section, we review studies that relate to feature engineering on relational data.

2.2.1 Efficient Aggregation Query Processing

First, we review technologies that can accelerate repetitive aggregation queries in database systems.

Pre-computation of Aggregation Values One major approach for accelerating aggregation operation is pre-computing and reusing aggregation values. As mentioned in the previous section, Moerkotte [73] proposed *SMA (Small Materialized Aggregates)*, which horizontally splits a table into sub-tables, computes statistics such as minimum, maximum, and sum values of each column in sub-table, and stores those statistics as a different table. SMA can be used as an index to avoid scanning unqualified records but can also be used to reduce the cost of aggregation operation by pre-computing and aggregation values. In the context of *OLAP (online analytical processing)*, Ho *et al.* proposed a data structure that hols prefix-sum and prefix-max on the range [43].

Approximate Query Processing In data analysis, sometimes analysts do not need exact answers but rough answers that can be computed instantly. To meet such requirements, *AQP (approximate query processing)* technologies, which compute approximate answers for aggregation query instantly, have been studied in the context of query processing in database systems [4, 41, 3, 13, 85, 87, 27, 14]. AQP is based on sampling subset of data from a database and compute unbiased estimates of aggregation queries such as sum and mean using the sampled subset. Since the size of the subset to sample can be arbitrarily small, AQP can return query results instantly. AQP can estimate errors in approximate answers such as confidence intervals or mean squared errors using statistical techniques like Hoeffding's inequality and central limit theorem. Chaudhuri *et al.* provided intensive literature reviews on AQP techniques [14].

2.2.2 Feature Engineering without Knowledge

In feature engineering on relational data, bottlenecks are not only in database systems side (query processing performance) but are also in analysts side; designing good features requires domain knowledge and long-time trial-and-error process. So far, several technologies have been studied to alleviate the burden on analysts.

Latent factor model In the prediction of an entity attribute (e.g., income of a customer) in relational data as in Figure 2.1, latent factors obtained from a relationship table (e.g., purchase histories) via *latent factor analysis* are known to work well. For example, *matrix factorization* on a matrix constructed from a relationship table, such as web browsing history [75, 44] and location check-in [97], has achieved successes. If the relationship table has foreign keys more than three, *tensor decomposition* method such as *Tucker decomposition* can be used instead of matrix factorization [105].

Representation learning Representation learning such as deep neural network frees data scientist from laborious feature engineering tasks by learning proper feature representations from training data. In the context of relational data, such deep neural networks are reported to work in various tasks including entity-attribute prediction [97] and link prediction[40].

2.3 Model Training

In the model training phase, data analysts pick up machine learning models to try considering the task (such as linear-regression for regression task, support vector machine for classification task) and train the models on the data by specifying hyper-parameter values of the models. Since a model is trained for each combination of algorithm and hyper-parameters, tons of models are trained in this phase, and it takes a long time

(# of algorithms $\times$ # of candidate values of first hyper-parameter $\times$ # of second hyper-parameter $\times$...).

In this section, we review studies that accelerate the training of machine learning models.

2.3.1 Sparse Matrix Representation

Sparse matrix representations represent a vector that is mostly filled with zero values in a space-efficient way. One of the most commonly-used representations is COO (coordinate list) [90, 31], which represents a vector with non-zero values and its positions. For example, a sparse vector

$$\boldsymbol{x} = (0, 0, 42, 99, 0, 0, 0)$$

is represented by two vectors

$$\begin{aligned}\text{values} &= (42, 99)\\ \text{positions} &= (2, 3)\end{aligned}$$

in COO format.

While the sparse vector representation reduces the space-efficiency, they also reduce the computational complexity of several mathematical operations. For example, the dot product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, which are comprised of B_x and B_y non-zero elements, can be carried out in $\mathcal{O}(B_x + B_y)$ time [90, 31]. Since training of machine learning models mostly consists of matrix computation, sparse matrix representation has been actively used to accelerate model training phase of machine learning.

One downside of the sparse vector representation is its inefficiency for dense data. For example, a dense vector

$$\boldsymbol{x} = (7, 7, 42, 99, 7, 7, 7)$$

is represented by

$$\begin{aligned}\text{values} &= (7, 7, 42, 99, 7, 7, 7)\\ \text{positions} &= (0, 1, 2, 3, 4, 5, 6)\end{aligned}$$

in COO representation, which doubled the size compared to the original vector.

Most of the linear algebra libraries implement sparse vectors. Popular libraries include Eigen [36] and Scipy [55, 77].

2.3.2 Run-Length Encoding

In contrast to sparse vector representation, *run-length encoding* can represent a dense data efficiently when the number of distinct elements is not large [80, 6]. Run-length encoding represents a sequence of n same consecutive values $x, \ldots, x$ with a block $\langle n, x \rangle$. For example, RLE represents

$$(1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1)$$

as $(\langle 5, 1 \rangle, \langle 9, 2 \rangle, \langle 3, 1 \rangle)$, reducing the number of elements to represent the data from 17 to 6.

Run-length encoding is widely used in data-intensive systems. For example, MADlib [42], a data analytics system built on top of a relational database system, use run-length encoding to handle sparse data efficiently. Also, some columnar database systems employ run-length encoding its column data [1, 69]. [78] also employs run-length encoding to accelerate data mining algorithms and machine learning algorithms.

Data Reordering

In run-length encoding for a collection of vectors, reordering the order of samples or vector dimensions can increase the data compression rate [78]. Brodie *et al.* tackled the row-reordering problem for maximizing the compression rate of run-length encoding of a matrix, and proposed a greedy method [10]. In their situation, the greedy method was enough, because they aimed to compress the state-transition tables of a regular expression, and commonly such tables are not large. In data mining area, the problem is extensively studied as the problem of reordering bitmap indices and several efficient algorithms are proposed [79, 68, 81].

When the interest is only in the compression of a (long) vector, Burrows-Wheeler Transform (BWT), a well-known data preprocessing technique to increase the compression rate of several compression algorithms, can be used [11].

2.3.3 Model-specific Data Compression

In machine learning research area, compressing the specific machine learning models to accelerate the training of machine learning algorithm is an active research direction. Tabei *et al.* studied partial least squares regression (PLS) on compressed data encoded by grammar-based codes [91]. To assist accessing the elements in the compressed matrix, they proposed a tree-based special data structure. Their approach targets to use binary-features (so-called fingerprints data), which differs from our approach that targets to arbitrary real values. Rendle proposed a method to accelerate machine-learning algorithms by utilizing block structures in a matrix [84]. In his work, input matrices are assumed to have special block structures that come from denormalization of relational tables.

2.3.4 Data Discretization

Many data compression algorithms, such as run-length encoding, is ineffective for real-valued data, because real-valued data rarely have same consecutive values. For dealing with the problem, data discretization techniques are often employed to convert real-valued vectors into discrete-valued vectors. Since data discretization will lose the information, how to keep the accuracy of subsequent algorithms is the important problem in data discretization. In machine learning area, several approaches for keeping the accuracy of machine learning are proposed [22, 83, 106].

2.4 Model Evaluation

In the model evaluation phase, data analysts check the quality of the models by examining its predictive performance. Since the number of models to be evaluated could be large and each model evaluation includes the prediction on the training dataset, which is costly for large datasets, this phase also takes a long time.

In this section, we review studies to accelerate the evaluation of the machine learning models.

2.4.1 Binary Embedding

Binary embedding techniques maps $\mathbb{R}^D$ to B-dimensional Hamming space, typically with $B < D$ [95, 62, 32, 28]. Benefits of this approach are twofold. First, binary embedding is space-efficient: B-dimensional binary vector can be stored in B bits. Second, computations over binary embedded vectors are fast: dot product of two binary embedded vectors can be carried out by taking `XOR` of two vectors and then counting the number of 1 in the vector, which is just a `popcount` instruction. Recently, [28] proposed an efficient binary-coding scheme. They demonstrated that a popular binary-encoding method proposed in [32] is a constrained version of their method, which results in weaker performance. They also showed that another state-of-the-art approach [62] could not satisfy all the optimal binary-coding conditions.

Recently, binary embedding approaches are actively employed in machine learning algorithms. For example, in deep neural networks research, binary embedding [83] and ternary embedding [106] are shown to drastically increase the predictive performance of large neural networks, which reduce the time of model evaluation drastically.

2.4.2 Vector Quantization

In computer vision and information retrieval area, *vector quantization* techniques, which represents a collection of vectors with a relatively small number of representative vectors, are also actively studied. Like binary embedding methods, vector quantization methods can also accelerate model evaluation phase. Compared to the binary embedding, vector quantization approaches tend to yield better compression rate while the compression speed is slower [8]. The most simple vector quantization approach is k-means clustering algorithm, which can be considered as an encoding from original vector to centroid vectors. Recently, *product quantization*, a generalization of k-means, is proposed [53] and its variants are actively studied [76, 96, 29]. Product quantization splits the vector into M disjoint subvector, and runs k-means on each sub-vectors. The concatenation of each encoded sub-vectors is used as the final encoded vector.

2.4.3 Model-Specific Compression

Since *deep neural network (DNN)* models contain a ton of parameters (e.g., 61M parameters in AlexNet [63]) and resulting in the huge memory usage and computational power, compression of DNN models is the hot topic in deep learning research community. Approaches include low-rank approximation of parameter matrix [19, 52], binary representation [83] or ternary representation [106] of parameters, and pruning unimportant nodes from deep neural network.

2.5 Summary

In this chapter, we have reviewed studies on improving the performance of each step in machine learning on relational data (ETL, feature engineering, model training, model evaluation). In this section, summarize the originality and technical contributions of this dissertation regarding the related work we have reviewed.

2.5.1 Adaptive Partial Aggregation Tree

Adaptive Partial Aggregation Tree (APA-Tree), a query processing technology for accelerating repetitive data aggregation operations (Chapter 3) relates to *fast loading and indexing methods* reviewed in Section 2.1.1 and *efficient aggregation query processing methods* reviewed in Section 2.2.1.

Comparison with fast loading and indexing methods APA-Tree can be used as a fast loading and indexing method while its main objective is on aggregation query acceleration. In the viewpoint of indexing methods, *adaptive index* methods such as *database cracking* [46] share similar construction policy with APA-Tree: both methods start without any indices and gradually construct indices that are optimized for actual workloads. Key difference is that our proposed APA-Tree has *shrink phase*, which merges unimportant nodes (infrequently accessed nodes) to keep the tree-size small, while previous methods do not have the corresponding phase.

Comparison with efficient aggregation query processing methods Conventional aggregation query processing methods such as partial aggregation methods [73, 34] and OLAP-oriented indices [43] are all *static* where structure-related parameters such as block sizes have to be pre-determined regarding the assumed workloads. Thus, resulting data structures in conventional methods may perform inefficiently on unanticipated workloads. In contrast, our proposed APA-Tree gradually adapts to actual workloads by changing its structure according to queries, making the method robust.

2.5.2 Relational Infinite Support Vector Machine

Relational Infinite Support Vector Machine (R-iSVM), a machine learning model for predicting entity attributes in relational database systems, which does not require feature-engineering process (Chapter 4) relates to feature engineering without knowledge.

Comparison with feature engineering without knowledge As mentioned in Section 2.2.2, conventional feature engineering without expert knowledge can be categorized into *latent factor models* such as matrix factorization and tensor decomposition [75, 44, 97, 105] and *representation learning* such as deep neural networks [97, 40]. While both methods can generate features effective for increasing predictive performance, those generated features rarely have meanings and are hard-to-interpret, making data analysts' understanding of the model and explanation to other people difficult. In contrast, our proposed R-iSVM does not produce such hard-to-interpret features while achieving the comparable or better performance with previous methods by taking a different way to leverage relational data: constructs multiple simple models regarding the relational data.

2.5.3 Compressed Vector Set

Compressed Vector Set (CVS), a method to make training and evaluation of various machine learning models fast and space-efficient using data compression technique (Chapter 5) relates to various works.

Comparison with sparse matrix representation CVS relates to vector and matrix computation frameworks that utilize data sparsity. For example, Eigen [36] and Scipy [55, 77] can represent sparse matrices and sparse vectors by concise data structures. In contrast to the sparse matrix representation, CVS targets to not only the sparse vector sets but also the dense vector sets.

Comparison with run-length encoding CVS compresses data using run-length encoding scheme. Run-length encoding is widely used in data-intensive systems such as machine learning frameworks [42] and columnar database systems [1, 69]. However, none of these works include optimization techniques introduced in our proposed method CVS, such as dimension-reordering and data discretization.

Chapter 3

Fast Feature Engineering by Adaptive Partial Aggregation

In this section, we propose a query processing technique for accelerating feature-engineering in DBMS. Concretely, our query processing technology accelerates repetitive range aggregation operations, which often appear in feature-engineering phase.

3.1 Introduction

Range aggregation query is a fundamental operation in DBMS that aggregates records in a certain range by computing some statistics such as maximum value, average, and standard deviation. For example,

```
SELECT AVG(Height)
FROM   Students
WHERE  Age < 16
```

computes the average height of students under 16.

Such a range aggregation query is a classic operation in DBMS and has been used for a long time, especially in OLAP (Online Analytic Processing) applications. However, the recent democratization of machine learning has provided range aggregation query a new role: feature-engineering in machine learning. Such statistics of a subset of data computed by range aggregation query are often used as features in machine learning task.

In machine learning process, when the predictive performance of the model trained by constructed features is not satisfactory, data scientists modify features by rewriting queries (such as changing aggregation range from $Age < 16$ to $Age < 14$) and re-dispatching them to the DBMS. As a result, similar looking tons of aggregation queries are sequentially sent to the DBMS resulting in a heavy workload, and it is reported to be the bottleneck in data analytics [5].

Motivated by the issue, in this chapter, we propose *Adaptive Partial Aggregation Tree* (*APA-Tree*), a query processing technology for accelerating repetitive data aggregation operations. APA-Tree is based on partial aggregation method [73, 34] that pre-computes and cache aggregation values on a subset of the data and reuse pre-computed aggregation values in further aggregation queries. While conventional partial aggregation method computes aggregation values on pre-determined sized subsets of data (typically a page or a block in storage), APA-Tree computes aggregation values on dynamically-decided sized subsets. Since aggregation operations in feature engineering tasks have locality of reference, APA-Tree finely computes aggregation values on frequently accessed data and coarsely on rarely accessed data. As a result, APA-Tree accelerates repetitive aggregation queries well with a small number of pre-computed aggregation values. Experimental results on synthetic workloads confirm APA-Tree's efficiently compared to the previous partial aggregation methods [73, 34].

3.2 Preliminaries

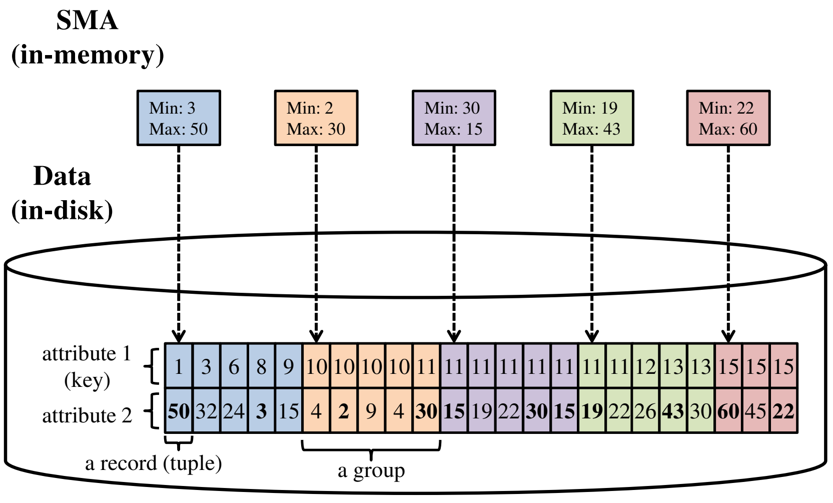


Fig. 3.1 Horizontal Partitioning [73, 34]

Our approach is based on *partial aggregation method* [73, 34], which has been tried to accelerate range aggregation query in classic OLAP context. The idea of partial aggregation method is to reuse the aggregation values among queries as much as possible. Partial aggregation method first divides records into several groups, pre-computes aggregation values called *partial aggregation values* on each group, caches those partial aggregation values into memory or CPU cache as shown in Figure 3.1. Later, partial aggregation reuses those pre-computed aggregation values as much as possible in aggregation query processing to avoid costly storage accesses. In this section, we provide the brief overview of partial aggregation method and its inherent issue in applying to feature engineering in machine learning.

3.2.1 Decomposability of Aggregation Operation

Partial aggregation accelerates aggregation operations that have the following property:
Definition 3.2.1 (Decomposable aggregation operation [100]) *Aggregation operation* $f : R(A_1, ..., A_n) \rightarrow \mathbb{R}$ *is decomposable if there exists a function* $g : \mathbb{R}^m \rightarrow \mathbb{R}$ *such that*

$$f(B) = g(f(B_1), ..., f(B_m))$$

for a relation $B = \bigcup_{i=1}^{m} B_i \in R(A_1, ..., A_n)$*, where* $R(A_1, ..., A_n)$ *is the schema of the relation* R*, comprising of attributes* $A_1, ..., A_n$*.*

Intuitively, decomposable aggregation operations refer to the operations that can be computed in divide-and-conquer manner. For example, $\text{Max}_a(X)$ operation, which computes the maximum value of attribute a in set X, is a decomposable aggregation operation because

$$\text{Max}_{age}(B_1 \cup B_2) = \max\{\,\text{Max}_{age}(B_1), \text{Max}_{age}(B_2)\,\}$$

holds where max works as the combiner. In contrast, $\text{Count}(X)$ operation, which computes the cardinality of set X, is not decomposable because there is no combiner that recovers the correct value

$$\text{Count}_{age}(B_1 \cup B_2)$$

from individually computed $\text{Count}(B_1)$ and $\text{Count}(B_2)$ for arbitrary sets B_1 and B_2. For example, let $B_1 = \{\,1, 1, 4\,\}$ and $B_2 = \{\,1, 2, 3\,\}$. Since same element 1 is in each set, correct aggregation value

$$\text{Count}(B_1 \cup B_2) = \text{Count}(\{\,1, 1, 1, 2, 3, 4\,\}) = 4$$

cannot be recovered from individual statistics ($\text{Count}(B_1) = 2$ and $\text{Count}(B_2) = 3$).

3.2.2 Reusing Pre-Computed Partial Aggregation

Partial aggregation method uses the decomposable property of aggregation operations to reuse pre-computed aggregation values among several queries. Consider a range aggregation query $\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(R))$. According to the distributive property of selection operation σ and decomposable property of Max,

$$\begin{aligned}
&\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(R))\\
&=\max\{\,\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(B_1)),\\
&\qquad\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(B_2)),\\
&\qquad\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(B_3))\}
\end{aligned} \tag{3.1}$$

holds where relation R is partitioned into three groups, $R = B_1 \cup B_2 \cup B_3$. Assume that group B_1 only contains small people ($130 \leq R.height \leq 150$), group B_2 only contains tall people ($185 \leq R.height$), and group B_3 only contains mid-height people ($175 \leq R.height \leq 180$). Since partial aggregation method holds pre-computed statistics such as minimum height and maximum height in memory, range aggregation query in Equation (3.1) can be processed as

$$\begin{aligned}
&\max\{\,\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(B_1)),\\
&\qquad\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(B_2)),\\
&\qquad\text{Max}_{age}(\sigma_{170\leq R.height\leq 180}(B_3))\}\\
&=\text{Max}(\{\,\text{Max}_{age}(\phi),\text{Max}_{age}(\phi),\text{Max}_{age}(B_3)\,\})\\
&=\text{Max}_{age}(B_3)
\end{aligned} \tag{3.2}$$

without scanning blocks B_1, B_2, and B_3, which reside in secondary storage. Because partial aggregation keeps the value of $\text{Max}_{age}(B_3)$ in memory as a pre-computed partial aggregation value, the whole query can be processed without costly secondary storage access.

3.3 Proposed Method

In this section, we describe the details of our proposed method *Adaptive Partial Aggregation Tree* (*APA-Tree*), a processing technique to accelerate repetitive range aggregation queries.

3.3.1 Issues in Conventional Partial Aggregation Method

Though conventional partial aggregation method can successfully reduce the I/O cost of range aggregation queries, it has difficulty in actual deployment, especially in feature-engineering of machine learning. The reason is in its group construction part; How it divides relation (aggregation target table) R into groups, such as $R = B_1 \cup B_2 \cup B_3$. Conventional approaches [73, 34] divide the relation into groups based on its primary key so that every group has the equal-sized key range (For example, B_1 consists of the range $1 \leq k \leq 30$, B_2 consists of the range $31 \leq k \leq 60$, and B_3 consists of the range $61 \leq k \leq 90$). We call this policy as *Horizontal Partitioning* in this chapter.

The difficulty of Horizontal Partitioning in actual deployment is that analysts or administrator have to decide the proper group size in *a priori* manner, before the actual analysis. If the group size is too small, I/O on most queries cannot be eliminated. If the group size too large, scanning and holding pre-computed statistics could be a heavy process.

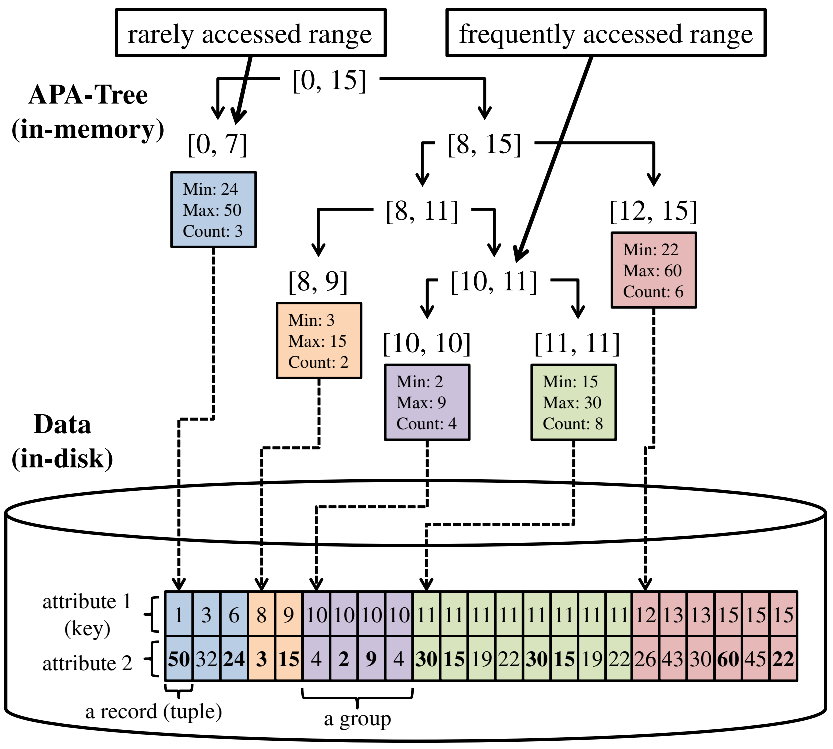
3.3.2 APA-Tree

Motivated by the problem in Horizontal Partitioning, we propose a new approach to maintain aggregation values in partial aggregation method, namely, *Adaptive Partial Aggregation Tree* (*APA-Tree*).

Figure 3.2 shows the structure of APA-Tree. APA-Tree is a binary tree similar to Segment tree. As in Segment tree, a node recursively divides key range equally (each node maintains a specific range), and a leaf node represents a group that holds pre-computed aggregation values of records in the group. Unlike Segment tree, APA-Tree is not a balanced binary tree. In APA-Tree, frequently accessed key ranges are finely divided, and less accessed ranges are coarsely divided. In doing so, APA-Tree successfully maintains partial aggregation values in proper group sizes that reflect the access patterns of actual queries, which leads to high reusability in partial aggregation method.

Maintaining APA-Tree

To maintain partial aggregation values in a proper granularity that reflects the access patterns of actual queries, APA-Tree grows or shrinks the tree when it processes a range aggregation query. Algorithm 1 and Algorithm 2 show the algorithms of range aggregation query processing in APA-Tree. It recursively traverses the tree to find all



(a)

Fig. 3.2 The illustration of the proposed APA-Tree. While Horizontal Partitioning strategy maintains aggregation values inside statically decided groups, APA-Tree maintains aggregation values dynamically and adaptive to actual queries.

leaf nodes that are enclosed by the range specified by the query, and combines partial aggregations using the combiner for the aggregation operation[1] (line 17 of Algorithm 1).

Growing the tree If the query cannot be answered by pre-computed partial aggregations in the current tree structure, APA-Tree grows the tree so that the same query can be processed by pre-computed partial aggregations, that is, without any I/Os (line 13 of Algorithm 1). This growing process corresponds to computing the partial aggregation values finely on frequently accessed key ranges.

Shrinking the tree Since only growing the tree monotonically increase the tree size, APA-Tree shrinks a subtree if the subtree is not frequently accessed (line 6 in Algorithm 2). This shrinking process corresponds to computing the partial aggregation values coarsely

[1]As in previous studies [100, 73, 34], we assume that we have *a priori* knowledge on combiners for aggregation operations.

Algorithm 1: computeAggregationForNode(node, query): Compute partial aggregation result for the data maintained by the node.

```
Input: node: Node
Input: query: Range aggregation query
Output: Partial aggregation results for the data maintained by the node.
1  // Range comparison to check data maintained by the node is target of the query
2  answer ← compare(node.range, query)
3  if answer = OUTSIDE_QUERY then
4      // Terminate further traversal because the data maintained by the node is out
         of the query
5      return null
6  if answer = ALL_DATA_IN_QUERY then
7      // Reuse partial aggregation result of the node because the node is enclosed by
         the query
8      return node.getPrecomputedAggregations()
9  if answer = PART_DATA_IN_QUERY then
10     // The node and query overlaps.
11     if node.isLeafNode() then
12         // Grow APA-Tree: Split the node so that the partial aggregation values
             can be reused in further queries.
13         node.split(query)
14     leftAggregations = computeAggregationForNode(node.leftChild, query)
15     rightAggregations = computeAggregationForNode(node.rightChild, query)
16     // Merge partial aggregation results using Partial aggregation method method.
17     return leftAggregations.merge(rightAggregations)
```

on rarely accessed key ranges. To this end, APA-Tree maintains the access frequency of each node using cache-replacement algorithms such as *Least Recently Used* (*LRU*).

3.4 Experiments

We compared the performance of our proposed APA-Tree with conventional Horizontal Partitioning [73, 34]. For the cache replacement policy, we use CLOCK[92], a lightweight alternative to LRU policy because LRU itself is difficult to be implemented in low memory/computational footprints. We used synthetic data that has the following schema

```
Members(age INT, height DOUBLE, weight DOUBLE)
```

Algorithm 2: evalRangeAggregationQuery(query): Range aggregation query evaluation in APA-Tree

```
Input: query: Range aggregation query
Output: Result of the renge aggregation query
Data: rootNode: Root node of the APA-Tree
Data: TREE_SIZE_LIMIT: Maximum number of nodes in APA-Tree (group size)
1 // Compute aggregation value
2 aggregations ← computeAggregationForNode(rootNode, query)
3 while rootNode.treeSize > TREE_SIZE_LIMIT do
4     // Shrink APA-Tree: Find the least accessed node and merge the node with
        its sibling node.
5     leastAccessedLeaf ← rootNode.findLeastAccessedLeaf()
6     leastAccessedLeaf.parent.mergeChildren()
7 return aggregations
```

and used age attribute as the clustered index (primary key). All attribute values are generated from the uniform distribution. The number of records is set to 100K for all the experiments.

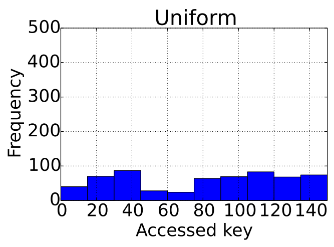
3.4.1 Workloads

As mentioned earlier, our proposed method, APA-Tree, is especially effective when queries are skewed. To confirm this characteristic, we generated the following two workloads and used in experiments:

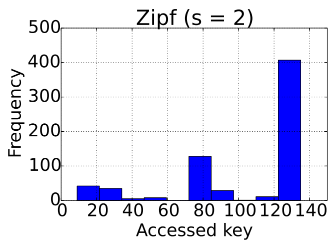
1. **(Uniform)** Uniform workload consists of range aggregation queries on attribute "age". In uniform workload, a query that has the range $[l, l+w]$ is generated in the following steps: first l is generated from uniform distribution, and then range width w is generated from normal distribution $N(\mu, \sigma^2)$.

2. **(Zipf)** Zipf workloads simulate skewed queries. In a Zipf workload, a query that has the range $[l, l+w]$ is generated in the following steps: first, l is generated from Ziphian with skewness parameter s, and then range width w is generated from normal distribution $N(\mu, \sigma^2)$. To check the influence of the skewness on the performance, we varied the skewness parameter s of Ziphian from $s = 2$ (moderately skewed) to $s = 4$ (highly skewed).

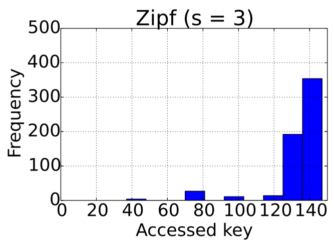
Figure 3.3 show the accessed key distribution in each generated workload. We can confirm that in Ziphian, the larger s is, the workload is more skewed.



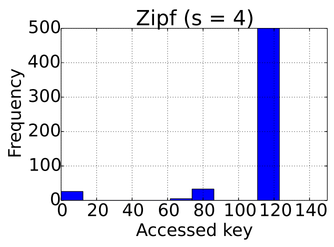
(a) Uniform



(b) Zipfian (s=2)



(c) Zipfian (s=3)



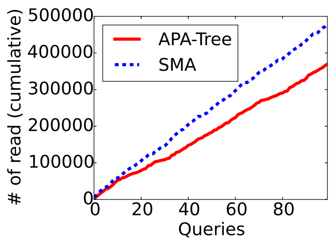
(d) Zipfian (s=4)

Fig. 3.3 Accessed key distribution in each workload.

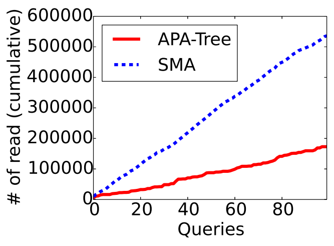
3.4.2 I/O Performance Comparison

Figure 3.4 shows the amount of cumulative I/O of our proposed APA-Tree and Horizontal Partitioning (SMA) in each workload where the maximum number of leaf nodes is set to 32. From Figure 3.4, we can observe that the more workload skews, the better our proposed APA-Tree performs compared to conventional Horizontal Partitioning. Even in the uniform workload, our approach outperformed conventional Horizontal Partitioning since the uniform also has some skews as shown in Figure 3.3 because the workload is generated in a randomized manner.

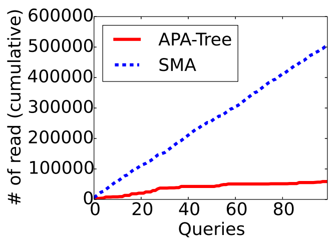
Next, we confirm the effect of the number of groups on the performance of conventional Horizontal Partitioning and our proposed APA-Tree. Figure 3.4 to Figure 3.7 shows the performance comparison of those two methods in different numbers of groups (32, 64, 128, 256). Naturally, the higher the number of groups is, the amount of I/Os is reduced because the probability of a group is enclosed by a query increases. Also, we can observe that as the number of groups grows, the performance difference between conventional Horizontal Partitioning and our proposed APA-Tree becomes smaller. Since increasing the number of groups increases the memory footprints used by pre-computed aggregations, the maximum number of groups is determined by the available memory in actual deployment.



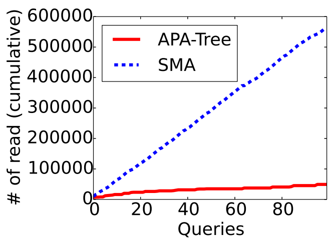
(a) Uniform



(b) Zipfian (s=2)

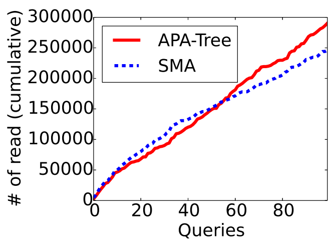


(c) Zipfian (s=3)

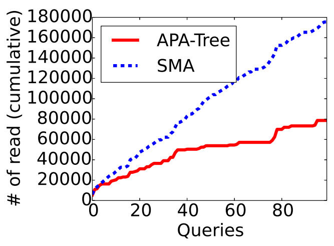


(d) Zipfian (s=4)

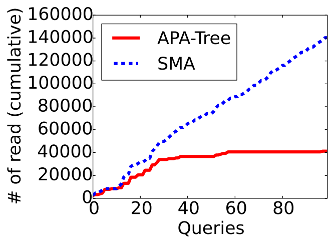
Fig. 3.4 Cumulative I/O in each workload (number of groups is 32). APA-Tree stands for our proposed approach and SMA stands for Horizontal Partitioning [73, 34].



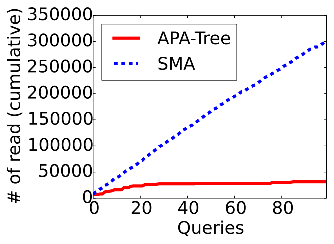
(a) Uniform



(b) Zipfian (s=2)

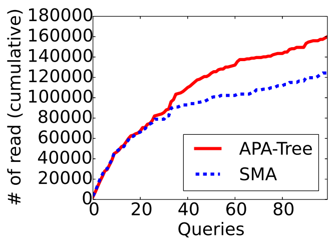


(c) Zipfian (s=3)

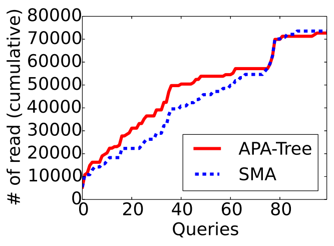


(d) Zipfian (s=4)

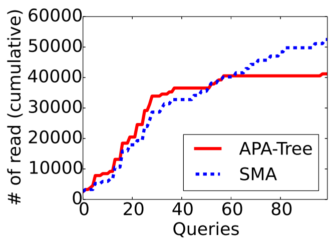
Fig. 3.5 Cumulative I/O in each workload (number of groups is 64).



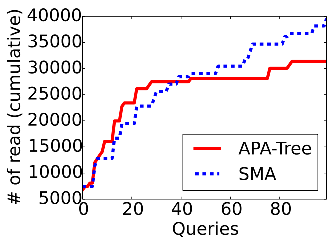
(a) Uniform



(b) Zipfian (s=2)

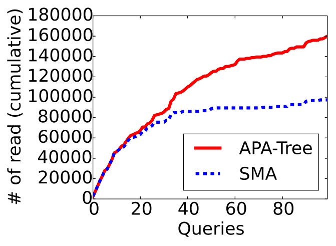


(c) Zipfian (s=3)

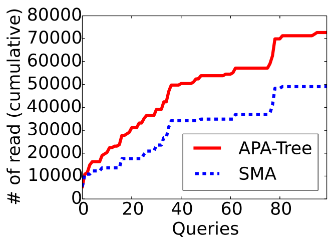


(d) Zipfian (s=4)

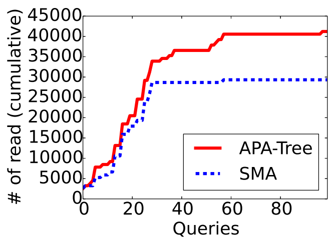
Fig. 3.6 Cumulative I/O in each workload (number of groups is 128).



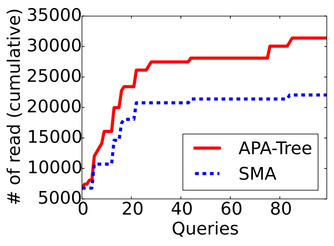
(a) Uniform



(b) Zipfian (s=2)



(c) Zipfian (s=3)



(d) Zipfian (s=4)

Fig. 3.7 Cumulative I/O in each workload (number of groups is 256).

3.5 Related Work

One major approach for accelerating aggregation operation is pre-computing and reusing aggregation values. As mentioned in the previous section, Moerkotte [73] proposed *SMA (Small Materialized Aggregates)*, which horizontally splits a table into sub-tables, computes statistics such as minimum, maximum, and sum values of each column in sub-table, and stores those statistics as a different table. SMA can be used as an index to avoid scanning unqualified records but can also be used to reduce the cost of aggregation operation by pre-computing and aggregation values. In the context of *OLAP (online analytical processing)*, Ho *et al.* proposed a data structure that hols prefix-sum and prefix-max on the range [43].

In data analysis, sometimes analysts do not need exact answers but rough answers that can be computed instantly. To meet such requirements, *AQP (approximate query processing)* technologies, which compute approximate answers for aggregation query instantly, have been studied in the context of query processing in database systems [4, 41, 3, 13, 85, 87, 27, 14]. AQP is based on sampling subset of data from a database and compute unbiased estimates of aggregation queries such as sum and mean using the sampled subset. Since the size of the subset to sample can be arbitrarily small, AQP can return query results instantly. AQP can estimate errors in approximate answers such as confidence intervals or mean squared errors using statistical techniques like Hoeffding's inequality and central limit theorem. Chaudhuri *et al.* provided intensive literature reviews on AQP techniques [14].

3.6 Summary

In this chapter, we proposed *Adaptive Partial Aggregation Tree (APA-Tree)*, a query processing technology for accelerating repetitive data aggregation operations. APA-Tree is based on partial aggregation method [73, 34] that pre-computes and cache aggregation values on a subset of the data and reuse pre-computed aggregation values in further aggregation queries. While conventional partial aggregation method computes aggregation values on pre-determined sized subsets of data (typically a page or a block in storage), APA-Tree computes aggregation values on dynamically-decided sized subsets. Since aggregation operations in feature engineering tasks have locality of reference, APA-Tree finely computes aggregation values on frequently accessed data and coarsely on rarely accessed data. As a result, APA-Tree accelerates repetitive aggregation queries with a small number of pre-computed aggregation values. Experimental results on synthetic workloads confirmed APA-Tree's efficiently compared to the previous partial aggregation methods [73, 34].

Chapter 4

Machine Learning without Feature Construction

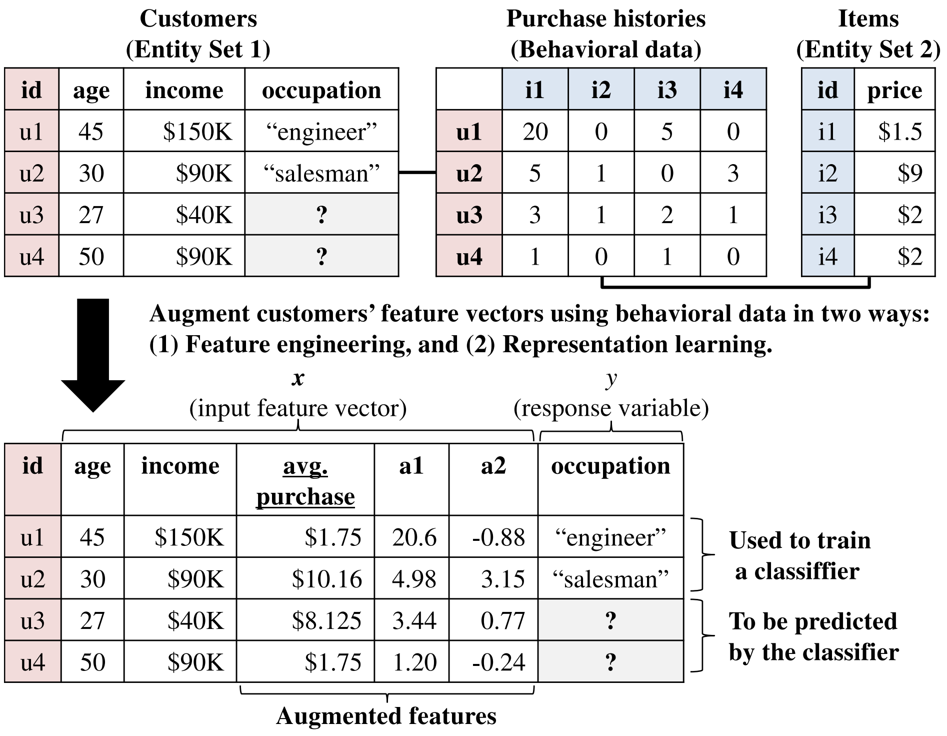
In the previous section, we tried to reduce the time of feature-engineering phase in machine learning process. In this section, we take a different approach: we show that feature-engineering process can be eliminated for a specific (but a common) machine learning task, namely, customers' demographics prediction.

4.1 Introduction

Customer demographics, such as age and income, are essential information in various tasks including product planning, advertisement, and item recommendation [104]. Since not every customer makes available sensitive demographics like occupation, *customer demographics prediction* has received notable attention from both industry and academia [75, 44, 21, 105, 18, 97, 103].

Demographics prediction is conventionally formalized as a classification problem that predicts unknown demographics from known demographics [75, 44, 105, 97]. Because demographics have correlations like "older people tend to have large incomes (income and age positively correlate)," this formulation has achieved successes and been widely used. As shown in Figure 4.1, if "occupation" is unavailable for customers $u3$ and $u4$ but "age" and "income" are available for customers $u1$ to $u4$, we can learn a classifier using demographics of $u1$ and $u2$ as a training set. Then the classifier predicts "occupation" of customers $u3$ and $u4$ from their "age" and "income."

Customer's *behavioral data*, such as purchase histories and web browsing histories, has been actively used to predict demographics because just using demographic correlations sometimes results in poor predictive performance [75, 44, 21, 105, 18, 97, 103]. The



id	age	income	occupation
u1	45	$150K	"engineer"
u2	30	$90K	"salesman"
u3	27	$40K	?
u4	50	$90K	?

	i1	i2	i3	i4
u1	20	0	5	0
u2	5	1	0	3
u3	3	1	2	1
u4	1	0	1	0

id	price
i1	$1.5
i2	$9
i3	$2
i4	$2

id	age	income	avg. purchase	a1	a2	occupation
u1	45	$150K	$1.75	20.6	-0.88	"engineer"
u2	30	$90K	$10.16	4.98	3.15	"salesman"
u3	27	$40K	$8.125	3.44	0.77	?
u4	50	$90K	$1.75	1.20	-0.24	?

Fig. 4.1 Conventional demographics prediction methods augment customer's feature vectors using "behavioral data" to improve the predictive performance in two ways: (1) application-specific feature engineering, in which domain experts aggregate behavioral data to produce a meaningful feature (average purchased item price is effective for predicting "occupation"), and (2) representation learning, such as Singular Value Decomposition, finds good feature vectors automatically (a1 and a2). Feature engineering requires considerable effort, and representation learning lacks interpretability. Our proposed model avoids these problems while achieving high predictive performance.

underlying intuition of these studies is that behavioral data can describe demographics well, which is sometimes expressed in a more catchy phrase: "you are what you buy" [103]. These studies can be divided into two approaches: (1) *application-specific feature engineering* using behavioral data [18] and (2) *representation learning*, which learns feature vectors from behavioral data [75, 44, 105, 97]. In (1) application-specific feature engineering, as depicted in Figure 4.1, we can augment a customer's feature vector by gathering his behavioral data, such as the average price of items he has purchased or categories of books he has read, which may be a good indicator of his occupation. In this approach, how to design a good feature is an important problem that requires domain experts' knowledge. In (2) representation learning, *Singular Value Decomposition (SVD)* and *Tucker Decomposition* on behavioral data (e.g., user-item

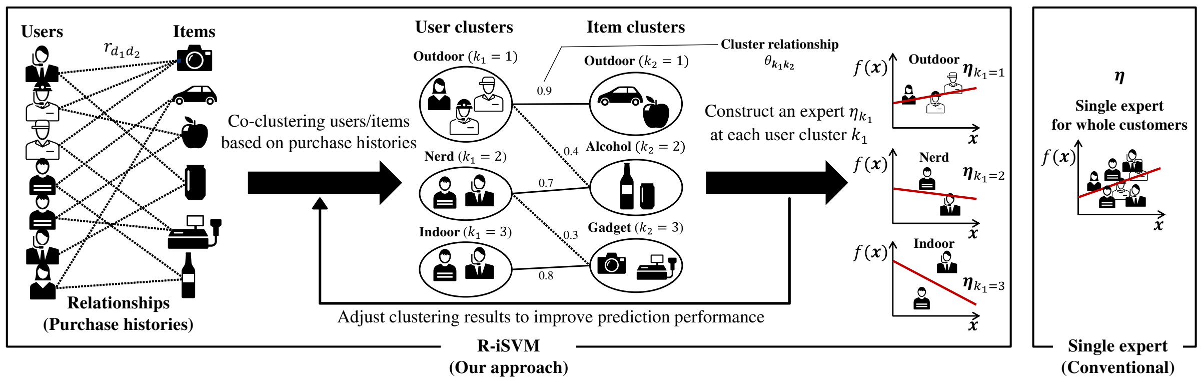


Fig. 4.2 Instead of using a prediction model for all customers (right), R-iSVM trains a local prediction model for each behaviorally similar customer cluster (left). R-iSVM first simultaneously groups users and items into clusters using behavioral data (purchase histories) in co-clustering fashion to identify behaviorally similar customers (E-Step), and identifies the strengths of cluster-cluster relationships between user- and item-clusters (M-Step), which later help interpretation of each cluster. Then R-iSVM trains a local demographics prediction model $\boldsymbol{\eta}_{k_1}$ at each user cluster (QP-Step). After that, R-iSVM adjusts the clustering results to improve predictive performance by checking training errors of experts (E-Step). By repeating these three steps in EM-fashion, R-iSVM constructs a mixture-of-experts model.

matrix or user-item-store tensor) has been actively used for producing low-dimensional feature vectors for customers [75, 44, 105]. A neural-embedding method has recently been applied to learn a more discriminative representation of a customer from his item purchase histories [97].

Although these demographics prediction methods achieved successes, we encountered several problems in deploying these methods to production. (1) Application-specific feature engineering requires domain-experts to make hard effort to find good features and is one of the most time-consuming tasks, involving significant trial and error [5]. Furthermore, application specific features limit the model's applicability to other domains. (2) Representation learning is apparently effective for improving predictive performance. However, automatically generated feature vectors (representations) rarely have meanings, and the prediction result would be difficult to interpret and explain to customers and colleagues (for example, "income and age positively correlate" is an intuitive explanation, but "income and auto-generated-feature-1 positively correlate" is hard to explain).

4.1.1 Design Goals

To overcome the aforementioned issues in conventional demographics prediction methods, we set **four design goals** for our model:

1. **(General)** It does not require application specific modeling, and is not even limited to demographics prediction.

2. **(Interpretable)** *It does not produce hard-to-interpret features.* Furthermore, it provides additional information that helps people interpret prediction results.

3. **(Accurate)** It achieves good predictive performance. Further, *it can co-exist with conventional feature engineering or representation learning methods* and helps to improve their predictive performances.

4. **(Scalable)** Its computational complexity is linear in the size of behavioral data. Further, the training algorithm can be fully parallelized.

In this chapter, for a model that fulfills the four design goals, we present a *Relational Infinite Support Vector Machine* (R-iSVM), a mixture-of-experts model that can leverage behavioral data. As shown in Figure 4.2, instead of constructing a prediction model for all customers, R-iSVM finds behaviorally similar customers through co-clustering [61, 99] on behavioral data, and constructs a local demographics prediction model at each customer cluster. R-iSVM jointly models co-clustering and training of prediction models as a unified optimization problem, and thus those tasks affect each other to improve the model quality. Further, R-iSVM determines the characteristic of each customer demographics prediction model using the co-clustering results, which helps to increase interpretability. For a local demographics prediction model at each customer cluster, we use a multi-class kernel machine [17] that has better predictive performance than generalized linear models. To fill the gap between the co-clustering model (Bayesian generative model) and the multi-class kernel machine (discriminative model), we leverage the recently developed *Regularized Bayes* theory [108].

4.2 Preliminaries

In this section, we review several concepts that appear in our model: *entity-relationship data*, *mixture-of-experts* models, and *co-clustering* methods.

Table 4.1 List of synonyms. Depending on the context, we sometimes use these words interchangeably.

Entity-Relationship Data	**Demographics Prediction**
Entity set	Customers / items
Entity	Customer / item
Entity attributes	Customer demographics
Relationship information	Behavioral data

4.2.1 Entity-Relationship Data

In enterprise, customer information including demographics and purchase histories are often organized as *Entity-Relationship Data.* As shown in Figure 4.1, entity-relationship data consists of (1) several *entity sets* that contain "master" information (e.g., customer demographics or item prices), and (2) a *relationship information* that connects those entity sets (e.g., purchase histories such as "customer A bought item B" in retail stores). Although its main target is demographics prediction, our relational mixture-of-experts model can be used in the more general task: *entity attribute prediction in entity-relationship data.* In this chapter, to make the discussion general, we sometimes use the terms in Table 4.1 interchangeably.

4.2.2 Mixture-of-Experts Model

In classification problems such as customer demographics prediction, a classifier must capture a non-linear dependency between feature vector $\boldsymbol{x}$ and its true label y to achieve accurate prediction. So far, many non-linear classifiers such as non-linear kernel machines [17] and neural networks [97] have been proposed and widely used. However, such non-linear models tend to lack interpretability because they do not show how each dimension of the input feature vector affects the classification result [15, 70].

Mixture-of-experts model [51] is an approach for capturing non-linearity without sacrificing interpretability. Rather than constructing a single prediction model for a whole dataset, a mixture-of-experts model separates input feature space into K regions and constructs a local prediction model $\boldsymbol{w}_k$, called an *expert*, at each region k. Since feature vector $\boldsymbol{x}$ and label y are likely to show a linear dependency at each region k, a simple linear model can work well as an expert, allowing us to investigate how each dimension of the feature vector $\boldsymbol{x}$ affects the classification result y.

In the prediction phase, a mixture-of-experts model classifies input feature $\boldsymbol{x}$ as label y using experts $\mathcal{W} = \{ \boldsymbol{w}_1, \ldots, \boldsymbol{w}_K \}$ in accordance with

$$\begin{aligned} p(y \mid \boldsymbol{x}, \mathcal{W}) &:= \mathbb{E}_{p(z|\boldsymbol{x})}[\, p(y \mid \boldsymbol{x}, \boldsymbol{w}_z)\,] \\ &= \sum_{k}^{K} p(z = k \mid \boldsymbol{x}) p(y \mid \boldsymbol{x}, \boldsymbol{w}_k), \end{aligned} \tag{4.1}$$

where z is the latent variable of input feature $\boldsymbol{x}$ representing the expert assignment. Probability density function $p(z = k \mid \boldsymbol{x})$ is called a *gating function* (or gating network) that softly assigns the input feature $\boldsymbol{x}$ to expert $\boldsymbol{w}_k$ in accordance with its value. Depending on the form of a gating function, mixture-of-experts models can become various prediction models including a Gaussian mixture of linear classifiers [51], a probabilistic decision tree [56], and a supervised co-clustering model [20].

In this chapter, we present R-iSVM, a mixture-of-experts model for entity-relationship data. R-iSVM has a special gating function that assigns entity d_1 (e.g., a customer) to an expert by considering its relationship information $\{ r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}$ (e.g., his purchase histories) where $r_{d_1 d_2}$ indicates a relationship information (e.g., customer d_1 has bought item d_2) and $\mathcal{I}$ indicates indices of observed relationships. That is, the gating function in R-iSVM has the form of

$$p(z_{d_1} = k \mid \boldsymbol{x}_{d_1}, \{ r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}), \tag{4.2}$$

whereas the gating function in a conventional mixture-of-experts model has the form of $p(z_{d_1} = k \mid \boldsymbol{x}_{d_1})$, which use the value of entity attribute $\boldsymbol{x}_{d_1}$ (e.g., basic demographics of customer d_1). We elaborate on this discussion in Sections 4.3.1 and 4.3.2.

4.2.3 Co-Clustering

Co-clustering (relational clustering) methods conduct clustering on multiple datasets simultaneously by using the relationship information [98, 61, 99]. Since co-clustering is based on the relationship information that represents "behavior," it can find clusters of behaviorally similar entities. For instance, as shown in Figure 4.2, co-clustering on users, items, and purchase histories detects customers who have similar buying habits and items bought by similar customers. *Stochastic Block Model* (SBM) [98] is a seminal work in probabilistic latent variable modeling for such co-clustering. Kemp *et al.* developed the *Infinite Relational Model* (IRM) that extends SBM into a Bayesian nonparametrics

Table 4.2 List of symbols/notations.

Symbol/Notation	Description
$[N]$	$\{1,\dots,N\}$
$d_i \in D_i$	Identifier of an entity in i-th entity set (e.g., a customer / an item)
$\boldsymbol{x}_{d_i} \in \mathbb{R}^{M_i}$	Input feature vector of entity d_i, which is M_i-dimensional (e.g., basic demographics)
$y_{d_i} \in Y_i$	Class label of entity d_i (e.g., sensitive demographics)
$\mathcal{I} \subset D_i \times D_j$	Indices of observed relationships
$r_{d_i d_j} \in \mathbb{R}$	Relationship between entity d_i and entity d_j ($(d_i, d_j) \in \mathcal{I}$) (e.g., purchase amount)
$K_i \in \mathbb{N}$	# of clusters in i-th entity set
$k_i \in [K_i]$	Identifier of a cluster in i-th entity set
$\boldsymbol{\eta}_{k_i} \in \mathbb{R}^{\lvert Y_i \rvert M_i}$	Expert at cluster k_i (e.g., demographics prediction model)
$z_{d_i} \in [K_i]$	Latent variable of entity d_i (cluster assignment)

model [61, 99]. IRM uses a Dirichlet process for cluster construction, and it can infer the number of clusters without computationally intensive model selection, which is required in SBM.

4.3 Proposed Model

In this section, we present the *Relational Infinite Support Vector Machine* (R-iSVM), a mixture-of-experts model that can leverage behavioral data.

4.3.1 Motivation

Our model is based on the intuition that incorporating behavioral data into demographics prediction will improve the predictive performance, as in previous demographics prediction methods [75, 44, 21, 105, 18, 97, 103]. However, in contrast to the previous work, we pursue an interpretable model, and augmenting a feature vector with representation learning on behavioral data or non-linear transformation of a feature vector is unpromising. This poses a research question:

Research Question (Explainable Demographics Prediction) *Can we utilize behavioral data to improve the performance of demographics prediction without losing interpretability, i.e., keeping original feature vectors as they are?*

We found that mixture-of-experts [51, 56] has the similar purpose: to improve the predictive performance without changing the original feature vectors. However, applying the vanilla mixture-of-experts model does not meet our needs since mixture-of-experts models select prediction model on the basis of feature vector $\boldsymbol{x}$. In demographics prediction, feature vector $\boldsymbol{x}$ represents basic demographics, such as age and gender, and selecting a prediction model on the basis of these demographics means *customers who have similar basic demographics will have the same prediction model.* This policy differs from our expectation: *customers who have similar behavior will have the same prediction model.*

To make behaviorally similar customers have the same prediction model, we need to change the form of the gating function from

$$p(z_{d_1} = k \mid \boldsymbol{x}_{d_1}),$$

to

$$p(z_{d_1} = k \mid \boldsymbol{x}_{d_1}, \{\, r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \,\}), \tag{4.3}$$

where $\mathcal{I}$ indices indices of all behaviors and $r_{d_1 d_2}$ indicates a behavior such as customer d_1 has bought item d_2. In this chapter, we use probabilistic co-clustering models [98, 61, 99] to define a gating function, though various gating functions that have the form of Equation (4.3) can be chosen. This is because co-clustering models have achieved successes in behavioral data modeling and we empirically confirmed that it performs well as a gating function on real-world datasets. Designing a new gating function can be an interesting future research direction in relational mixture-of-experts models.

4.3.2 Overview

Before explaining R-iSVM in more detail, we first provide the overview: how it trains a mixture-of-experts model from existing demographics and behavioral data, and uses the trained model to predict demographics. For ease of explanation, we hereinafter focus on a simple situation:

- Entity set D_1 represents customers, and each customer is identified by $d_1 \in D_1$.
- Entity set D_2 represents items, and each item is identified by $d_2 \in D_2$.

- Relationship $r_{d_1d_2}$ represents a purchase history[1]. When customer d_1 has bought item d_2, $(d_1, d_2) \in \mathcal{I}$ and $r_{d_1d_2} = 1$.

Training

Given a set of customer demographics

$$\{ (\boldsymbol{x}_{d_1}, y_{d_1}) \}_{d_1 \in D_1}$$

and their purchase histories $\{ r_{d_1d_2} : (d_1, d_2) \in \mathcal{I} \}$ as a training set, R-iSVM trains K_1 experts and computes the probability of the expert assignment $p(z_{d_1} \mid \{ r_{d_1d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \})$ for all customers. Refer to the caption of Figure 4.2 for the detailed training flow.

Prediction

Given customer d_1's basic demographics $\boldsymbol{x}_{d_1}$ and his purchase histories $\{ r_{d_1d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}$, R-iSVM predicts his sensitive demographics y^* by

$$y^* = \arg\max_{y \in Y_1} F(y, \boldsymbol{x}_{d_1})$$

where

$$\begin{aligned} F(y, \boldsymbol{x}_{d_1}) := \sum_{k_1}^{K_1} & p(z_{d_1} = k_1 \mid \{ r_{d_1d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}) \\ & \mathbb{E}_{q(\boldsymbol{\eta})}[\, F(y, \boldsymbol{x}_{d_1}; \boldsymbol{\eta}_{k_1}) \,], \end{aligned} \tag{4.4}$$

and $F(y, \boldsymbol{x}_{d_1}; \boldsymbol{\eta}_{k_1})$ is the discriminant function in a multi-class kernel machine [17], which computes the score of classifying feature vector $\boldsymbol{x}_{d_1}$ to label y by prediction model $\boldsymbol{\eta}_{k_1}$. We use *Maximum Entropy Discrimination* (*MED*) [50] to treat the multi-class kernel machine in a probabilistic way and MED infers posterior distribution of the prediction model $q(\boldsymbol{\eta})$. Since the prediction model is obtained as a distribution, MED computes the expectation of discriminant function over the posterior distribution, as $\mathbb{E}_{q(\boldsymbol{\eta})}[\, F(y, \boldsymbol{x}_{d_1}; \boldsymbol{\eta}_{k_1}) \,]$.

What is noteworthy in Equation (4.4) is the gating function

$$p(z_{d_1} = k_1 \mid \{ r_{d_1d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}),$$

[1]Note that R-iSVM can support more than two entity sets by extending a relationship matrix (e.g., customers $\times$ items) to a tensor (e.g., customers $\times$ items $\times$ stores).

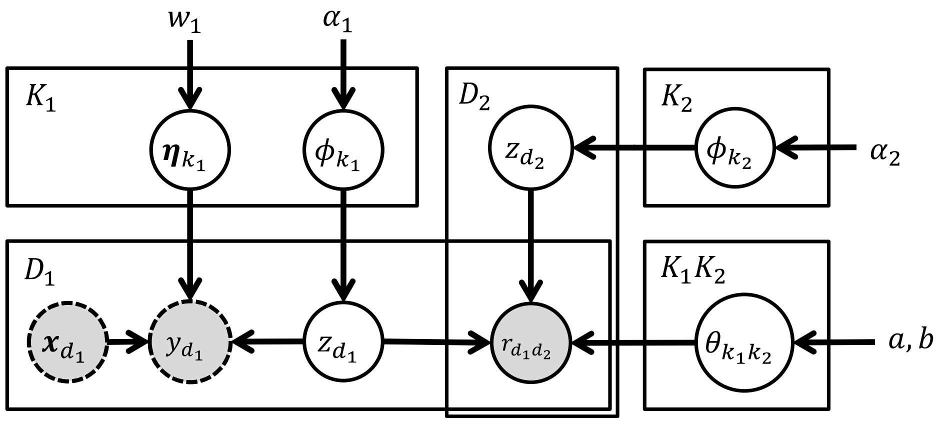


Fig. 4.3 Graphical representation of R-iSVM for two entity sets and one relationship matrix. Dirichlet processes for users/item clusters are formulated by truncated SBPs with truncation level K_1 and K_2, respectively.

which selects experts for customer d_1 in accordance with his purchase histories $\{ r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}$. As mentioned before, this behavior is different from the ordinary mixture-of-experts models that use the customer demographics $\boldsymbol{x}_{d_1}$ for expert-selection. If we model customer demographics in a probabilistic manner [99], we can define a gating function

$$p(z_{d_1} = k_1 \mid \boldsymbol{x}_{d_1}, \{ r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}),$$

which uses both customer demographics and his purchase histories in expert-selection.

4.3.3 Generative Process

Figure 4.3 shows a graphical model of R-iSVM. The generative process of R-iSVM can be represented as follows:

$$
\begin{aligned}
v_{k_1}|\alpha_1 &\sim \text{Beta}(1, \alpha_1) && (4.5)\\
\phi_{k_1} &= v_{k_1} \prod_{k_1'=1}^{k_1-1} (1 - v_{k_1'}) && (4.6)\\
v_{k_2}|\alpha_2 &\sim \text{Beta}(1, \alpha_2) && (4.7)\\
\phi_{k_2} &= v_{k_2} \prod_{k_2'=1}^{k_2-1} (1 - v_{k_2'}) && (4.8)\\
\theta_{k_1 k_2} &\sim \text{Beta}(a, b), && (4.9)\\
z_{d_1} \mid \boldsymbol{\phi}_1 &\sim \text{Multinomial}(\boldsymbol{\phi}_1), && (4.10)\\
z_{d_2} \mid \boldsymbol{\phi}_2 &\sim \text{Multinomial}(\boldsymbol{\phi}_2), && (4.11)\\
r_{d_1 d_2} \mid \boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta} &\sim \text{Bernoulli}(\theta_{z_{d_1} z_{d_2}}), && (4.12)\\
\boldsymbol{\eta}_{k_1} \mid \boldsymbol{w}_1 &\sim \mathcal{N}(\boldsymbol{w}_1, I). && (4.13)
\end{aligned}
$$

First, for each entity set (customers or items), cluster mixing parameter $\boldsymbol{\phi}$ is drawn from a Dirichlet Process (DP) with concentration parameter α (Equation (4.5) to (4.8)).[2] Next, for each combination of customer cluster k_1 and item cluster k_2, its cluster relationship strength $\theta_{k_1 k_2}$ is drawn (Equation (4.9)). Then, for all customers and items, cluster assignments z are drawn (Equation (4.10) to (4.11)). After that, for all combinations of customers and items, purchase information $r_{d_1 d_2}$ is drawn in accordance with cluster assignments z and cluster relationship strength $\boldsymbol{\theta}$ (Equation (4.12)). Finally, for each customer cluster k_1, the demographics prediction model $\boldsymbol{\eta}_{k_1}$ is constructed by a multi-class kernel machine [17] with MED [50] with a Gaussian prior (Equation (4.13)). MED makes the training of a multi-class kernel machine into a posterior distribution inference of $\boldsymbol{\eta}_k$. We elaborate on this discussion in Section 4.4.

As in *Infinite Relational Model* (IRM) [61], Equations (4.5) to (4.12) model a generative process of customer-item purchase histories. In R-iSVM, this process defines a gating function $p(z_{d_1} = k_1 \mid \{ r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \})$, which computes the weight of k_1-th expert (prediction model) for the customer d_1 from his behavioral data $\{ r_{d_1 d_2} : (i, d_2) \in \mathcal{I} \wedge i = d_1 \}$.

[2] To develop an efficient variational inference algorithm, we use stick-breaking process (SBP) [86] to formulate Dirichlet Process instead of Chines Restaurant Process (CRP).

4.4 Inference

In this section, we introduce the inference algorithm of our proposed model based on variational EM-algorithm and quadratic programming.

4.4.1 Bayesian Inference with Discriminative Model

By using the result by Zeller [101], ordinary Bayesian inference, which finds the posterior distribution of the parameters, can be formalized as the optimization problem:

$$\begin{aligned}\min_{q \in \mathcal{Q}} \; & \mathrm{KL}(q(\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \,||\, p(\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)) \\ & - \mathbb{E}_q[\, \log p(\{\, r_{d_1 d_2} : (d_1, d_2) \in \mathcal{I} \,\} \mid \boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)\,] \end{aligned} \tag{4.14}$$

where $\mathcal{Q}$ is the space of the valid probability distributions. Since it is in the optimization form, it can incorporate other optimization problems as constraints to the posterior distribution. This technique is called a *Regularized Bayes* and is recently actively studied [108].

To incorporate a multi-class kernel machine [17] into a co-clustering model, we use *Maximum Entropy Discrimination* (MED) [50], which formulates the training of discriminative model in a probabilistic manner. With MED, the training of a multi-class kernel machine can be formalized as follows:

$$\begin{aligned} \min_{q \in \mathcal{Q}, \boldsymbol{\xi}} \; & \mathrm{KL}(q(\boldsymbol{\eta}_1) \,||\, p(\boldsymbol{\eta}_1)) + C_1 \sum_{d_1}^{D_1} \xi_{d_1} \\ \text{s.t.} \quad & \forall d_1 \in D_1, \forall y_1 \in Y_1 : \\ & \quad l^{\Delta}_{d_1}(y_1) - \mathbb{E}_q[\, \boldsymbol{\eta}_1 \,]^{\top} f^{\Delta}_{d_1}(y_1) \leq \xi_{d_1}, \xi_{d_1} \geq 0, \end{aligned} \tag{4.15}$$

where $\boldsymbol{\eta}_1$ is the hyperplane of a multi-class support vector machine that is treated as a random variable, C_1 is a cost parameter, $\xi_{d_1} \in \boldsymbol{\xi}_1$ is the slack variable of entity d_1, $l^{\Delta}_{d_1}(y_1) := I(y_1 \neq y_{d_1})$ is a label-loss function that returns 1 if y_1 is not equal to the true class label y_{d_1}, and $f^{\Delta}_{d_1} : Y_1 \to \mathbb{R}^{|Y_1| M_1}$ is a feature mapping function that returns input feature vector of d_1 regarding it as class y_1. Solving this optimization problem results in different algorithms depends on the choice of prior distribution $p(\boldsymbol{\eta}_1)$: Gaussian prior results in ℓ_2-SVM and Laplace prior results in ℓ_1-SVM [107]. In this chapter, we use Gaussian prior as shown in Equation (4.13).

Problem Definition

By unifying the Problem 4.14 and 4.15, we get the objective of R-iSVM as follows:

$$\begin{aligned}
&\min_{q \in \mathcal{Q}, \boldsymbol{\xi}} \ \mathrm{KL}(q(\boldsymbol{z}_1, \boldsymbol{\eta}_1) \mid\mid p(\boldsymbol{z}_1, \boldsymbol{\eta}_1)) + C_1 \sum_{d_1}^{D_1} \xi_{d_1} \\
&+ C_2 \Big\{ \mathrm{KL}(q(\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \mid\mid p(\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)) \\
&\qquad - \mathbb{E}_q[\log p(\{ r_{d_1 d_2} : (d_1, d_2) \in \mathcal{I} \} \mid \boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2)] \Big\} \\
&\text{s.t.} \quad \forall d_1 \in D_1, \forall y_1 \in Y_1 : \\
&\quad l_{d_1}^{\Delta}(y_1) - \mathbb{E}_q[\boldsymbol{\eta}_{z_{d_1}}]^{\top} f_{d_1}^{\Delta}(y_1) \leq \xi_{d_1}, \xi_{d_1} \geq 0
\end{aligned} \tag{4.16}$$

where C_2 is a hyperparameter that controls the effect of the discriminative model on the Bayesian inference.

Problem 4.16 shows two coupled tasks in R-iSVM: (1) the third term corresponds to the co-clustering of customers and items, and (2) the other terms and constraints correspond to the training of the demographics prediction model at each customer cluster. Since the latent variables $\boldsymbol{z}_1$ appear in both objectives, both tasks affect each other to improve the quality of their tasks: co-clustering results are adjusted to improve the performance of the customer demographics prediction.

4.4.2 Variational Inference

Since directly solving the Problem 4.16 is intractable, we impose the *mean-field approximation* on the posterior distribution q as in ordinary *variational inference*:

$$\begin{aligned}
q(\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2) = &\prod_{d_1}^{D_1} q(z_{d_1}) \prod_{d_2}^{D_2} q(z_{d_2}) \\
&\prod_{k_1}^{K_1} \prod_{k_2}^{K_2} q(\theta_{k_1 k_2}) \prod_{k_1}^{K_1} q(\phi_{k_1}) \prod_{k_2}^{K_2} q(\phi_{k_2}),
\end{aligned} \tag{4.17}$$

and

$$q(\boldsymbol{z}_1, \boldsymbol{\eta}_1) = \prod_{d_1}^{D_1} q(z_{d_1}) \prod_{k_1}^{K_1} q(\boldsymbol{\eta}_{k_1}). \tag{4.18}$$

Then, we use the *Lagrangian method* and *coordinate ascent* method to minimize the objective of Problem 4.16. Alternatively maximizing the dual form of Problem 4.16 by

Algorithm 3: Inference Algorithm

```
while not converged do
    begin (QP-Step) Solve SVM's dual QP:
        Solve QP Equation (4.24) to obtain Lagrangians.
        for $k_1 \in [K_1]$ do
            Update $q(\eta_{k_1})$ by Equation (4.26).
    begin (VB E-Step) Update latent variables:
        for $d_1 \in [D_1]$ do
            for $k_1 \in [K_1]$ do
                Update $q(z_{d_1} = k_1)$ by Equation (4.23).
        for $d_2 \in [D_2]$ do
            for $k_2 \in [K_2]$ do
                Update $q(z_{d_2} = k_2)$.
    Reorder clusters in descending order of size [65].
    begin (VB M-Step) Update parameters:
        for $k_1 \in [K_1]$ do
            Update $q(v_{k_1})$ by Equation (4.20).
        for $k_2 \in [K_2]$ do
            Update $q(v_{k_2})$.
        for $(k_1, k_2) \in [K_1] \times [K_2]$ do
            Update $q(\theta_{k_1 k_2})$ by Equation (4.22).
```

random variables and slack variables results in an iterative inference algorithm shown in Algorithm 3.

In the remainder of this section, we elaborate on the derivation of the each step.

Model Parameters

For random variables other than $\boldsymbol{\eta}_1$ and $\boldsymbol{z}_1$, we obtain ordinary variational posteriors of IRM [99, 49] as follows: [3] The variational posterior of the parameter of the *stick-breaking process* is

$$q(v_{k_1}) = \mathrm{Beta}(\hat{\alpha}_{k_1}, \hat{\beta}_{k_1}), \tag{4.19}$$

[3]We omit variational posteriors for second entity sets (items), since they have the same form with the first entity sets (customers).

and its update is

$$\hat{\alpha}_{k_1} = 1 + \sum_{d_1}^{D_1} q(z_{d_1} = k_1)$$

$$\hat{\beta}_{k_1} = \alpha_1 + \sum_{d_1}^{D_1} \sum_{k'_1=k_1+1}^{K_1} q(z_{d_1} = k'_1). \tag{4.20}$$

The variational posterior of the relationship strength is

$$q(\theta_{k_1k_2}) = \mathrm{Beta}(\hat{a}_{k_1k_2}, \hat{b}_{k_1k_2}) \tag{4.21}$$

and its update is

$$\hat{a}_{k_1k_2} = a + \sum_{(d_1,d_2)\in\mathcal{I}} q(z_{d_1} = k_1) q(z_{d_2} = k_2) r_{d_1d_2}$$

$$\hat{b}_{k_1k_2} = b + \sum_{(d_1,d_2)\in\mathcal{I}} q(z_{d_1} = k_1) q(z_{d_2} = k_2)(1 - r_{d_1d_2}). \tag{4.22}$$

Latent Variables

Optimizing the dual form of Problem 4.16 by $q(z_{d_1} = k_1)$, we obtain the variational posterior of the latent variable as follows:

$$\begin{aligned}
&\log q(z_{d_1} = k_1) \propto \\
&\quad \mathbb{E}_q[\,\log p(z_{d_1} = k_1 | v_{k_1})\,] \\
&\quad + \rho \sum_{d_2\in\mathcal{I}_{d_1}} \sum_{k_2}^{K_2} q(z_{d_2} = k_2) \mathbb{E}_q[\,\log p(r_{d_1d_2} | \theta_{k_1k_2})\,] \\
&\quad + (1-\rho) \left\{ \sum_{y_1}^{Y_1} \omega_{y_1d_1} \mathbb{E}_q[\,\boldsymbol{\eta}_{k_1}\,]^\top f_{d_1}^{\Delta}(y_1) \right\},
\end{aligned} \tag{4.23}$$

where $\mathcal{I}_{d_1} = \{\, d_2 : (i, d_2) \in \mathcal{I} \wedge i = d_1 \,\}$, $\rho = C_2/(1 + C_2)$, and $\omega_{y_1d_1}$ is a Lagrangian variable.

In Equation (4.23), the first and second terms are the same as with ordinary IRM [99, 49]. What is noteworthy here is the third term, which is the expected demographics prediction score of customer d_1, indicating the *goodness* of cluster k_1 for the customer d_1.

Through incorporating such discriminative prediction scores into the posterior inference, R-iSVM adjusts clustering results to reduce the training error.

4.4.3 Quadratic Programming

Optimizing the dual form of Problem 4.16 by the slack variables $\boldsymbol{\xi}$, we obtain a quadratic programming

$$\begin{aligned} \max_{\boldsymbol{\omega}} \quad & -\frac{1}{2}\sum_{k_1}^{K_1} \hat{\mu}_{k_1}^\top \hat{\mu}_{k_1} + \sum_{y_1}^{Y_1}\sum_{d_1}^{D_1} \omega_{y_1 d_1} l_{d_1}^{\Delta}(y_1) \\ \text{s.t.} \quad & \forall d_1 : 0 \leq \sum_{y_1}^{Y_1} \omega_{y_1 d_1} \leq C_1, \end{aligned} \tag{4.24}$$

where $l_{d_1}^{\Delta}(y_1) := I(y_1 \neq y_{d_1})$ is a label-loss function that returns 1 if y_1 is not equal the true class label y_{d_1}.

Optimizing the dual form of Problem 4.16 by the hyperplane of a multi-class kernel machine $\boldsymbol{\eta}_{k_1}$, we obtain the update equation of its variational posterior distribution

$$q(\boldsymbol{\eta}_{k_1}) = \mathcal{N}(\hat{\mu}_{k_1}, I) \tag{4.25}$$

as

$$\hat{\mu}_{k_1} = w_1 + \sum_{d_1}^{D_1} q(z_{d_1} = k_1) \sum_{y_1}^{Y_1} \omega_{y_1 d_1} f_{d_1}^{\Delta}(y_1). \tag{4.26}$$

As shown in Algorithm 3, we solve the *quadratic programming* Problem (4.24) inside our variational-EM algorithm to adjust the clustering results. By using a relaxation technique that decomposes Problem (4.24) into K_1 sub problems [107], the quadratic programming can be solved by ordinary SVM solvers, such as a linear-time *one-slack cutting plane solver* [54].

4.4.4 Computational Complexity

Algorithm 3 is scalable because its computational complexity is linear in the number of observed relationships $|\mathcal{I}|$, which is often much smaller than the number of possible relationships $|D_1||D_2|$ in real-world data. As shown in Algorithm 3, variational inference of co-clustering runs in $O(K_1 K_2 |\mathcal{I}|)$, given K_1 customer clusters, K_2 item clusters, and $\mathcal{I}$ observed relationships. For quadratic programming of a multi-class kernel machines, we have K_1 experts for customers, and each expert can be learned by a linear-time one-slack cutting plane algorithm [54], the computational complexity of which is $O(M_1|D_1|)$, where M_1 is the dimension of explanatory variable in the first entity set's attribute prediction. Thus, the final computational complexity of our algorithm is $O(K_1 K_2 |\mathcal{I}| + K_1 M_1 |D_1|)$, which is scalable.

Furthermore, for loops of variational E-Steps and M-Steps in Algorithm 3 are fully-parallelizable, which plays an important role in practice. We use *OpenMP* to parallelize the for loops in our R-iSVM implementation.

4.4.5 Convergence

Convergence of Algorithm 3 is guaranteed. As mentioned in Section 4.4.2, Algorithm 3 is a *coordinate ascent* maximization of the dual form of Problem 4.16, which alternately maximizes the objective by each variable. Coordinate ascent is guaranteed to converge to a *local optimum*, because each step (such as QP-, E-, and M-steps in Algorithm 3) maximizes the objective with respect to a variable and does not decrease the objective value [9].

4.5 Experiments

In this section, we evaluate the proposed R-iSVM on real datasets, and demonstrate its effectiveness. Through the experiments, we attempted to confirm that R-iSVM satisfies our four design goals:

- **(Generality)** R-iSVM does not require application specific modeling. To confirm this characteristic, we evaluate three real-world datasets in two domains: real retailers and online movie review site.
- **(Interpretability)** R-iSVM does not produce hard-to-interpret features. Furthermore, it provides additional information that helps people interpret prediction results. To confirm this characteristic, we prepare visualizations that help interpretation of the model.
- **(Accuracy)** R-iSVM improves entity attribute predictive performance by adopting relationship data in a mixture-of-experts manner. Further, *it can co-exist with conventional feature engineering or representation learning methods*, and helps to improve their predictive performances.

Table 4.3 Datasets used in experiments.

Dataset	Relationship type	Entity attributes	$\lvert D_1\rvert$	$\lvert D_2\rvert$	$\lvert\mathcal{I}\rvert$	Density
MovieLens 1M	user $\times$ movie (5-star review)	gender, age, occupation	6,040	3,900	1,000,209	4.24%
Ta-Feng	user $\times$ item (purchase history)	age, resident	16,578	17,860	227,827	0.07%
BeiRen	user $\times$ item (purchase history)	gender, age, marital, income, education	57,693	61,097	6,396,551	0.18%

- **(Scalability)** The computational complexity of R-iSVM training is linear in the number of observed relationships in entity-relationship data. Moreover, it can be fully parallelized.

4.5.1 Setup

Datasets

We used three real-world datasets in our experiments (details are summarized in Table 4.3):

(MovieLens) The first is the MovieLens 1M dataset [39], which comes from an online movie review website[4]. The dataset contains $1,000,209$ ratings ($\mathcal{I}$) for $6,040$ users (D_1) on $3,900$ movies (D_2). Each user has three demographics: **gender** (male/female), **age** (categorized into 7 ranges), and **occupation** (21 types). Since these attributes are categorical, we convert them into dummy variables, producing 31 dimensional vector for each user ($|x_{d_1}| = 31$). We also convert five-star ratings into a binary value if the rating is higher than the user's average rating, and vice versa.

(Ta-Feng) The second is the Ta-Feng dataset[5], which comes from a real supermarket in China. The dataset contains transactions collected by a supermarket from November 2000 to February 2001. We picked out records of November 2000, which contains $16,578$ users (D_1), $17,860$ items (D_2), and $6,396,551$ purchase histories ($\mathcal{I}$). Each user has two demographic attributes: **age** (categorized into 10 ranges) and **residence** (8 areas). We convert these categorical attributes into dummy variables as in MovieLens dataset.

(BeiRen) The third is the BeiRen dataset[6], which comes from a real retailer in China. The dataset contains $57,693$ users (D_1), $61,097$ items (D_2), and $6,396,551$ purchase histories ($\mathcal{I}$). Each user has five demographic attributes: **gender** (male/female), **age** (categorized into four ranges), **marital status** (single/married), **income** (categorized into four ranges), and **education level** (six levels). We convert these categorical attributes into dummy variables as in MovieLens dataset.

[4]We chose MovieLens 1M because larger MovieLens datasets such as MovieLens 10M do not have user demographics.

[5]http://recsyswiki.com/wiki/Grocery_shopping_datasets

[6]http://www.bigdatalab.ac.cn/benchmark/bm/bd?code=SNE

Compared Methods

We compared four methods in terms of entity attribute prediction:

- **(MC)** A multi-class kernel machine [17]. For the implementation, we used a Python implementation of multi-class support vector machine in PyStruct package [74].
- **(DA+MC)** Demographics augmentation by *Tucker decomposition* of behavioral data [105]. Latent feature vectors are retrieved from Tucker decomposition of behavioral data and used to augment original basic demographics. Then, we use the augmented features to train a prediction model with **MC**.
- **(R-iSVM)** Our proposed mixture-of-experts model with basic demographics.
- **(DA+R-iSVM)** Our proposed model with user demographics augmentation as in **DA+MC**. Before running R-iSVM, user demographics are augmented as in **DA+MC**, and then R-iSVM constructs a mixture-of-experts model.

Tasks and Parameters

For each combination of a method and a prediction target demographics, we repeated nested five-fold cross validation five times, and computed the average and standard deviation of F1-micro scores.

For hyperparameter selection, we used two parameter grids in grid-search:

- $C_1 = \{0.1, 1, 10\}$ for SVM's hyperparameter (in MC, DA+MC, R-iSVM, and DA+R-iSVM).
- $C_2 = \{1, 64\}$ for R-iSVM's (in R-iSVM and DA+R-iSVM). Note that for other hyperparemeters in R-iSVM that come from a Bayesian part (namely, α, a, and b), we took empirical Bayes approach to learn hyperparameter values from data, and no grid-search was needed.

4.5.2 Predictive Performance

Table 4.4 shows average F1-micro scores of compared methods in each task, together with their standard deviations. By comparing the predictive performance between MC and R-iSVM, we can observe that R-iSVM successfully increased predictive performances without producing hard-to-interpret features. By comparing DA+MC (demographics

Table 4.4 (Accuracy) Predictive performance of demographics prediction. Each value represents average the F1-micro score of five trials with standard deviation. For all tasks, R-iSVM (with or without demographics augmentation) achieved the highest performance.

Dataset	**MovieLens 1M**			**Ta-Feng**		**BeiRen**				
Method	gender	age	occupation	age	residence	gender	age	education	marital	income
MC	62.7 ± 1.5	14.9 ± 3.4	9.5 ± 0.9	10.7 ± 0.7	14.0 ± 2.0	63.7 ± 0.7	56.2 ± 1.0	18.8 ± 4.2	26.9 ± 5.6	24.9 ± 8.0
DA+MC	69.4 ± 1.3	22.6 ± 0.7	11.1 ± 0.7	12.5 ± 2.9	14.9 ± 3.7	63.7 ± 0.4	56.2 ± 0.6	20.4 ± 2.3	26.0 ± 1.9	26.4 ± 1.1
R-iSVM*	69.1 ± 3.2	22.8 ± 1.9	14.2 ± 2.4	12.9 ± 2.3	20.1 ± 3.5	$\mathbf{66.1 \pm 3.2}$	$\mathbf{61.5 \pm 2.8}$	27.2 ± 4.4	$\mathbf{41.1 \pm 4.8}$	36.7 ± 4.6
DA+R-iSVM*	$\mathbf{73.7 \pm 1.2}$	$\mathbf{30.3 \pm 1.8}$	$\mathbf{15.3 \pm 0.8}$	$\mathbf{15.3 \pm 1.6}$	$\mathbf{23.9 \pm 1.9}$	64.4 ± 3.2	58.4 ± 2.0	$\mathbf{27.9 \pm 3.0}$	41.0 ± 1.3	36.8 ± 2.4

augmentation approach) and DA+R-iSVM, we can observe that our proposed model, R-iSVM, can co-exist with state-of-the-art demographics augmentation and can further improve the performance.

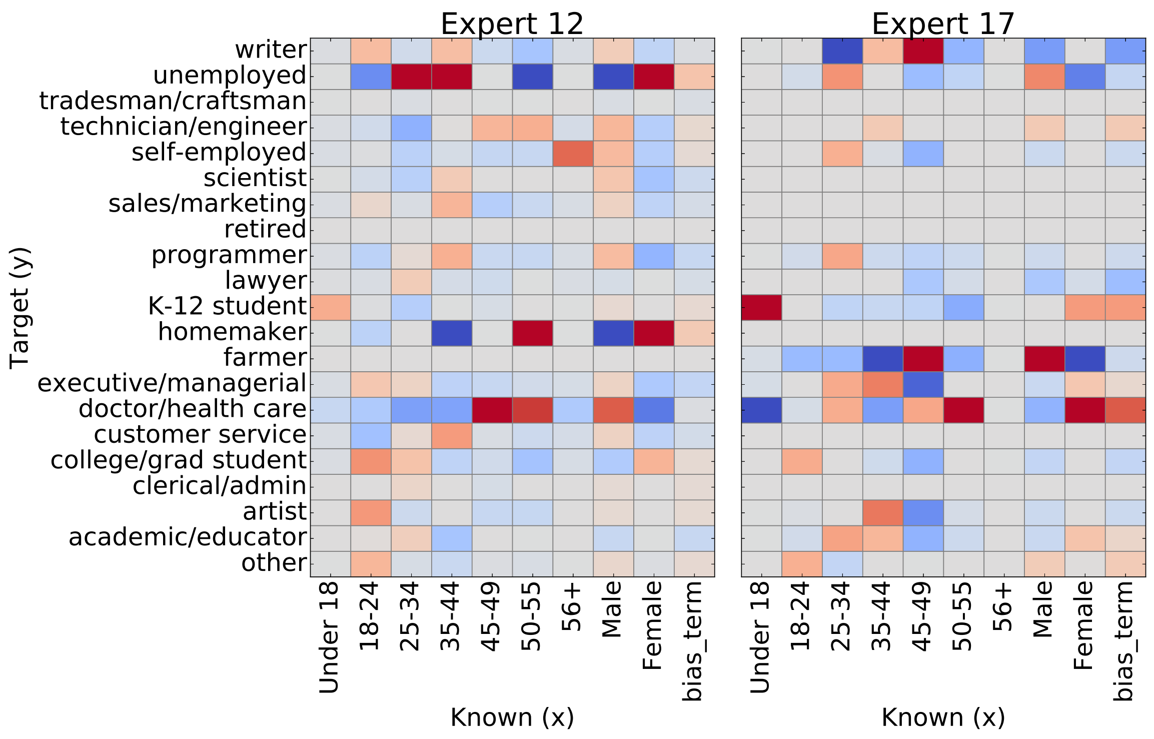


Fig. 4.4 (Interpretability) Visual representation of experts in R-iSVM on MovieLens 1M dataset. We can check effects of original features on classification results at a glance. For models with large number of input feature and classes, we can limit the visualization to prominent parts.

4.5.3 Interpretability

Next, we show the interpretability of R-iSVM in details. Figure 4.4 shows a visual representation of two experts in an R-iSVM model that predicts "occupation." In the figure, a row represents a class y, a column represents a feature dimension of x, and the corresponding cell represents the positive (red) or negative (blue) impact of the feature dimension on the class. From the figure, we can find out how each feature dimension affects the classification results. For example, by checking red cells in expert 12, we can find several intuitive rules in user cluster 12, such as ***if*** *gender="Female"* ***and*** *age="50–55"* ***then*** *"homemaker"*. Further, as we mentioned before, R-iSVM offers additional information to experts thanks to the co-clustering nature of its model. Figure 4.5 shows the cluster relationship information between experts and movie clusters in the R-iSVM model, namely, the values of $q(\boldsymbol{\theta})$. We omit weak relationships from the figure and thus it only shows *heavy movie watchers* and their *movie preferences*. From Figure 4.5, we can learn that users in clusters 12, 14, and 18 are heavy movie watchers. By incorporating this knowledge into the rules obtained from the visual representation of the experts in Figure 4.4, we can further improve the interpretation. These results confirm that R-iSVM is interpretable.

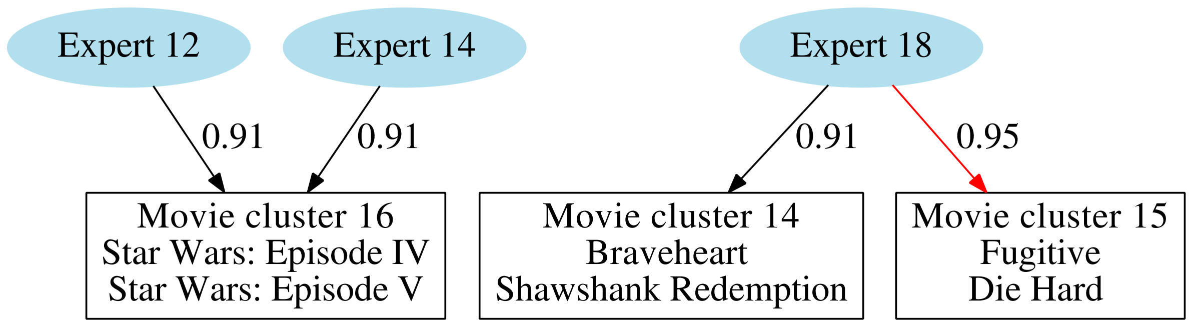


Fig. 4.5 (Interpretability) Visual representation of relevance between experts (user demographics prediction model) and movie clusters on MovieLens 1M dataset. For each expert, R-iSVM identifies relevant movie clusters, which helps to clarify experts' behavior in collaboration with visualization in Figure 4.4.

4.5.4 Scalability

Finally, we demonstrate the scalability of R-iSVM. We subsampled BeiRen dataset to construct seven different-sized datasets. Figure 4.6 shows the runtime of R-iSVM on a computer that has two Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz with 256GB DDR memory. As in Figure 4.6, the runtime of R-iSVM increases linearly with the size of observed relationships, which demonstrates that R-iSVM is scalable.

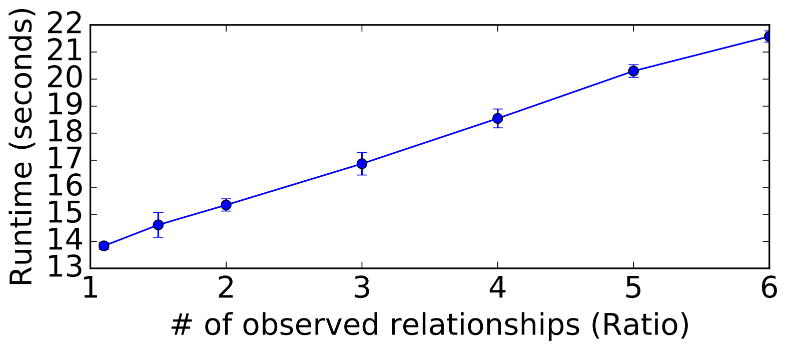


Fig. 4.6 (Scalability) R-iSVM's runtime on BeiRen dataset varying the number of observed relationships $|\mathcal{I}|$. Runtime increases linearly with the size of observed relationships $|\mathcal{I}|$, confirming computational complexity discussed in Section 4.4.4.

4.6 Related Work

Customer demographics prediction has been conducted with a wide variety of problem settings and methods [75, 44, 21, 105, 18, 97, 103]. We elaborate on two lines of work that we overviewed in the Introduction: *application-specific feature engineering* and *representation learning*. For the feature engineering approach, Culotta *et al.* proposed constructing a user's feature vector from users he follows on a social-network [18]. For representation learning, singular value decomposition (SVD) on a relationship matrix (users' behavioral data), such as web browsing history [75, 44] and location check-in [97], has achieved successes. To capture multiple types of behaviors simultaneously, Zhong *et al.* has extended relationship matrix to tensor (i.e., location check-ins and online-review), and used Tucker decomposition to obtain the latent features of users [105]. Wang *et al.* proposed using a user's purchase history in retail stores in the prediction [97]. They formulated the problem as multi-task learning that predicts multiple attributes of a user simultaneously and used a neural embedding approach to obtain a highly discriminative representation for each user. In contrast to these approaches, our design goal is to achieve high predictive performance without producing any additional hard-to-interpret features.

4.7 Summary

To solve customer demographics prediction problems, we developed a *Relational Infinite Support Vector Machine* (R-iSVM), a novel mixture-of-experts model that can leverage behavioral data. R-iSVM has four properties:

1. **(General)** R-iSVM does not require application-specific modeling, and is not even limited to demographics prediction. *It is widely applicable for general entity-relationship data.*

2. **(Interpretable)** *R-iSVM does not produce hard-to-interpret features.* Furthermore, it provides additional information that helps people interpret prediction results.

3. **(Accurate)** R-iSVM improves entity attribute predictive performance by adopting relationship data in a mixture-of-experts manner. Further, *it can co-exist with conventional feature engineering or representation learning methods*, and helps to improve predictive performance of these methods.

4. **(Scalable)** The computational complexity of R-iSVM training is linear in the number of observed relationships in entity-relationship data. Moreover, it can be fully-parallelized.

We have evaluated R-iSVM on various real world datasets including retail store data and online movie-review data. The experimental results demonstrated its generality, interpretability, accuracy, and scalability.

Chapter 5

Fast and Space-Efficient Machine Learning with Data Compression

As mentioned in Section 1, machine learning on relational data has three bottleneck phases, namely "feature-engineering", "model training", and "model evaluation". In the previous two sections, we have focused on the first phase, "feature-engineering", by accelerating the aggregation operation in relational database systems (Section 3) and designing a machine learning model that does not require feature-construction (Section 4). In this section, we tackle the second and third phase, "model training" and "model evaluation", by presenting a way to reduce the computational time of the training and prediction of machine learning models using data compression technique.

5.1 Introduction

Edge devices, such as sensors and mobile devices, are prevalent in our life. To conduct data mining tasks on edge devices in a real-time manner, it is necessary to conduct the tasks in the edge devices themselves, instead of delegating the tasks to the powerful remote computers. However, since edge devices have relatively small memory footprints and low computational power, the applicability of edge devices to the data mining tasks is limited. Motivated by this problem, several studies have tried to reduce the memory usage and the computational time for specific tasks or specific data types [30, 36, 42].

For sparse matrix data that is mostly filled with zero values, *sparse matrix representation* [30, 36], which only holds non-zero value and its position, can concisely represent the matrix and reduce the cost of some mathematical operations. However, actual data is not always sparse but can be dense. Since storing dense data with a sparse matrix format can increase the size, sparse matrix representation cannot be used in a versatile way.

In this chapter, we present *CVS (Compressed Vector Set)*, a general framework for fast and space-efficient data mining, which successfully supports both sparse and dense datasets. CVS compresses vector sets by *run-length encoding* and conducts fundamental mathematical operations on them without decompression. By combining fundamental mathematical operations, CVS runs advanced data mining algorithms such as k-nearest neighbor search, stochastic gradient descent on logistic regression, and kernel methods. To reduce the size of compressed vectors as much as possible, CVS (1) reorders the dimensions of vectors if the mathematical operations are dimension-order insensitive, and (2) discretizes vectors if the result of data mining algorithms is less affected by the precision of values in vectors.

We summarize our contributions:

1. We present algorithms to conduct advanced data mining tasks on vectors compressed by run-length encoding (Section 5.3 and section 5.4).

2. We observe that reordering dimensions of the vectors further reduce the compression size without changing computational results. Based on the observation, we tackle the dimension-order reordering problem that finds best reordering pattern, which is NP-hard. We show a polynomial time algorithm for finding a dimension-reordering pattern that empirically yields good compression rate (Section 5.5).

3. We demonstrate the combination of data discretization and dimension-reordering can drastically improve the performance without much affecting the accuracy of data mining tasks (Section 5.6).

4. We demonstrate the effectiveness of CVS compared to the conventional sparse vector representation in data mining and machine learning tasks on real datasets (Section 5.7).

The rest of this chapter is organized as follows. Section 5.3 describes how CVS compresses and conduct mathematical operations on a set of vectors. Section 5.3 shows how CVS conducts concrete data mining and machine learning tasks on compressed vectors without decompression. Section 5.5 describes how CVS improves the compression rate with dimension-reordering. Section 5.5 describes how CVS improves the compression rate with data discretization. Section 5.7 evaluates CVS by experiments with various datasets. Section 5.8 describes related work. Section 5.9 concludes the chapter.

5.2 Preliminaries

In this section, we review concepts that relate to Compressed Vector Set (CVS): sparse vector representation and run-length encoding.

5.2.1 Sparse Vector Representation

Sparse matrix representations [30, 36] represent a vector that is mostly filled with zero values in a space-efficient way. One of the most commonly-used representations is to represent a vector with non-zero values and its positions. For example, a sparse vector

$$\boldsymbol{x} = (0, 0, 42, 99, 0, 0, 0)$$

is represented by two vectors

$$\begin{aligned} \text{values} &= (42, 99) \\ \text{positions} &= (2, 3) \end{aligned}$$

in this format.

While the sparse vector representation reduces the space-efficiency, they also reduce the computational complexity of several mathematical operations. For example, the dot product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, which are comprised of B_x and B_y non-zero elements, can be carried out in $\mathcal{O}(B_x + B_y)$ time.

One downside of the sparse vector representation is its inefficiency for dense data. For example, a dense vector

$$\boldsymbol{x} = (7, 7, 42, 99, 7, 7, 7)$$

is represented by

$$\begin{aligned} \text{values} &= (7, 7, 42, 99, 7, 7, 7) \\ \text{positions} &= (0, 1, 2, 3, 4, 5, 6) \end{aligned}$$

in the sparse vector representation, which doubled the size compared to the original vector. In contrast, our proposed CVS can represent a dense data efficiently when the number of distinct elements is not large. We compare the performance of sparse vector representation and CVS empirically in Section 5.7.

5.2.2 Run-length Encoding

Run-length encoding (*RLE*) [6] is a data compression technique that represents a sequence of n same consecutive values $x, \ldots, x$ with a block $\langle n, x \rangle$. For example, RLE represents $(1,1,1,1,1,2,2,2,2,2,2,2,2,2,1,1,1)$ as $(\langle 5,1 \rangle, \langle 9,2 \rangle, \langle 3,1 \rangle)$, reducing the number of elements to represent the data from 17 to 6. In this chapter, we denote the RLE-compressed form of the vector $\boldsymbol{x}$ as

$$\mathrm{RLE}(\boldsymbol{x}) = (\langle n_1, x_1 \rangle, \ldots, \langle n_B, x_B \rangle),$$

where B is the total number of blocks and n_b is the number of values in the b-th block. We also use $|\mathrm{RLE}(\boldsymbol{x})|$ to represent the total number of blocks of vector $\boldsymbol{x}$.

When the input vector $\boldsymbol{x}$ does not contain many same consecutive values , RLE cannot efficiently encode the data and sometimes increase the size; RLE represents $(1,2,3,4,1,1,1,1)$ as $(\langle 1,1 \rangle, \langle 1,2 \rangle, \langle 1,3 \rangle, \langle 1,4 \rangle, \langle 4,1 \rangle)$, which increased the data size. *PackBits* [6] is a simple but effective method to alleviate this problem, which has two encoding rules: (1) ordinary RLE encoding and (2) raw data encoding. (1) When PackBits finds a sequence of n same consecutive values $x, \ldots, x$, it does RLE encoding: replaces the sequence with a block $\langle n, x \rangle$. (2) Otherwise, PackBits is facing a sequence of n consecutive different values $x_1, \ldots, x_n$, and PackBits does raw data encoding: replaces the sequence by $\langle -n, x_1, \ldots, x_n \rangle$. For example, PackBits encodes a data sequence $(1,2,3,4,1,1,1,1)$ to $(\langle -4,1,2,3,4 \rangle, \langle 4,1 \rangle)$ successfully reducing the data size, whereas RLE encodes the sequence to $(\langle 1,1 \rangle, \langle 1,2 \rangle, \langle 1,3 \rangle, \langle 1,4 \rangle, \langle 4,1 \rangle)$ increasing the data size.

5.3 Compressed Vector Set

In this section, we describe the idea of Compressed Vector Set (CVS), which compresses a vector set with run-length encoding to save the storage space, and conducts mathematical operations on the set of vectors without decompression reducing the computational time.

Table 5.1 List of symbols.

Symbol	Description
$\boldsymbol{x}$	A vector ($\boldsymbol{x} \in \mathbb{R}^D$)
$\mathcal{X}$	A vector set ($\mathcal{X} = \{\, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_J \,\}$)
J	Number of vectors in a vector set
X	Vector set as a matrix ($X \in \mathbb{R}^{D \times J}$)
D	Dimension of a vector
$\mathrm{RLE}(\boldsymbol{x})$	A compressed form of a vector $\boldsymbol{x}$
$\langle n, x \rangle$	A block in a compressed vector (n-consecutive x)
B	Number of blocks in a compressed vector

5.3.1 Vector Compression

Given a set of vectors $\mathcal{X} = \{\, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5 \,\}$, where

$$\begin{aligned}
\boldsymbol{x}_1 &= (1, 1, 2, 2, 2),\\
\boldsymbol{x}_2 &= (2, 2, 1, 2, 2),\\
\boldsymbol{x}_3 &= (2, 3, 3, 3, 3),\\
\boldsymbol{x}_4 &= (2, 3, 3, 2, 2),\\
\boldsymbol{x}_5 &= (2, 3, 1, 2, 2),
\end{aligned}$$

CVS compresses each vector in $\mathcal{X}$ using RLE/PackBits:

$$\begin{aligned}
\mathrm{RLE}(\boldsymbol{x}_1) &= (\langle 2, 1 \rangle, \langle 3, 2 \rangle),\\
\mathrm{RLE}(\boldsymbol{x}_2) &= (\langle 2, 2 \rangle, \langle 1, 1 \rangle, \langle 2, 2 \rangle),\\
\mathrm{RLE}(\boldsymbol{x}_3) &= (\langle 1, 2 \rangle, \langle 4, 3 \rangle),\\
\mathrm{RLE}(\boldsymbol{x}_4) &= (\langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 2, 2 \rangle),\\
\mathrm{RLE}(\boldsymbol{x}_5) &= (\langle -3, 2, 3, 1 \rangle, \langle 2, 2 \rangle).
\end{aligned}$$

5.3.2 Operation without Decompression

CVS conducts mathematical operations on compressed vectors directly without decompressing the vectors, reducing both the computational time and the memory usage. The key insight under the technique is that an RLE-encoded vector contains preliminary knowledge that *value x appears n times consecutively*. In the rest of this section, we demonstrate how CVS utilizes this knowledge to perform actual mathematical operations without decompression.

ℓ_p-norm

First, we demonstrate how CVS computes ℓ_p-norm of a compressed vector. ℓ_p-norm is an essential statistics, which appears in many complex operations including a cosine similarity and mathematical optimization algorithms [25].

Consider ℓ_2-norm (Euculidian-norm) of a vector

$$\boldsymbol{x} = (1, 2, 2, 2, 2).$$

Naïvely, ℓ_2-norm of $\boldsymbol{x}$ is computed as

$$\|\boldsymbol{x}\|_2 = \sqrt{1^2 + 2^2 + 2^2 + 2^2 + 2^2}, \tag{5.1}$$

which requires five multiplications and four additions inside the square root.

In CVS, we have

$$\mathrm{RLE}(\boldsymbol{x}) = (\langle 1, 1 \rangle, \langle 4, 2 \rangle),$$

and ℓ_2-norm of $\boldsymbol{x}$ is computed as

$$\|\boldsymbol{x}\|_2 = \sqrt{1^2 + 4 \times 2^2}, \tag{5.2}$$

which only needs three multiplications and one addition inside the square root, of which the computational cost is less than Equation (5.1), the naïve one.

CVS can compute general ℓ_p-norm without decompression. Suppose we have a vector $\boldsymbol{x}$ that can be expressed by B blocks in RLE:

$$\boldsymbol{x} = (x_1, \ldots, x_D),$$
$$\mathrm{RLE}(\boldsymbol{x}) = (\langle n_1, x_1 \rangle, \ldots, \langle n_B, x_B \rangle).$$

Naïvely, ℓ_p-norm of $\boldsymbol{x}$ is computed as

$$\|\boldsymbol{x}\|_p = \left(\sum_{d=1}^{D} x_d^p \right)^{1/p} \tag{5.3}$$

Algorithm 4: Dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$ without decompression.

```
Input: RLE($\boldsymbol{x}$) = $\langle n_1, x_1\rangle, \dots, \langle n_{B_x}, x_{B_x}\rangle$ and RLE($\boldsymbol{y}$) = $\langle n'_1, y_1\rangle, \dots, \langle n'_{B_y}, y_{B_y}\rangle$,
       where $B_x \le B_y$.
Output: $\boldsymbol{x}^\top \boldsymbol{y}$
1  $p \leftarrow 0$
2  $y_{\text{remains}} \leftarrow n'_1$
3  $j_{\text{next}} \leftarrow 0$
4  for $i \leftarrow 2$ to $B_x$ do
5      $x_{\text{remains}} \leftarrow n_i$
6      $y_{\text{sum}} \leftarrow 0$
7      for $j \leftarrow j_{next}$ to $B_y$ do
8          $y_{\text{sum}} \leftarrow y_{\text{sum}} + \min(x_{\text{remains}}, y_{\text{remains}}) \times \text{RLE}(\boldsymbol{y})[j]$
9          if $x_{remains} < y_{remains}$ then
10             $y_{\text{remains}} \leftarrow y_{\text{remains}} - x_{\text{remains}}$
11             break
12         else
13             $x_{\text{remains}} \leftarrow x_{\text{remains}} - y_{\text{remains}}$
14             $y_{\text{remains}} \leftarrow n'_{j_{\text{next}}+1}$
15             $j_{\text{next}} \leftarrow j_{\text{next}} + 1$
16     $p \leftarrow p + \text{RLE}(\boldsymbol{x})[i] \times y_{\text{sum}}$
17 return $p$
```

whereas CVS computes ℓ_p-norm by

$$\|\text{RLE}(\boldsymbol{x})\|_p = \left(\sum_{b=1}^{B} n_b x_b^p\right)^{1/p}. \tag{5.4}$$

From the Equation (5.3) and Equation (5.4), it is obvious that CVS reduces the computational complexity of ℓ_p-norm from $\mathcal{O}(D)$ to $\mathcal{O}(B)$ where D is the number of dimensions of the input vector and B is the number of blocks in the compressed vector.

Dot product

Consider the dot product of two vectors $\boldsymbol{x}_1 \in \mathbb{R}^{11}$ and $\boldsymbol{x}_2 \in \mathbb{R}^{11}$, whose compressed forms are

$$\text{RLE}(\boldsymbol{x}_1) = (\langle 5, 4\rangle, \langle 6, 8\rangle),$$
$$\text{RLE}(\boldsymbol{x}_2) = (\langle 3, 1\rangle, \langle 5, 3\rangle, \langle 3, 2\rangle).$$

Naïvely, the dot product requires eleven multiplications and ten additions. In contrast, CVS computes the dot product as

$$\underbrace{4}_{n_1} \times \underbrace{(1 \times 3 + 3 \times 2)}_{\text{1st } y_{\text{sum}}} + \underbrace{8}_{n_2} \times \underbrace{(3 \times 3 + 2 \times 3)}_{\text{2nd } y_{\text{sum}}}, \tag{5.5}$$

by using the Algorithm 4. In Equation (5.5), six multiplications and three additions are required, which is less than the naïve one. The computational complexity of the Algorithm 4 is $\mathcal{O}(B_x + B_y)$, where B_x and B_y are the number of blocks to represent the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. The Algorithm 4 is analogous to the algorithm for merging two sorted arrays or sort-merge join algorithm in relational database systems. In similar way, CVS can compute addition $\boldsymbol{x} + \boldsymbol{y}$, subtraction $\boldsymbol{x} - \boldsymbol{y}$, and division $\boldsymbol{x}/\boldsymbol{y}$ of two compressed vectors in $\mathcal{O}(B_x + B_y)$ time.

5.4 Data Mining Algorithms on CVS

So far, we have described two basic computations on CVS: ℓ_p-norm on a vector and dot product of two vectors. In practice, many complex data mining algorithms can be conducted without decompression using the concept of those computations. In this section, we demonstrate several data mining algorithms on CVS.

5.4.1 k-Nearest Neighbors Algorithm

k-Nearest Neighbors Algorithm (*k-NN*) is an important algorithm used in many applications including document search [71], density estimation [26], and instance-based classifiers. Given a query vector $\boldsymbol{q}$ and a set of vectors $\mathcal{X}$, k-NN algorithm finds k vectors that are closest to query vector $\boldsymbol{q}$ from $\mathcal{X}$.

Since k-NN computes all the distances between the query vector $\boldsymbol{q}$ and the set of vectors $\mathcal{X}$, its runtime is slow on large vector sets. While spatial index structures such as R-Tree and and SR-Tree [60] can reduce the number of distance computations by pruning unpromising vectors, still high-dimensional Euclidean distance computations are required, which can be heavy. CVS can co-exists with these methods and support reducing the computational time and memory usage of the distance computation.

Recall the Euclidean distance of vector $\boldsymbol{x}$ and vector $\boldsymbol{y}$ is defined as

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{\|\boldsymbol{x}\|_2^2 + \|\boldsymbol{y}\|_2^2 - 2\boldsymbol{x}^\top \boldsymbol{y}}. \tag{5.6}$$

Since Equation (5.6) mainly consists of ℓ_2-norm and dot product, CVS efficiently computes the Euclidean distance of $\boldsymbol{x}$ and $\boldsymbol{y}$ without decompression.

5.4.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) algorithm is a mathematical optimization algorithm to find a local minimum of a function. In machine learning, many machine learning models including linear regression, logistic regression, and deep neural networks can be trained by SGD algorithm.

Let us consider the logistic regression with SGD algorithm on CVS. SGD randomly picks a training data $(\boldsymbol{x}_i, y_i)$ and update the regression coefficients $\boldsymbol{w}_{t+1}$ in the following equation

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \gamma \boldsymbol{x}_i \left\{ y_i - \frac{1}{1 + \exp\left(-\boldsymbol{w}_t^\top \boldsymbol{x}_i\right)} \right\}, \tag{5.7}$$

where $\boldsymbol{w}_t$ is the current regression coefficients (prediction model) and γ is the learning rate. Iteratively repeating the update Equation (5.7), SGD converges to the global optima and completes the training of the model. Since Equation (5.7) mainly consists of dot products and additions of vectors, CVS can be naturally applied.

Furthermore, prediction using the trained model $\boldsymbol{w}$ can also be made by CVS efficiently, since the prediction simply consists of a dot product. Concretely, logistic regression predicts the objective value $\hat{y}$ for input feature vector $\boldsymbol{x}$ by

$$p(\hat{y}|\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + \exp\left(-\boldsymbol{w}^\top \boldsymbol{x}\right)}. \tag{5.8}$$

5.4.3 Kernel Method

Kernel methods, such as support vector machine and spectral clustering, are widely used in machine learning tasks because it can capture non-linearity in data well. One downside of kernel method is its high computational cost; Kernel method computes data-to-data similarity $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ through a kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ for all $(i, j) \in [J] \times [J]$, i.e., kernel methods compute the similarities of all vector pairs. This similarity computation requires $\mathcal{O}(D^2 J^2)$ time, which is quite large.

CVS can alleviate the computational cost of data-to-data similarity computation. Consider *radial basis function kernel* (*RBF-Kernel*), which is defined as

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2). \tag{5.9}$$

CVS reduces the computational complexity of Equation (5.9) from $\mathcal{O}(D)$ to $\mathcal{O}(B_i + B_j)$ where B_i and B_j are number of blocks to represent original vectors in RLE, which is often much smaller than D. Thus, the total computational complexity of kernel method is reduced from $\mathcal{O}(D^2 J^2)$ to $\mathcal{O}((B_i + B_j)^2 J^2)$.

5.5 Dimension Reordering

Directly compressing the provided set of vectors does not always yield a good compression result. In this section, we show that CVS can improve the compression rate by reordering the dimensions of vectors without affecting the computational result.

5.5.1 Motivation of Dimension Reordering

Suppose we have a set of 6-dimensional vectors

$$\begin{aligned}
\boldsymbol{x}_1 &= (2, 1, 2, 1, 2, 1), \\
\boldsymbol{x}_2 &= (1, 2, 1, 1, 2, 2), \\
\boldsymbol{x}_3 &= (2, 1, 1, 2, 1, 1).
\end{aligned}$$

By compressing each vector with RLE/Packbits, we obtain the compressed form of the vectors as

$$\begin{aligned}
\mathrm{RLE}(\boldsymbol{x}_1) &= (\langle -6, 2, 1, 2, 1, 2, 1 \rangle), \\
\mathrm{RLE}(\boldsymbol{x}_2) &= (\langle -2, 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 2 \rangle), \\
\mathrm{RLE}(\boldsymbol{x}_3) &= (\langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 1 \rangle).
\end{aligned}$$

However, if we reorder the dimensions by considering a permutation

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 1 & 4 & 6 & 3 & 2 \end{pmatrix}$$

on the dimensions of the vectors, we obtain vectors

$$\begin{aligned} \boldsymbol{x}'_1 &= (1,1,2,2,2,1),\\ \boldsymbol{x}'_2 &= (2,2,2,1,1,1),\\ \boldsymbol{x}'_3 &= (1,1,1,1,2,2), \end{aligned}$$

which yield the better result compared to the original vectors as

$$\begin{aligned} \mathrm{RLE}(\boldsymbol{x}'_1) &= (\langle 2,1\rangle, \langle 3,2\rangle, \langle 1,1\rangle),\\ \mathrm{RLE}(\boldsymbol{x}'_2) &= (\langle 2,3\rangle, \langle 1,3\rangle),\\ \mathrm{RLE}(\boldsymbol{x}'_3) &= (\langle 4,1\rangle, \langle 2,2\rangle). \end{aligned}$$

Many data mining tasks consist of mathematical operations that are not affected by order of vector dimensions. We refer to such a mathematical operation as *dimension-order insensitive*. If data mining tasks on CVS are known to be dimension-order insensitive, CVS reorders the dimensions of vectors to improve the compression rate. All the operations we demonstrated in Section 5.3.2 are dimension-order insensitive.

5.5.2 Problem Definition

We hereafter treat a set of vectors $\mathcal{X}$ to be compressed by CVS as matrix X whose column-vector represents a vector and row-vector represents a dimension. For instance, the vector set in the previous section is treated as

$$X = \begin{bmatrix} \boldsymbol{x}_1^\top & \boldsymbol{x}_2^\top & \boldsymbol{x}_3^\top \end{bmatrix} = \begin{bmatrix} 2 & 1 & 2\\ 1 & 2 & 1\\ 2 & 1 & 1\\ 1 & 1 & 2\\ 2 & 2 & 1\\ 1 & 2 & 1 \end{bmatrix}.$$

Given a set of D-dimensional J vectors as matrix $X \in \mathbb{R}^{D\times J}$, CVS solves the following problem to find the best dimension-order:

Problem 1 (Dimension-reordering)

$$\sigma^* = \underset{\sigma \in \mathcal{P}(D)}{\arg\min} \sum_{j}^{J} |RLE(\sigma(\boldsymbol{x}_j))|$$

where $\mathcal{P}(D)$ is all permutations on D-dimensions, $|\cdot|$ is the number of blocks to represent the vector in the compression, $\boldsymbol{x}_j$ is the j-th column vector of matrix X, and $\sigma(\boldsymbol{x}_j)$ is the vector obtained by reordering the dimensions of $\boldsymbol{x}_j$ using the permutation σ.

Unfortunately, Problem 1 is NP-hard. The problem is equivalent to the *Bitmap-index* reordering problem in column-oriented database systems, which is proved to be an NP-hard problem [68]. Also, the problem is found to be reduced from the Traveling Salesman Problem (TSP) under the Hamming distance, allowing us to investigate the use of heuristics for TSP [68]. In the rest of this section, we introduce two heuristic approaches for dimension reordering: greedy method and Scored-lex-sort method, both of which run in polynomial time.

5.5.3 Greedy Method

First, we introduce a greedy approach to find a good dimension order for Problem 1. The greedy method selects a row (a dimension), and then greedily selects the row that has minimum hamming distance with the previously selected row from the remaining rows as shown in Algorithm 5. In Algorithm 5, $a_{i*} \leftarrow b_{j*}$ denotes replacing the i-th row of the matrix A with the j-th row of the matrix B and $O_{D,J}$ denotes a zero matrix whose size is $D \times J$.

The greedy method runs in polynomial time. The number of row comparison regarding Hamming distance in the greedy method is $(D-1)+(D-2)+...+1 = D(D-1)/2$, and each row comparison needs J element comparison. Thus, the computational complexity of the greedy method is $\mathcal{O}(D^2\,J)$. Although $\mathcal{O}(D^2\,J)$ is polynomial, it is not appropriate for processing vectors with a large number of dimensions, D.

5.5.4 Scored-Lex-Sort Method

Second, we introduce *Scored-lex-sort method*, an efficient approach to find an approximate solution for the Problem 1 based on lexicographical sort. Scored-lex-sort method runs in $\mathcal{O}(J\,D\log D)$, which is better than greedy method's $\mathcal{O}(D^2\,J)$. To introduce scored-lex-sort method, we first describe ordinary lexicographical sort of a matrix.

Algorithm 5: Greedy method

```
   Input: $X \in \mathbb{R}^{D\times J}$: Input matrix.
   Output: Reordered matrix
   Data: $P$: Row pool.
   $\mathbb{X}_{\text{cand}}$: Candidates set.
   $X'$: Reordered matrix.
1  foreach $i \in \{1, ..., D\}$ do
2      $X' \leftarrow O_{D,J}$
       /* Select $i$-th row in matrix $X$ as the beginning row, and add
          other rows to the pool $P$.                                    */
3      $P \leftarrow \{1, ..., D\} \setminus \{i\}$
4      $x'_{1*} \leftarrow x_{i*}$
5      for $k \leftarrow 2$ to $D$ do
           /* Pick up $j$-th row that has minimum hamming distance with the
              beginning row from pool $P$, and add it to reordered matrix
              $X'$.                                                        */
6          $j \leftarrow \arg\min_{j\in P} \text{HammingDistance}(X, k-1, j)$
7          $P \leftarrow P \setminus \{j\}$
8          $x'_{k*} \leftarrow x_{j*}$
9      $\mathbb{X}_{\text{cand}} \leftarrow \mathbb{X}_{\text{cand}} \cup \{X'\}$
10 return $\arg\min_{X'\in\mathbb{X}_{cand}} \mathit{Size}(X')$
```

Lexicographical sort of a matrix reorders rows of a matrix in lexicographical order, where the order of two rows are defined by comparing whose elements from left-to-right. For example, lexicographical sort of the rows of matrix

$$X = \begin{pmatrix} 3 & 4 & 7 & 3 & 0 & 4 \\ 2 & 4 & 2 & 2 & 0 & 0 \\ 3 & 4 & 2 & 5 & 8 & 4 \\ 1 & 3 & 7 & 3 & 8 & 4 \\ 5 & 7 & 7 & 5 & 8 & 4 \end{pmatrix} \tag{5.10}$$

produces a sorted matrix

$$X_{\text{lex}} = \begin{pmatrix} 1 & 3 & 7 & 3 & 8 & 4 \\ 2 & 4 & 2 & 2 & 0 & 0 \\ 3 & 4 & 2 & 5 & 8 & 4 \\ 3 & 4 & 7 & 3 & 0 & 4 \\ 5 & 7 & 7 & 5 & 8 & 4 \end{pmatrix}. \tag{5.11}$$

Since lexicographical sorting of a matrix preferentially sorts left-side columns, it sometimes does not much improve the compression rate of the matrix. For instance, X_{lex}'s compression rate is worse than the original matrix X's one. To alleviate this problem, we introduce *Scored-lex-sort method.* Scored-lex-sort method first computes scores of all columns of the matrix, and then preferentially sorts columns that have high score. Effectiveness of Scored-lex-sort method highly depends on the definition of the score. We use $1/\text{Cardinality}(\mathbf{x})$ as the score of a column $\mathbf{x}$, where the function $\text{Cardinality}(\mathbf{x})$ counts the number of distinct elements in column $\mathbf{x}$. This score definition is based on the insight that a column with low-cardinality contains a lot of same elements, and preferentially sorting such columns improves the compression effect of RLE. For example, matrix X in the previous example is sorted into

$$X_{\text{lex}^*} = \begin{pmatrix} 2 & 4 & 2 & 2 & 0 & 0 \\ 3 & 4 & 2 & 5 & 8 & 4 \\ 3 & 4 & 7 & 3 & 0 & 4 \\ 1 & 3 & 7 & 3 & 8 & 4 \\ 5 & 7 & 7 & 5 & 8 & 4 \end{pmatrix}, \tag{5.12}$$

which has better compression rate than the original matrix X and the matrix X_{lex} that is sorted by the normal lexicographical order.

Algorithm 6 shows the algorithm of row comparison function in Scored-lex-sort method. In the algorithm, x_{ij} indicates the (i, j) element in the matrix X. For the sorting part, we can use arbitrary sorting algorithms such as the merge-sort algorithm to reorder a huge matrix in the external sorting manner.

Scored-lex-sort method runs in polynomial time, and its computational complexity is smaller than the one of the greedy method. As mentioned before, Scored-lex-sort method can use arbitrary general sorting algorithms. General sorting algorithms are known to sort D records in $\mathcal{O}(D \log D)$ [16]. In a comparison operation of two rows,

Algorithm 6: Row-compare function in Scored-lex-sort method

```
    Input: $X \in \mathbb{R}^{D \times J}$: Input matrix.
    $p \in \{ 1, ..., D \}$: Row number of the first input row.
    $q \in \{ 1, ..., D \}$: Row number of the second input row.
    Output: A numerical number indicating whether $p$-th row is bigger/smaller than
            or equal to $q$-th row
    Data: $Q$, Priority queue
1   foreach $k \in \{ 1, ..., J \}$ do
2       Get the score of $k$-th column, and add the column number $k$ to the priority
          queue $Q$ using the score as the priority.
3   while $Q$ is not empty do
4       $k \leftarrow$ Dequeue($Q$)
5       if $x_{pk} > x_{qk}$ then
6           return 1, which indicates $p$-th row > $q$-th row.
7       else if $x_{pk} < x_{qk}$ then
8           return -1, which indicates $p$-th row < $q$-th row.
9   return 0, which indicates $p$-th row = $q$-th row.
```

Scored-lex-sort method needs to compare J elements as described in Algorithm 6. Thus, the computational complexity of Scored-lex-sort method is $\mathcal{O}(J\,D \log D)$, which is sufficiently applicable to large matrices.

5.6 Lossy Compression by Discretization

We have targeted discrete-valued vectors. However, real-world data is not only discrete-valued but also real-valued. Unfortunately, RLE is ineffective for real-valued data, because real-valued data rarely have same consecutive values. To deal with the problem, CVS employ discretization technique to convert real-valued vectors into discrete-valued vectors.

CVS uses a clustering-based discretization approach to improve the compression rate of the vectors as shown in Algorithm 7. In this approach, we first standardize each vector dimensions to have zero mean and unit variance, gather all feature values, and make k clusters from all feature values. Then, values in each cluster are replaced by the cluster centroid. In doing so, we can successfully convert real-valued data into k-discretized data.

One downside of discretization is that it loses the precision of vectors. Through empirical studies, we found that the data mining results, such as predictive performance

Algorithm 7: Vector discretization

```
   Input:  $X \in \mathbb{R}^{D \times J}$: Collection of $D$-dimensional $J$ vectors.
           $K$: Discretization level.
 1 foreach $d \in \{ 1, ..., D \}$ do
 2     foreach $j \in \{ 1, ..., J \}$ do
 3         Standardize $d$-th dimension ($\mu_d$ is the mean and $\sigma_d$ is the standard
            deviation of $d$-th dimension).
 4         $X_{dj} = \frac{X_{dj} - \mu_d}{\sigma_d}$
 5 Groups values into $k$ clusters using $k$-means algorithm, obtaining centroid $c_k$ and
    cluster assignment $z_{dj} \in \{ 1, ..., K \}$ for each value.
 6 $\{ c_k \}_{k=1}^{K}, \{ z_{dj} \}_{d=1,j=1}^{D,J} \leftarrow$ kMeans($\{ X_{dj} \}_{d=1,j=1}^{D,J}, K$).
 7 foreach $d \in \{ 1, ..., D \}$ do
 8     foreach $j \in \{ 1, ..., J \}$ do
 9         Replace the value of $j$-th vector by the centroid of the cluster the vector
            belongs to.
10         $X_{dj} = c_{z_{dj}}$
11 return $X$
```

in k-NN classifiers, are not so affected by the precision of vector values. For example, in binary image classification task, k-NN classifier on CVS with 3-level discretization only degrades the F1-Score 0.005 point while gaining $6\times$ prediction performance improvement (Fig.5.1). We will elaborate on this discussion in the experiments section.

5.7 Experiments

5.7.1 Experimental Setup

System: All of our experiments were run on a machine that has 16GB RAM and dual-core 3.6GHz CPU running Linux 3.8.0. Our proposed system, CVS, is implemented in C++ and compiled by clang++ 3.8.

Datasets: We used two types of vector sets in our experiments: sparse vector sets (bag of words) [7] and dense high-dimensional vector sets [38]. Table 5.2 shows the information of the datasets we used.

Methods: We measured the performance of conventional sparse matrix representation and CVS in several different configurations: **(Sparse)** refers to the conventional sparse matrix representation that only holds non-zero values by $(position, value)$ format,

(RLE) refers to RLE/Packbits on vectors, **(RLE-Sort)** refers to RLE/Packbits on vectors with dimension-reordering, **(RLEn)** refers to RLE/Packbits with n-level data discretization, and **(RLEn-Sort)** refers to RLE/Packbits with n-level data discretization and dimension-reordering. In the reset of this section, we use these notations to explain the configuration on each result.

Table 5.2 Datasets used in experiments.

	Sparse (Bag of Words) [7]			Dense [38]		
	NIPS	KOS	Enron	Madelon	Arcene	Gisette
# of vectors	1,500	3,430	39,861	2,000	900	6,000
# of dimensions	12,419	6,906	28,102	500	10,000	5,000
Density (Non-zero elements ratio)	0.040	0.014	0.003	0.999	0.540	0.129
# of distinct values	120	33	134	660	899	692

5.7.2 Space-Efficiency

Table 5.3 Data compression effect on each dataset by conventional sparse matrix representation (position + value) and our proposed framework CVS. Each number refers to the main/secondary storage usage in megabytes and (%) refers to the compression rate. Compression formats with asterisk (*) refer to lossy compression. While conventional sparse matrix representation successfully compresses the sparse datasets, it fails to compresses the dense datasets. In contrast, our proposed CVS successfully compresses dense and sparse datasets.

	Sparse (Bag of Words)			Dense		
	NIPS	KOS	Enron	Madelon	Arcene	Gisette
Original	71.06	90.36	4273.12	3.81	3.81	114.44
Sparse Matrix	8.54 (12.01%)	4.04 (4.47%)	42.46 (0.99%)	11.44 (300.26%)	6.19 (162.46%)	44.53 (38.91%)
RLE	10.0 (14.07%)	5.07 (5.62%)	53.69 (1.25%)	3.86 (101.40%)	3.84 (100.90%)	49.92 (43.62%)
RLE-Sort	9.12 (12.84%)	4.75 (5.26%)	51.71 (1.21%)	3.88 (101.79%)	3.20 (83.97%)	39.03 (34.10%)
RLE8*	9.94 (13.99%)	5.07 (5.62%)	53.67 (1.25%)	3.55 (93.10%)	2.91 (78.94%)	41.18 (36.31%)
RLE8-Sort*	8.88 (12.50%)	4.76 (5.27%)	51.66 (1.20%)	3.51 (92.13%)	2.65 (71.74%)	38.70 (34.13%)
RLE3*	9.84 (13.85%)	4.52 (5.00%)	53.36 (1.24%)	2.43 (63.73%)	1.53 (51.33%)	40.40 (35.63%)
RLE3-Sort*	8.54 (12.02%)	4.52 (5.00%)	50.34 (1.17%)	2.39 (62.76%)	1.02 (34.16%)	38.79 (34.21%)
RLE2*	9.82 (13.82%)	5.04 (5.58%)	53.32 (1.24%)	2.10 (62.26%)	0.83 (42.19%)	38.13 (33.66%)
RLE2-Sort*	8.41 (11.84%)	4.49 (4.97%)	49.79 (1.16%)	2.05 (60.75%)	0.48 (24.85%)	35.82 (31.63%)

First, we demonstrate the space-efficiency of CVS on real-world vector sets with different configurations including vanilla RLE, dimension-reordering by Scored-lex-sort method, and data discretization. Table 5.3 shows the compression rates of CVS with different configurations, where the compression rate is defined as `compression_rate` =

`compressed_size` / `original_size`. Data sizes shown in Table 5.3 correspond to both the secondary storage usages of the compressed datasets and memory usages of data mining algorithms introduced in Section 5.4 (k-NN classifier, logistic regression with SGD[1] and kernel method) on the datasets.

From Table 5.3, we have the following observations:

- Vanilla RLE is effective for sparse matrices because sparse matrices are mostly filled by zero and RLE effectively represents such zero sequences.
- Discretization is especially effective for dense matrices. Without discretization, RLE on dense matrices can increase the storage usage (e.g., RLE in Madelon).
- Only doing discretization is not enough. Combining discretization and dimension-ordering can drastically improve the compression rate (e.g., RLE8 in Gisette).

5.7.3 Runtime Speedup

Table 5.4 Runtime speedups of k-NN classifiers with Euclidean distance on compressed data. Compression formats with asterisk (*) refer to lossy compression. While conventional sparse matrix represents in sparse data, our proposed method successfully improves the performance in both sparse and dense data. Further, sorting vector dimension and discretization drastically improve the performance.

	Sparse (Bag of Words)			Dense		
	NIPS	KOS	Enron	Madelon	Arcene	Gisette
Sparse	4.68	16.33	132.95	0.18	0.50	1.30
RLE	8.69	23.36	146.97	1.42	1.22	1.97
RLE-Sort	8.99	23.52	202.55	1.31	1.27	3.03
RLE8*	8.67	44.47	169.47	1.33	1.27	3.06
RLE8-Sort*	18.83	46.56	192.66	2.20	1.62	3.15
RLE3*	15.07	45.80	170.65	3.80	2.41	3.38
RLE3-Sort*	20.36	46.71	203.52	3.87	6.13	9.33
RLE2*	18.32	22.76	151.62	4.13	4.93	3.62
RLE2-Sort*	20.82	48.02	218.20	4.26	7.82	8.77

[1]Because SGD is an online algorithm, its memory usage can be reduced to the size of a vector instead of the whole dataset. However, because vectors are randomly scanned, loading whole dataset into the memory is required for efficient computation.

Table 5.5 Runtime speedups of logistic regression with SGD optimization on compressed data.

	Sparse (Bag of Words)			Dense		
	NIPS	KOS	Enron	Madelon	Arcene	Gisette
Sparse	12.73	41.55	174.21	0.65	1.30	4.81
RLE	16.72	46.24	182.26	1.81	2.38	6.61
RLE-Sort	19.08	51.26	199.80	1.85	2.78	7.92
RLE8*	22.65	44.28	165.18	2.64	2.68	7.48
RLE8-Sort*	24.98	50.28	187.48	2.69	3.08	8.19
RLE3*	23.04	45.61	164.27	4.28	4.10	7.86
RLE3-Sort*	26.24	49.57	132.69	4.34	5.36	7.99
RLE2-Sort*	25.51	52.24	189.67	5.19	6.31	9.47

Table 5.6 Runtime speedups of a kernel machine on compressed data.

	Sparse (Bag of Words)			Dense		
	NIPS	KOS	Enron	Madelon	Arcene	Gisette
Sparse	5.15	20.97	182.89	0.16	0.33	1.60
RLE	8.40	24.12	198.21	0.74	0.90	2.51
RLE-Sort	8.94	26.50	205.23	0.75	1.01	3.00
RLE8*	8.55	24.17	193.54	1.02	1.31	2.93
RLE8-Sort*	9.13	26.61	198.31	1.07	1.48	3.23
RLE3*	8.33	22.92	176.66	1.65	2.19	3.15
RLE3-Sort*	11.46	27.30	258.16	1.99	3.31	3.38
RLE2*	8.77	22.95	188.04	1.96	2.71	3.45
RLE2-Sort*	11.70	35.47	306.45	2.81	4.42	3.78

Second, we compare the runtime-speedup of our method CVS and competitive method sparse vector representation on real-world vector sets. In this experiment, we measured the computational time of three classifiers we introduced in Section 5.4: k-NN classifier, logistic regression by SGD optimization, and a nonlinear kernel machine (support vector machine with RBF-kernel).

Setup

For k-NN, we measured the time of the prediction for test vectors, a computationally intensive process that looks through all the training vectors. Test vector is randomly picked up from the vector set, and top-k nearest vectors are selected by its Euclidean distance. We set $k = 5$, repeat the procedure 100 times measuring the elapsed time, and aggregate the results by taking the average. We used efficient Euclidean distance computation for both sparse vector representation and CVS, which run in $\mathcal{O}(B_x + B_y)$ time.

In logistic regression with SGD optimization and kernel method (support vector machine with RBF-kernel), we measured the time to train the model. Since the model vector $\boldsymbol{w}$ trained is rarely sparse in both methods, we used ordinary dense vector representation for $\boldsymbol{w}$, and only used the compression method (sparse vector representation and CVS) for the input vectors $\{ \boldsymbol{x}_j \}_j^J$.

Results

Table 5.4, Table 5.5, and Table 5.6 show the performance comparison of classifiers. From Table 5.4 to Table 5.6, we have the following observations:

- In sparse data, both sparse vector representation and our CVS improve the performance by orders of magnitude.

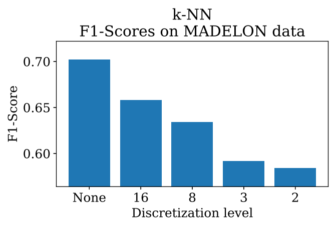
- While sparse vector representation is ineffective for dense data, CVS successfully improved the performance on dense data.

- Dimension reordering gives drastic performance improvements on both sparse and dense vector sets. For example, in Enron, a sparse dataset, the performance improvement increased from 146.97× to 202.55× by dimension reordering in k-NN classifier. In Gisette, a dense dataset, the performance improvement also increased from 1.97× to 3.03×, almost doubled, in k-NN classifier. We can also observe similar results in logistic regression (Table 5.5) and kernel method (Table 5.6).

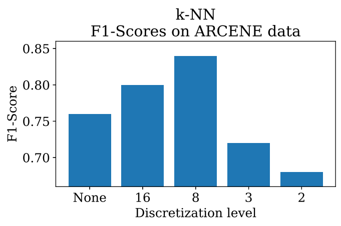
- Data discretization is effective for datasets that have large number of distinct values (Madelon, Arcene, Gisette). For datasets that have relatively smaller number of distinct values (NIPS, KOS, Enron), data discretization does not much improve the performance.

5.7.4 Discretization and Accuracy

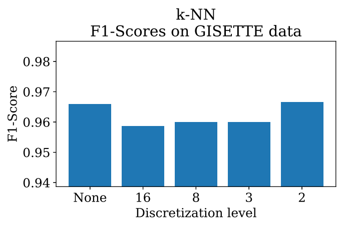
Third, we discuss the effect of lossy-compression (discretization) on accuracy. We measured the predictive performance differences that come from data discretization for three classifiers we introduced in Section 5.4: k-NN classifier, logistic regression



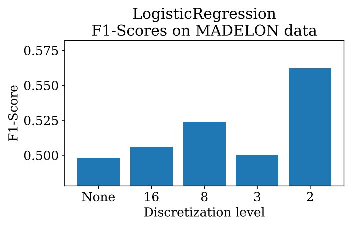
(a) Madelon



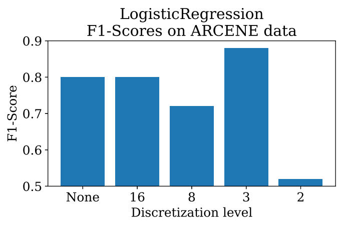
(b) Arcene



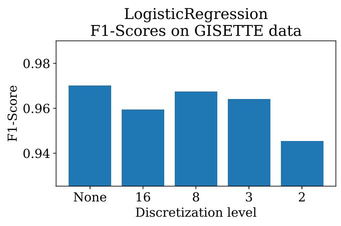
(c) Gisette



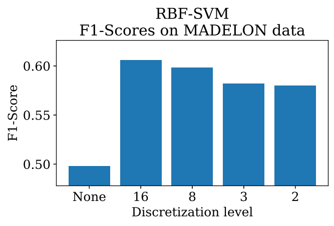
(d) Madelon



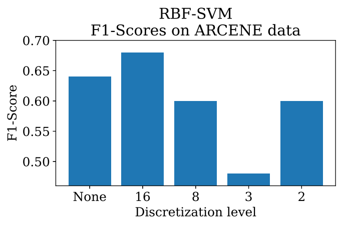
(e) Arcene



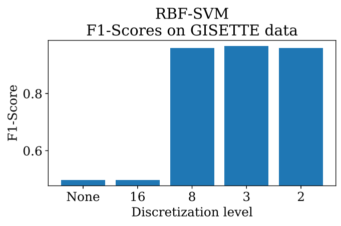
(f) Gisette



(g) Madelon



(h) Arcene



(i) Gisette

Fig. 5.1 F1-scores (harmonic mean of precision and recall) of each classifier (k-NN classifier, logistic regression, and RBF-SVM) on each dataset with discretization. X-axis refers to the discretization level (# of distinct values) and Y-axis refers to the F1-score. We can observe that the discretization does not much degrade the predictive performance and sometimes does improve F1-scores.

by SGD optimization, and a nonlinear kernel machine (support vector machine with RBF-kernel). We used Algorithm 7 for data discretization and varied discretization level $K \in \{2, 3, 8, 16\}$. For the distance measure in k-means algorithm, we used Euclidean distance.

Fig.5.1 shows the F1-score (harmonic mean of precision and recall) of each classifier on each dataset. X-axis refers to the discretization level (# of distinct values) and Y-axis refers to the F1-score. From Fig.5.1 (that shows accuracies) and Table 5.4 (that shows speedups), we observe that the discretization does not degrade the accuracy of classifiers much, while it drastically improves the compression rate and computational performance. Furthermore, discretization sometimes can improve the predictive performance classifiers.

One may think this phenomena strange but data discretization is a well adopted feature-engineering technique to improve the predictive performance by reducing the effect of observation noise and outliers, which is referred to as *data binning* in machine learning area.

5.8 Related Work

CVS relates to vector and matrix computation frameworks that utilize data sparsity. For example, Eigen [36], a vector computation library, can represent sparse matrices and sparse vectors by concise data structures. In contrast to the sparse matrix representation, CVS targets to not only the sparse vector sets but also the dense vector sets.

Run-length encoding is widely used in data-intensive systems. For example, MADlib [42], a data analytics system built on top of a relational database system, uses run-length encoding to handle sparse data efficiently. Also, some columnar database systems employ run-length encoding its column data [1, 69]. However, to the best of our knowledge, none of these works addressed the performance improvement of run-length encoding on data mining and machine learning tasks and the effect of dimension-reordering and data discretization.

In machine learning research area, compressing the specific machine learning models has been actively studied recently. **(Binary features)** Tabei *et al.* studied partial least squares regression (PLS) on compressed data encoded by grammar-based codes. To assist accessing the elements in the compressed matrix, they proposed a tree-based special data structure. Their approach targets to use binary-features (so-called fingerprints data), which differs from our approach that targets to arbitrary real values. **(Relational data)** Rendle proposed a method to accelerate machine-learning algorithms by utilizing block structures in a matrix [84]. In his work, input matrices are assumed to have special block structures that come from denormalization of relational tables, whereas CVS does not impose any assumptions on the input data. **(Deep neural networks)** In deep neural network research community, to reduce the size of huge deep learning models, DNN model compression has been recently actively studied. Approaches include low-rank approximation of parameter tensors [19, 52], binary representation [83] or ternary representation [106] of parameters, and pruning unimportant nodes from parameters [66].

Brodie *et al.* tackled the row-reordering problem we have defined in Section 5.5.2, and proposed a method that is similar to our greedy method, whose computational complexity is $\mathcal{O}(D^2 J)$ [10]. In their situation, the greedy method was enough, because they aimed to compress the state-transition tables of a regular expression, and commonly such tables are not large. In database systems area, the problem of reordering bitmap indices has been studied to get better compression result, which is equivalent to our dimension-reordering problem with binary values [68, 81].

5.9 Summary

In this chapter, we proposed CVS (Compressed Vector Set), a general framework for concisely storing vector sets and conducting mathematical operations on the vector sets efficiently. CVS holds a set of vectors in a compressed format and conducts mathematical operations, such as ℓ_p-norm and dot product, without decompression. We demonstrate that CVS accelerates several data mining algorithms including k-nearest neighbor algorithm, stochastic gradient descent algorithm on logistic regression, and kernel methods. Our experimental results demonstrated that CVS can process both dense datasets and sparse datasets faster than conventional sparse vector representation with smaller memory usage.

Chapter 6

Conclusions

6.1 Summary

This dissertation explored methods to reduce the time of processes inside the trial-and-errors in the machine learning on relational data.

Fast Feature Engineering by Adaptive Partial Aggregation First, we proposed *Adaptive Partial Aggregation Tree (APA-Tree)*, a query processing technology for accelerating repetitive data aggregation operations (Chapter 3). APA-Tree is based on partial aggregation method [73, 34] that pre-computes and cache aggregation values on a subset of the data and reuse pre-computed aggregation values in further aggregation queries. While conventional partial aggregation method computes aggregation values on pre-determined sized subsets of data (typically a page or a block in storage), APA-Tree computes aggregation values on dynamically-decided sized subsets. Since aggregation operations in feature engineering tasks have locality of reference, APA-Tree finely computes aggregation values on frequently accessed data and coarsely on rarely accessed data. As a result, APA-Tree accelerates repetitive aggregation queries with a small number of pre-computed aggregation values. Experimental results on synthetic workloads confirmed APA-Tree's efficiently compared to the previous partial aggregation methods [73, 34].

Machine Learning without Feature Construction Second, we proposed a machine learning model for predicting entity attributes in relational database systems, which does not require conventional time-consuming feature-engineering process (Chapter 4). As a motivating application of entity attribute prediction in relational data, we tackled the problem of customers' demographics prediction based on their behavioral data. This demographics prediction problem is modeled as a classification task in which a customer's sensitive demographic y is predicted from his feature vector $\boldsymbol{x}$. So far, two lines of work have tried to produce a "good" feature vector $\boldsymbol{x}$ from the customer's behavioral data: (1) application-specific feature engineering using behavioral data and (2) representation learning (such as singular value decomposition or neural embedding) on behavioral data. Although these approaches successfully improve the predictive performance, (1) designing a good feature requires domain experts to make a great effort and (2) features obtained from representation learning are hard to interpret. To overcome these problems, we presented a *Relational Infinite Support Vector Machine (R-iSVM)*, a mixture-of-experts model that can leverage behavioral data. Instead of augmenting the feature vectors of customers, R-iSVM uses behavioral data to find out behaviorally similar customer clusters and constructs a local prediction model at each customer cluster. In doing so, R-iSVM successfully improves the predictive performance without requiring application-specific feature designing and hard-to-interpret representations. Experimental results on three real-world datasets demonstrated the predictive performance and interpretability of R-iSVM. Furthermore, R-iSVM can co-exist with previous demographics prediction methods to further improve their predictive performance.

Fast and Space-Efficient Machine Learning with Data Compression Third, we proposed a method to make training and evaluation of various machine learning models fast and space-efficient, which is based on data compression technique (Chapter 5). Our proposed *compressed vector set (CVS)* runs machine learning algorithms in a fast and space-efficient manner on both sparse and dense datasets. CVS holds a set of vectors in a compressed format and conducts primitive vector operations, such as ℓ_p-norm and dot product, without decompression. By combining these primitive operations, CVS accelerates prominent data mining or machine learning algorithms including k-nearest neighbor algorithm, stochastic gradient descent algorithm on logistic regression, and kernel methods. In contrast to the commonly used sparse matrix/vector representation, which is not effective for dense datasets, CVS efficiently handles sparse datasets and dense datasets in a unified manner. Our experimental results demonstrated that CVS could process both dense datasets and sparse datasets faster than conventional sparse vector representation with smaller memory usage.

6.2 Future Direction

There are a number of promising future research directions.

Supporting Wide Variety of Queries in Aggregation Reusing Methods Aggregation reusing methods such as partial aggregation methods and our proposed APA-Tree accelerate simple aggregation queries with selection operation such as range filter. While such aggregation queries are frequently used in feature engineering, other types of queries such as group-by aggregation queries are also often used. Thus, developing a way to reuse aggregation values among such wide variety of queries is a promising direction. Resulting technology may embody database systems techniques such as *materialized view maintenance* [37] for reusability analysis on complicated queries and machine learning techniques such as *independence criterion* [35] for determining important features to be cached.

Assisting Data Analysts' Understanding of Trained Models One reason of trial-and-error process in machine learning being laborious is difficulty in understanding the trained models. Data analysts have to understand the behavior of models for debugging the models or explaining the results to other people. While we addressed the issue by proposing R-iSVM, an interpretable machine learning model for customer demographics prediction, there remains a laborious task: data analysts have to understand and name customer clusters by investigating demographics distributions of a cluster and relationship to item clusters. Thus, assisting such task through technology is necessary. Using techniques for understanding and naming latent factors (topics) using textual features [12, 72] is a promising direction.

Heterogeneous Data Representation and Planner for Machine Learning Our proposed CVS, which encodes a set of vectors with run-length encoding and apply some optimization techniques such as dimension reordering and data discretization, can be considered as a data representation format. Besides CVS, there exists a tremendous amount of data representation formats. Handling these heterogeneous data representations in a unified framework is a promising direction. Since different data representation yields different performance (computational time and memory usage) for a data and algorithms to be executed, there exists a chance for the planning that chooses the best data representation and operation execution orders in the algorithm. While the problem partially shares the concept of classical query optimization technique in database systems, there exists a new issues such as lack of operation rewriting rules in machine learning algorithms, which are available in rigorous relational algebra. Several studies share the similar motivation and are trying to develop planner for linear algebra and machine learning algorithms [23, 64, 89].

Bibliography

[1] Abadi, D., Madden, S., and Hachem, N. (2008). Column-Stores vs. Row-Stores: How different are they really? In *SIGMOD*.

[2] Abouzied, A., Abadi, D. J., and Silberschatz, A. (2013). Invisible loading. In *EDBT*, pages 1–10, New York, New York, USA. ACM Press.

[3] Acharya, S., Gibbons, P. B., and Poosala, V. (2000). Congressional Samples for Approximate Answering of Group-By Queries. In *SIGMOD*, pages 487–498.

[4] Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., and Stoica, I. (2013). BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*, pages 29–42, New York, NY, USA. ACM.

[5] Anderson, M. R. and Cafarella, M. (2016). Input Selection for Fast Feature Engineering. In *ICDE*, pages 577–588.

[6] Apple Inc. (1996). Apple Technical Note TN1023.

[7] Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository.

[8] Blalock, D. W. and Guttag, J. V. (2017). Bolt: Accelerated Data Mining with Fast Vector Compression. In *KDD*.

[9] Blei, D. M., Kucukelbir, A., and Mcauliffe, J. D. (2017). Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

[10] Brodie, B. C., Taylor, D. E., and Cytron, R. K. (2006). A Scalable Architecture for High-Throughput Regular-Expression Pattern Matching. In *ISCA*, pages 191–202. IEEE Computer Society.

[11] Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. *Systems Research*, Research R(124):24.

[12] Carmel, D., Roitman, H., and Zwerdling, N. (2009). Enhancing cluster labeling using wikipedia. In *SIGIR*, number January 2016, pages 139–146.

[13] Chaudhuri, S., Das, G., and Narasayya, V. (2007). Optimized Stratified Sampling for Approximate Query Processing. *TODS*.

[14] Chaudhuri, S., Ding, B., and Kandula, S. (2017). Approximate Query Processing: No Silver Bullet. In *SIGMOD*, pages 511–519.

[15] Cook, R. D. (1977). Detection of Influential Observation in Linear Regression. *Technometrics*, 19(1):15–18.

[16] Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

[17] Crammer, K. and Singer, Y. (2001). On The Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *JMLR*, 2:265–292.

[18] Culotta, A., Ravi, N. K., and Cutler, J. (2015). Predicting the Demographics of Twitter Users from Website Traffic Data. In *AAAI*, pages 72–78.

[19] Denton, E., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *NIPS*.

[20] Deodhar, M. and Ghosh, J. (2010). SCOAL: A Framework for Simultaneous Co-Clustering and Learning from Complex Data. *TKDD*, 4(3):1–31.

[21] Dong, Y., Yang, Y., Tang, J., Yang, Y., and Chawla, N. V. (2014). Inferring User Demographics and Social Strategies in Mobile Social Networks. In *KDD*, pages 15–24.

[22] Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features. In Prieditis, A. and Russell, S. J., editors, *ICML*, pages 194–202. Morgan Kaufmann.

[23] Elgohary, A., Boehm, M., Haas, P. J., Reiss, F. R., and Reinwald, B. (2016). Compressed Linear Algebra for Large-Scale Machine Learning. *VLDB*, 9(12):960–971.

[24] Francisco, P. and Others (2011). The Netezza data appliance architecture: A platform for high performance data warehousing and analytics.

[25] Fu, W. J. (1998). Penalized Regressions: The Bridge versus the Lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416.

[26] Fukunaga, K. and Hostetler, L. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40.

[27] Garofalakis, M. and Gibbons, P. B. (2001). Approximate Query Processing: Taming the TeraBytes! (Tutorial). In *VLDB*.

[28] Ge, T., He, K., Ke, Q., and Sun, J. (2014a). Optimized Product Quantization. *TPAMI*, 36(4):744–755.

[29] Ge, T., He, K., Ke, Q., and Sun, J. (2014b). Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755.

[30] Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

[31] Golub, G. H. and Van Loan, C. F. (2013). *Matrix Computations (4th Ed.)*. Johns Hopkins University Press.

[32] Gong, Y., Lazebnik, S., Gordo, A., and Perronnin, F. (2013). Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *TPAMI*, 35(12):2916–2929.

[33] Graefe, G. and Kuno, H. (2010a). Adaptive indexing for relational keys. In *ICDE*, pages 69–74.

[34] Graefe, G. and Kuno, H. A. (2010b). Fast loads and queries. In *TLDKS*, volume 6380 LNCS, pages 31–72.

[35] Gretton, A., Bousquet, O., Smola, A., and Schlkopf, B. (2005). Measuring Statistical Dependence with Hilbert-Schmidt Norms. In *ALT*, volume 3734 LNAI, pages 63–77.

[36] Guennebaud, G., Jacob, B., and Others (2010). Eigen v3. http://eigen.tuxfamily.org.

[37] Gupta, A. and Mumick, I. S. (1995). Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin*, 18:3–18.

[38] Guyon, I., Gunn, S. R., Ben-Hur, A., and Dror, G. (2004). Result Analysis of the NIPS 2003 Feature Selection Challenge. In *NIPS*.

[39] Harper, F. M. and Konstan, J. A. (2015). The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1—-19:19.

[40] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural Collaborative Filtering. In *WWW*.

[41] Hellerstein, J. M., Haas, P. J., and Wang, H. J. (1997). Online Aggregation. *SIGMOD Record*, 26(2):171–182.

[42] Hellerstein, J. M., Ré, C., Schoppmann, F., Wang, D. Z., Fratkin, E., Gorajek, A., Ng, K. S., Welton, C., Feng, X., Li, K., and Kumar, A. (2012). The MADlib analytics library: or MAD skills, the SQL. *VLDB*, 5(12):1700–1711.

[43] Ho, C.-T., Agrawal, R., Megiddo, N., and Srikant, R. (1997). Range Queries in OLAP Data Cubes. In *SIGMOD*, pages 73–88.

[44] Hu, J., Zeng, H.-J., Li, H., Niu, C., and Chen, Z. (2007). Demographic prediction based on user's browsing behavior. In *WWW*, pages 151–160.

[45] Idreos, S., Alagiannis, I., Johnson, R., and Ailamaki, A. (2011). Here are my Data Files . Here are my Queries . Where are my Results ? In *CIDR*.

[46] Idreos, S., Kersten, M., and Manegold, S. (2007a). Database Cracking. In *CIDR*.

[47] Idreos, S., Kersten, M. L., and Manegold, S. (2007b). Updating a cracked database. In *SIGMOD*, page 413.

[48] Idreos, S., Kersten, M. L., and Manegold, S. (2009). Self-organizing Tuple Reconstruction in Column-stores. In *SIGMOD*.

[49] Ishiguro, K., Sato, I., and Ueda, N. (2014). Collapsed Variational Bayes Inference of Infinite Relational Model. *arXiv*, 1:arXiv:1409.4757 [cs.LG].

[50] Jaakkola, T., Meila, M., and Jebara, T. (1999). Maximum Entropy Discrimination. In *NIPS*, volume AITR-1668, pages 470–476.

[51] Jacobs, R. A. and Jordan, M. I. (1991). Adaptive Mixtures of Local Experts. *Neural Computation*, 3:79–87.

[52] Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up Convolutional Neural Networks with Low Rank Expansions. *CoRR*, abs/1405.3.

[53] Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128.

[54] Joachims, T., Finley, T., and Yu, C. N. J. (2009). Cutting-Plane Training of Structural SVMs. *Machine Learning*, 77(1):27–59.

[55] Jones, E., Oliphant, T., Peterson, P., and Others (2001). SciPy.org.

[56] Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214.

[57] Kargin, Y., Ivanova, M., and Zhang, Y. (2013). Lazy ETL in action: ETL technology dates scientific data. *PVLDB*, 6(12):1286–1289.

[58] Kargin, Y., Kersten, M., Manegold, S., and Pirk, H. (2015). The DBMS-your big data sommelier. In *ICDE*, pages 1119–1130. IEEE.

[59] Karpathiotakis, M., Branco, M., Alagiannis, I., and Ailamaki, A. (2014). Adaptive query processing on RAW data. *PVLDB*, 7(12):1119–1130.

[60] Katayama, N. and Satoh, S. (1997). The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, pages 369–380. ACM.

[61] Kemp, C., Tenenbaum, J. J. B., Griffiths, T. L. T., Yamada, T., and Ueda, N. (2006). Learning Systems of Concepts with an Infinite Relational Model. In *AAAI*, volume 21, pages 381–388.

[62] Kong, W. and Li, W.-j. (2012). Isotropic hashing. In *NIPS*, pages 1–9.

[63] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *NIPS*, pages 1097–1105. Curran Associates, Inc.

[64] Kumar, A., Boehm, M., and Yang, J. (2017). Data Management in Machine Learning : Challenges , Techniques , and Systems (Tutorial). In *SIGMOD*, number Section 4, pages 1717–1722.

[65] Kurihara, K., Welling, M., and Teh, Y. W. (2007). Collapsed Variational Dirichlet Process Mixture Models. In *IJCAI*.

[66] LeCun, Y., Denker, J. S., and Solla, S. A. (1989). Optimal Brain Damage. In Touretzky, D. S., editor, *NIPS*, pages 598–605. Morgan Kaufmann.

[67] Leis, V., Kemper, A., and Neumann, T. (2013). The adaptive radix tree: ARTful indexing for main-memory databases. In *ICDE*, pages 38–49.

[68] Lemire, D., Kaser, O., and Gutarra, E. (2012). Reordering Rows for Better Compression: Beyond the Lexicographic Order. *TODS*, 37(3):20:1—-20:29.

[69] Li, Y. (2013). BitWeaving : Fast Scans for Main Memory Data Processing. In *SIGMOD*.

[70] Lipton, Z. C. (2016). The Mythos of Model Interpretability. In *ICML Workshop on Human Interpretability in Machine Learning*.

[71] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[72] Mcauley, J. and Leskovec, J. (2013). Hidden Factors and Hidden Topics : Understanding Rating Dimensions with Review Text. In *RecSys*, pages 165–172.

[73] Moerkotte, G. (1998). Small Materialized Aggregates: A Lightweight Index Structure for Data Warehousing. In *VLDB*.

[74] Müller, A. and Behnke, S. (2013). PyStruct - Learning Structured Prediction in Python. *JMLR*, 15:2055–2060.

[75] Murray, D. and Durrell, K. (2000). Inferring demographic attributes of anonymous internet users. *Web Usage Analysis and User Profiling*, 1836:7–20.

[76] Norouzi, M. and Fleet, D. J. (2013). Cartesian k-means. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3017–3024.

[77] Oliphant, T. E. (2007). SciPy: Open source scientific tools for Python. *Computing in Science and Engineering*, 9:10–20.

[78] Oyamada, M., Liu, J., Narita, K., and Araki, T. (2014). MOARLE: Matrix Operation Accelerator Based on Run-Length Encoding. In Chen, L., Jia, Y., Sellis, T., and Liu, G., editors, *APWeb*, pages 425–436. Springer International Publishing.

[79] Pinar, A. and Heath, M. T. (1999). Improving Performance of Sparse Matrix-vector Multiplication. In *SC*, New York, NY, USA. ACM.

[80] Pountain, D. (1987). Run-length encoding. *Byte*, 12 no. 6:317–319.

[81] Pourabbas, E., Shoshani, A., and Wu, K. (2012). Minimizing Index Size by Reordering Rows and Columns. In *SSDBM*, SSDBM'12, pages 467–484, Berlin, Heidelberg. Springer-Verlag.

[82] Raman, V., Attaluri, G. K., Barber, R., Chainani, N., Kalmuk, D., KulandaiSamy, V., Leenstra, J., Lightstone, S., Liu, S., Lohman, G. M., Malkemus, T., Müller, R., Pandis, I., Schiefer, B., Sharpe, D., Sidle, R., Storm, A. J., and Zhang, L. (2013). DB2 with BLU Acceleration: So Much More than Just a Column Store. *PVLDB*, 6(11):1080–1091.

[83] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *ECCV*, pages 525–542, Cham. Springer International Publishing.

[84] Rendle, S. (2013). Scaling Factorization Machines to Relational Data. *PVLDB*, 6(5):337–348.

[85] Sapia, C. (2000). PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems. In *DaWaK*. Springer LNCS.

[86] Sethuraman, J. (1994). A Constructive Deefinition of Dirichlet Priors. *Statistica Sinica*, 4:639–650.

[87] Sidirourgos, L., Kersten, M., and Boncz, P. (2011). SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *CIDR*.

[88] Slezak, D., Wroblewski, J., Eastwood, V., and Synak, P. (2008). Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries. *VLDB*, 1(2):1337–1345.

[89] Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J., Franklin, M. J., Jordan, M. I., and Kraska, T. (2013). MLI: An API for distributed machine learning. In *ICDM*, pages 1187–1192.

[90] Stoer, J. and Bulirsch, R. (1982). Introduction to Numerical Analysis.

[91] Tabei, Y., Saigo, H., Yamanishi, Y., and Puglisi, S. J. (2016). Scalable Partial Least Squares Regression on Grammar-Compressed Data Matrices. In *KDD*.

[92] Tanenbaum, A. S. (2007). *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[93] Tobias, M., Wolf, R., Seilbeck, R., Reiser, A., Kemper, A., and Neumann, T. (2014). Instant Loading for Main Memory Databases. *PVLDB*, 6(14):1702–1713.

[94] Vartak, M., Ortiz, P., Siegel, K., Subramanyam, H., Madden, S., and Zaharia, M. (2016). Supporting Fast Iteration in Model Building. In *NIPS ML Systems Workshop*, pages 1–6.

[95] Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for Similarity Search: A Survey. *arXiv preprint arXiv:1408.2927*.

[96] Wang, J., Wang, J., Song, J., Xu, X. S., Shen, H. T., and Li, S. (2015). Optimized Cartesian K-means. *IEEE Transactions on Knowledge and Data Engineering*, 27(1):180–192.

[97] Wang, P., Guo, J., Lan, Y., Xu, J., and Cheng, X. (2016). Your Cart tells You: Inferring Demographic Attributes from Purchase Data. In *WSDM*.

[98] Wang, Y. J. and Wong, G. Y. (1987). Stochastic Blockmodels for Directed Graphs. *Journal of the American Statistical Association*, 82(397):8–19.

[99] Xu, Z. (2007). *Statistical Relational Learning with Nonparametric Bayesian Models*. PhD thesis, Ludwig-Maximilians-University of Munich.

[100] Yan, W. P. and Larson, P.-b. (1995). Eager Aggregation and Lazy Aggregation. In *VLDB*.

[101] Zellner, A. (1988). Optimal Information Processing and Bayes's Theorem. *The American Statistician*, 42(4):278–280.

[102] Zhang, C., Kumar, A., and Ré, C. (2014). Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*.

[103] Zhang, J., Du, K., Cheng, R., Wei, Z., Qin, C., You, H., and Hu, S. (2016). Reliable Gender Prediction Based on Users' Video Viewing Behavior. In *ICDM*, pages 649–658.

[104] Zhao, X. W., Guo, Y., He, Y., Jiang, H., Wu, Y., and Li, X. (2014). We Know What You Want to Buy: A Demographic-based System for Product Recommendation on Microblogs. In *KDD*, pages 1935–1944.

[105] Zhong, Y., Yuan, N. J., Zhong, W., Zhang, F., and Xie, X. (2015). You Are Where You Go: Inferring Demographic Attributes from Location Check-ins. In *WSDM*, pages 295–304.

[106] Zhu, C., Han, S., Mao, H., and Dally, W. J. (2016). Trained Ternary Quantization. In *ICLR*, pages 1–10.

[107] Zhu, J., Chen, N., and Xing, E. P. (2011). Infinite SVM: a Dirichlet Process Mixture of Large-margin Kernel Machines. In *ICML*, pages 617–624.

[108] Zhu, J., Chen, N., and Xing, E. P. (2014). Bayesian Inference with Posterior Regularization and Applications to Infinite Latent SVMs. *JMLR*, 15:1799–1847.

Publications

Publications related to this dissertation

Journal papers

- Masafumi Oyamada, Jianquan Liu, Shinji Ito, Kazuyo Narita, Takuya Araki, and Hiroyuki Kitagawa, Compressed Vector Set: A Fast and Space-Efficient Data Mining Framework, IPSJ Transaction on Databases **(IPSJ-TOD)**, vol.77 (to appear), 2017.

Conference papers

- Masafumi Oyamada and Shinji Nakadai, Relational Mixture of Experts: Explainable Demographics Prediction with Behavioral Data, IEEE International Conference on Data Mining **(ICDM 2017)**, pp. 357-366, 2017 **(Regular paper)**.
- Masafumi Oyamada, Jianquan Liu, Kazuyo Narita, and Takuya Araki. MOARLE: Matrix Operation Accelerator based on Run-Length Encoding, 16th Asia-Pacific Web Conference **(APWeb 2014)**, pp. 425-436, 2014 **(Best paper runner up)**.

Other publications

Journal papers

- Masafumi Oyamada, Hideyuki Kawashima, and Hiroyuki Kitagawa, Data Stream Processing with Concurrency Control, ACM SIGAPP Applied Computing Review **(ACM SIGAPP ACR)**, Vol. 13, No. 2, pp. 54-65, June 2013.

Conference papers

- Katsufumi Tomobe, Masafumi Oyamada, and Shinji Nakadai, Link Prediction for Isolated Nodes in Heterogeneous Network by Topic-based Co-Clustering, 21st Pacific Asia Conference on Knowledge Discovery and Data Mining **(PAKDD 2017)**, pp. 147-159, 2017.
- Masafumi Oyamada, Hideyuki Kawashima, and Hiroyuki Kitagawa, Continuous Query Processing with Concurrency Control: Reading Updatable Resources Consistently, Proc. 28th ACM Symposium on Applied Computing **(SAC 2013)**, Coimbra, Portugal, pp. 788-794, March 18-22, 2013.

- Masafumi Oyamada, Hideyuki Kawashima, and Hiroyuki Kitagawa, Efficient Invocation of Transaction Sequences Triggered by Data Streams The 2nd International Workshop on Streaming Media Delivery and Management Systems **(SMDMS 2011)**, Proceedings of 3PGCIC, Barcelona, Spain, pp. 332-337, October 26-28, 2011.

- Masafumi Oyamada, Hideyuki Kawashima, and Hiroyuki Kitagawa, Integration of Data Streams and Relations with Main Memory Database, Proc. 8th International Conference on Networked Sensing Systems **(INSS 2011)**, Penghu, Taiwan, June 13-15, 2011.