A NOVEL MULTIVARIATE DISCRETIZATION ALGORITHM USING DYNAMIC PROGRAMMING

DİNAMİK PROGRAMLAMA KULLANAN ÖZGÜN BİR ÇOK DEĞİŞKENLİ AYRIKLAŞTIRMA ALGORİTMASI

ALİ BURAK ERDOĞAN

ASSOC. PROF. DR. BURKAY GENÇ

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

January 2023

ABSTRACT

A NOVEL MULTIVARIATE DISCRETIZATION ALGORITHM USING DYNAMIC PROGRAMMING

Ali Burak Erdoğan

Master of Science , Computer Engineering

Supervisor: Assoc. Prof. Dr. Burkay GENÇ

January 2023, 81 pages

Discretization is the task of converting quantitative (continuous) numerical data into qualitative (categorical) by assigning them into non-overlapping intervals. It is an important step in reducing the complexity of data in data mining and exploratory data analysis studies. There are many methods that provide discretization schemes on continuous attributes, such as equal-width, equal-frequency, and minimum description length principle (MDLP). On the other hand, these methods ignore the multivariate nature of the dataset and focus on a single feature space for discretization. This causes a loss of information with respect to the correlations between attributes. Moreover, unlabeled data cannot be discretized with supervised methods (e.g. MDLP) that use class labels. We propose a new technique for unsupervised, multivariate, global, and static discretization; a discretizer based on information entropy which employs a constrained shortest-path algorithm. We test our technique on manually crafted randomized synthetic datasets as well as well-known real datasets. We show that our approach provides a more meaningful discretization in test cases. This may allow the retrieval of meaningful intervals, which are hidden, for data exploratory tasks. Also, classification accuracy on real datasets generally improves with our method unlike other univariate benchmark methods. Hence, our method may serve to achieve better accuracy on classification tasks.

Keywords: multivariate discretization, discretizer, data mining, exploratory data analysis

ÖZET
DİNAMİK PROGRAMLAMA KULLANAN ÖZGÜN BİR ÇOK DEĞİŞKENLİ AYRIKLAŞTIRMA ALGORİTMASI

Ali Burak Erdoğan

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Doç. Dr. Burkay GENÇ

Ocak 2023, 81 sayfa

Ayrıklaştırma, nicel ve sürekli sayısal verileri, kesişmeyen aralıklara atayarak, nitel ve sınıflandırılabilir bir veriye dönüştürme işlemine verilen isimdir. Ayrıklaştırma, veri madenciliği ve keşifsel veri analizi çalışmalarında verinin karmaşıklığını azaltmak için uygulanan önemli bir adımdır. Eşit-genişlik, eşit-sıklık ve MDLP (minimum tanım uzunluğu prensibi) gibi sürekli sayısal verileri ayrıklaştırmak için kullanılan birçok yöntem mevcuttur. Bununla beraber, saydığımız yöntemler verinin çok değişkenli doğasını göz önüne almayıp, sadece bir değişkene odaklanmaktadır. Bu da verinin öz nitelikleri arasındaki mevcut korelasyon bilgisinin kaybolmasına sebep olmaktadır. Ayrıca, sınıflandırılmamış veriler, MDLP gibi sınıf bilgisine dayalı denetimli yöntemler ile ayrıklaştırılamamaktadır. Bu çalışmada, kısıtlanmış en kısa yol algoritması kullanan ve bilgi entropisine dayanan; denetimsiz, çok değişkenli, evrensel ve statik bir ayrıklaştırıcı öneriyoruz. Bu ayrıklaştırıcı tekniğimizi manuel olarak hazırlanmış rastgele sentetik veri kümeleri üzerinde test ederek, yaklaşımımızın ilişkili öznitelikler üzerinden hesaplanan entropiye göre çoğu test durumunda daha başarılı bir ayrıklaştırma sağladığını gösteriyoruz. Bu yöntem, keşifsel veri analizi gibi görevler için veri içerisinde gizli olan anlamlı aralıkların keşfedilmesinde yardımcı bir rol üstlenebilir. Buna ek olarak, yöntemimizi gerçek veri kümeleri üzerinde test ettiğimizde sınıflandırma doğruluğunun genel olarak –tek değişkenli yöntemlerin aksine – iyileştiğini gözlemledik. Dolayısıyla, ayrıklaştırma yöntemimiz sınıflandırma görevlerinde daha yüksek bir doğruluk elde edilmesine yardım edebilir.

Keywords: çok değişkenli ayrıklaştırma, ayrıklaştırıcı, veri madenciliği, keşifsel veri analizi

ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious, the Most Merciful.

First, I would like to express my deep gratitude to my supervisor, **Assoc. Prof. Dr. Burkay Genç**, who has tirelessly and patiently assisted and guided me throughout this study, both mentally and with his knowledge.

Besides, I would like to thank to **Assoc. Prof. Dr. Hüseyin Tunç** for his valuable contributions and continuous feedback and to **Prof. Dr. İlyas Çiçekli** for his important comments on this thesis.

Finally, I especially feel indebted to **my dear mother Bilge, father Davut, and my wife** for their endless support and encouragement. They have always kept me motivated during the trying times of this journey and never let me give up.

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

PCA	:	Principal component analysis
CPD	:	Correlation-preserving discretization
IPD	:	Interaction-preserving discretization
MVD	:	Multi-variate discretization
ME-MDLP	:	Minimum Entropy - Minimum Description Length Principle

1. INTRODUCTION

Discretization is one of the most essential data preprocessing techniques in the data mining field. It is a data reduction mechanism which ensures a large variety of numeric values are mapped to a much smaller number of fixed intervals. Thus, quantitative data is transformed into qualitative data (i.e. categorical values) by mapping each value in a continuous attribute to a corresponding interval and transforming all continuous values into a discrete number of values. Three of the top ten data mining algorithms (C4.5 [2], Apriori, Naive Bayes) require data discretization [3], and some other data mining algorithms work better in terms of accuracy and efficiency with the use of discretization [4]. For example, Dougherty et al. [5] showed in their experiments that the classification accuracy of Naive-Bayes algorithm was significantly improved when attributes are discretized with entropy-based discretizers, as well as C4.5 algorithm performed better when the continuous features was discretized in pre-processing step. It is known that decision trees become more concise and provide more accurate results with the use of discretization [4]. Furthermore, discretization is not only used for improving performance of data mining algorithms; but also for gaining insights about data such as uncovering hidden patterns and understanding, using, and explaining data.

1.1. Contributions

It is well-known that discretization causes information loss on the dataset as a numerical attribute might include significant information to describe an observation's class. Here, we develop a dynamic programming based algorithm which discretizes a single attribute while taking other attributes (multi-variate information) into account so as to reduce information loss. The algorithm aims to be sufficiently efficient so that it can be employed in large datasets within a reasonable time frame. Also, the proposed algorithm provides an optimal (with respect to distribution entropy) discretization scheme that groups values into intervals containing data with similar distributions. In particular, we hereby propose a discretization algorithm that is:

- multivariate (considers the relationships between all attributes for choosing an interval)
- unsupervised (independent of class labels of data)
- static (executed in the preprocessing step before the learning stage)
- direct (determines multiple intervals at once)
- global (evaluates the entire dataset for picking the best interval among all possible ones)

1.2. Organization

The organization of the rest of the thesis is as follows:

- Chapter 2 provides definitions and background knowledge about discretization and its properties.
- Chapter 3 gives an overview of related work on discretization and demonstrates a brief taxonomy of existing methods.
- Chapter 4 introduces our proposed method and describes how our algorithm works.
- Chapter 5 demonstrates the experiments performed with our method and their results in detail.
- Chapter 6 states the summary of the thesis and possible future directions.

2. DEFINITIONS AND BACKGROUND

2.1. Formal Description

A general description of a discretization process is as below:

Assuming a dataset S consisting of N observations, we can discretize a continuous attribute A by splitting it into k intervals and obtain a discretization scheme $D = \{[d_0, d_1), [d_1, d_2), \ldots [d_{k-1}, d_k]\}$ where d_k is maximum and d_0 is minimum value that exists in A; and D is sorted on attribute A.

A general discretization process (as in Figure 2.1) consist of 4 steps: Sorting, Searching (Evaluation), Splitting (or Merging) and Stopping. Sorting process is generally expected to be performed only once in the beginning and in an efficient way using an algorithm of time complexity of $O(NlogN)$ at maximum. Search (evaluation) part is the stage of finding the most optimal cut points by means of an evaluation function. Splitting (or Merging) part divides a subinterval into two or merges two adjacent intervals if merging approach is followed. Stopping part is a stage where it is checked if any further splitting/merging is required or not, according to a stopping criterion determined by discretization method.

2.2. Properties of Discretization Methods

- **Static/Dynamic:** Static methods are independent of what learning algorithm is used, and are executed in the preprocessing stage before the learning stage starts. A vast majority of discretizers are of static type. Dynamic discretizers (for example ID3 decision tree discretizer) are built into learning algorithms and executed while learning is done.

- **Univariate/Multivariate:** Univariate discretizers only focus on a single attribute at a time without taking other continuous attributes into consideration. Multivariate discretizers take all continuous attributes into account to determine cut-points characterizing the discretization scheme. As such, multivariate discretizers aim to capture relations between attributes and determine cut points accordingly.

- **Supervised/Unsupervised:** Supervised discretization algorithms take the class labels of target variable into account. This enables such discretizers to discover the relationship between an attribute and the class label (e.g. entropy, correlations etc.) Supervised discretizers can be used only in supervised learning tasks whereas unsupervised counterparts can be used in both supervised and unsupervised learning.

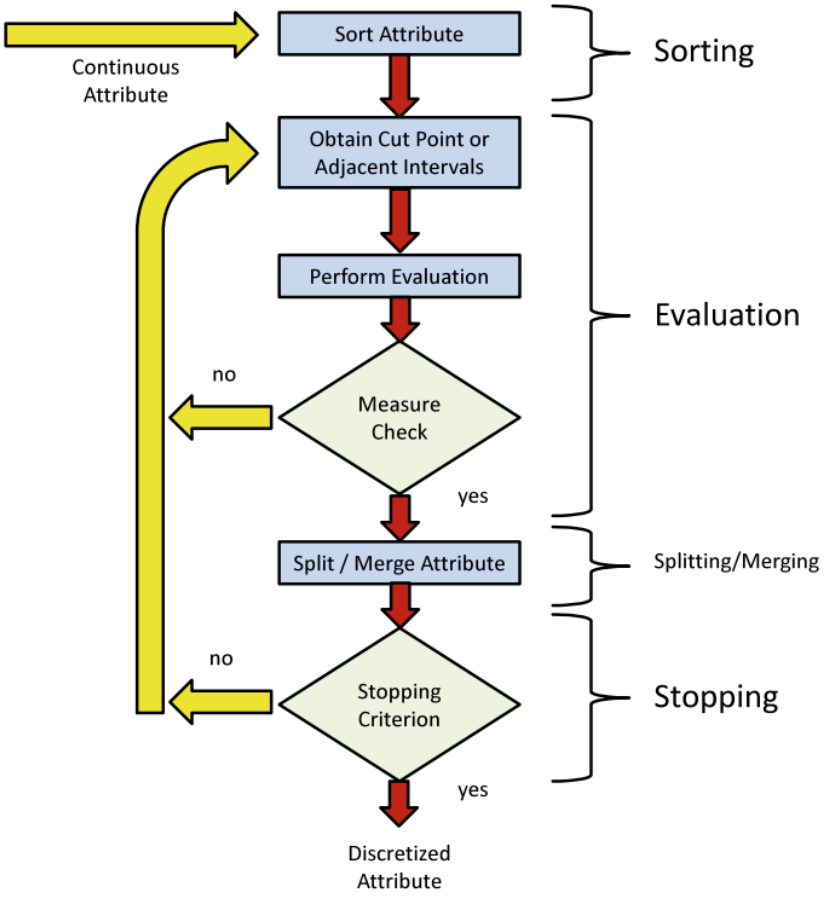


Figure 2.1 The overview of the steps of discretization [1].

- **Splitting/Merging:** After the evaluation step, a new interval is decided either by splitting a larger interval into two (e.g. top-down approach) or combining two smaller intervals into a larger one (e.g. bottom-up approach). There are also examples of hybrid approaches using both operations.

- **Global/Local:** Local discretizers take only a subset of data (i.e. the data exists in the currently evaluated sub-interval) into account during evaluation. Global discretizers, on the other hand, has access to the entire dataset to evaluate possible cut-points. To the best of authors' knowledge, most of the known discretizers are global. Yet, dynamic discretizers are local by definition [4]. For example, as ID3 uses a dynamic approach, the internal discretizer is able to access only the currently splitted data, therefore it is a local discretizer. ME-MDLP [6] can be given as another example to local discretizers.

- **Direct/Incremental:** Direct discretizers determine multiple cut-points at once. The incremental approach on the other hand, establishes a simple discretization initially and improves it further over iterations, either splitting the interval more or merging with a neighbor interval, and stops it when its stopping criterion is satisfied.

- **Evaluation Measure:** During the search (evaluation) stage, multiple candidate discretization schemes are compared according to an evaluation measure. There are five main types of measures:

  - *Information:* Mostly the entropy measure defined in information theory and some others such as Gini Index, Mutual Information are used in many discretizers like MDLP [6], ID3 [2].

  - *Statistical:* Dependency and correlation between attributes are statistically evaluated (e.g. Chi2 [7])

  - *Rough Sets:* Rough set methods, boolean reasoning, lower and upper approximations are used [8].

  - *Wrapper:* A classifier (e.g. Naive Bayes) is executed in each iteration and the error rate is used to evaluate.

  - *Binning:* Bins (intervals) are predefined and there is no evaluation measure. EqualWidth (each range has the same length) and EqualFrequency (each range includes the same number of observations) are examples for binning.

- **Parametric/Non-parametric:** Discretizers such as ME-MDLP [6] does not require manual input of desired intervals and automatically optimize the number of bins according to their stopping criterion. An example of a parametric method is ChiMerge [9] which requires an upper limit for the number of intervals.

3. RELATED WORK

3.1. Supervised Discretizers

Kerber [9] proposed ChiMerge which is a supervised and univariate algorithm based on statistical χ^2 test. It tries to produce such intervals that within each interval target class frequencies should be consistent and the class frequencies in different intervals should not be similar. It follows bottom-up strategy, that is there are intervals to the number of unique values at the beginning, and adjacent intervals with least χ^2 value are merged. Also, the merging step is controlled by using a threshold, which indicates the maximum χ^2 value that permits two neighbor intervals to be merged. The stopping criterion is determined manually with a maximum number of intervals given by user.

Fayyad and Irani [6] proposed ME-MDLP (Minimum Entropy-Minimum Description Length Principle) which is a supervised, univariate and splitting discretizer and it is one of the most basic discretizers proposed so far. Evaluation measure is based on entropy feature and Minimum Description Length Principle (MDLP) described in Information Theory. Basically, it recursively splits intervals until the entropy of interval gets minimized. MDLP is defined as "the minimum number of bits required to uniquely specify an object out of the universe of all objects." [1] The stopping criterion is the same as the evaluation measure. Since it is univariate, it generates intervals of an attribute regardless of its possible dependencies with other attributes and only takes account of target class labels.

3.2. Unsupervised Discretizers

3.2.1. Univariate Discretizers

Kontkanen and Myllymäki [10] proposed UD which is a univariate and unsupervised method based on MDL (minimum description length) principle like Fayyad and Irani's ME-MDL method.

Vannucci and Colla [11] proposed an unsupervised and univariate method by making use of SOMs (Self-organizing maps). They compared their method to classical kNN-binning approach and demonstrated lower error scores.

Schmidberger and Frank [12] proposed TUBE (Tree-based Unsupervised Bin Estimator), an unsupervised and univariate method which makes use of Tree-based Density Estimation. Their technique tries to construct an estimated density function without using any parameters. Using cross validation, the algorithm adapts widths of bins to the data.

Biba et al. [13] proposed an unsupervised, univariate and top-down (splitting) method using Kernel Density Estimation. It makes use of cross-validation of the log-likelihood for selecting the number of intervals, and kernel density estimation for selecting the cut-points.

Ferreira and Figueiredo [14] proposed U-LBG, which is an unsupervised version of well-known LBG (Linde–Buzo–Gray) Vector Quantizer Algorithm [15].

3.2.2. Multivariate Discretizers

Bay [16] proposed MVD which is an unsupervised multivariate discretization technique and compared it to previous supervised methods like ME-MDL proposed by Fayyad and Irani [6]. Bay argues that ME-MDL might perform well at classification, but it is not useful at data discovery and analysis tasks. This technique prioritizes that instances in each interval should have similar distributions among its attributes. Thus, instead of focusing on increasing predictive accuracy, it pursuits exploring hidden patterns and extracting semantically meaningful information for humans. For example, an interval of [$26K-$80K] for yearly income on census data might result in good scores in prediction, but it is not giving much exploratory information because it hides various groups in terms of education, occupation etc. This method was evaluated qualitatively and showed success in information extraction like discovering hidden patterns at UCI Admissions data. In addition, it outperformed ME-MDL in terms of CPU execution time.

Mehta et al. [17] proposed CPD (correlation preserving discretization) which is an unsupervised, dynamic, and multivariate method based on PCA. Since the correlations among all attributes are "intrinsically" preserved with the help of PCA algorithm, they proposed an efficient method for highly dimensional and large data sets. They also compared their performance with the prior method that Bay offered [16]. What they define as a meaningful and correlation-preserved interval is "instances within an interval exhibiting similar properties, and instances in different intervals exhibiting different properties". They tackled the problem of capturing correlations between continuous and categorical attributes by means of combining PCA with *association rules mining*. Also, CPD method is a dynamic discretization technique that takes account of all attributes at the same time.

Nguyen et al. [18] proposed IPD (interaction-preserving discretization) which aims at not losing dependencies among attributes while discretizing them. That is, "two multivariate regions should only be in the same bin if and only if the objects in those regions have similar multivariate joint distributions in the other dimensions. That is, we enforce each bin to only contain data of similar distributions." In this work, IPD was compared against earlier methods like CPD, MVD, ME-MDL in terms of classification accuracy with Random Forest classifier. IPD does not result in lower scores than classification scores of non-discretized original data as well as it rather produces higher scores in some data sets. Also, in most cases it outperformed CPD, ME-MDL and MVD.

4. PROPOSED METHOD

We propose a technique to discretize a continuous attribute using dynamic programming and multivariate information, and our discretizer has the following properties:

1. Multivariate (considers the relationships between all attributes for choosing an interval)
2. Unsupervised (independent of class labels of data)
3. Static (executed in the preprocessing step before the learning stage)
4. Direct (determines multiple intervals at once)
5. Global (evaluates the entire dataset for picking the best interval among all possible ones)

4.1. Overview of Method

Let S denote a dataset, that consists of N observations, and M be its set of attributes. We want to discretize the continuous attribute $A \in M$ by splitting it into L non-overlapping intervals, where L is between predetermined thresholds *MIN_BINS* and *MAX_BINS*. That is, the resulting discretization scheme must consist of L intervals, where *MIN_BINS* $\leq L \leq$ *MAX_BINS*, and L is to be determined by an optimization procedure (to be explained later) rather than being a predetermined parameter. Also, let $U = \{u_i \in A | i = 1 \dots k\}$ denote the unique set of values that exist in A, sorted in ascending order, i.e. $u_i < u_j$ for any $i < j$. Next, we build a directed acyclic graph $Q = (V, E)$ with vertices $V = \{v_i \mid v_i = u_i;\ u \in U; i = 1 \dots k\}$ and edges $E = \{e_{i,j} = (v_i, v_j) \mid i < j; i, j = 1 \dots k\}$ where each vertex in V is connected to all other vertices with an edge $e_{i,j}$ where $i < j$. By this definition, we can observe that E contains $\frac{k(k-1)}{2}$ edges.

Given that U is sorted, the graph Q is always a directed acyclic graph (DAG), that is it has no cycles. Moreover, it is guaranteed that it has a single source node (whose in-degree is zero)

denoted by v_1 and a single sink node (whose out-degree is zero) denoted by v_k. Observe that, each edge $e_{i,j} = (v_i, v_j)$ in E actually represents an interval $[u_i, u_j)$ within A. We assign a cost based on multivariate entropy to each edge (interval) whose calculation will be explained in Section 4.2.. Then, it can be seen that a path (a sequence of edges) starting from the source node v_1 and ending at the sink node v_k will represent a discretization scheme, and any discretization scheme can be represented by a path on this graph. Thus, the problem of finding an optimal discretization scheme is converted into finding an optimal (with respect to edge costs) path *P* in graph Q where P starts from v_1 and ends at v_k, also it is of length[1] L and *MIN_BINS* $\leq L \leq$ *MAX_BINS*.

Let us illustrate this with a small example. A sample set S taken from the well-known Iris dataset is given in Table 4.1. It consists of four attributes and fifteen rows.

A (sepal length)	B (sepal width)	C (petal length)	D (petal width)
4.8	3.1	1.6	0.2
6.9	6.9	6.9	6.9
5.3	5.2	5.2	5.2
6.3	6.3	6.3	6.3
7.7	7.6	7.4	7.3
6.7	6.7	6.6	6.6
5.5	4.2	1.4	0.2
6.1	6.1	6.1	6.1
7.7	7.7	7.7	7.7
6.3	6.3	6.3	6.3
5.7	5.7	5.6	5.6
6.7	6.7	6.7	6.7
6.9	6.8	6.8	6.8
5.5	2.3	4.0	1.3
4.8	3.4	1.6	0.2

Table 4.1 A sample subset S from Iris dataset with 15 observations.

Suppose that attribute A is to be discretized whose values are as follows:

$$\{4.8, 6.9, 5.3, 6.3, 7.7, 6.7, 5.5, 6.1, 7.7, 6.3, 5.7, 6.7, 6.9, 5.5, 4.8\}$$

[1] In our context, the "length" of a path means the number of nodes included in a path, in other terms "hop count", not the total weights of that path's edges.

Next, U is produced with the unique values in this set in ascending order:

$$U = \{4.8, 5.3, 5.5, 5.7, 6.1, 6.3, 6.7, 6.9, 7.7\}$$

A directed graph $Q = (V, E)$ is generated out of U where $V = \{v_1 = 4.8, v_2 = 5.3, v_3 = 5.5, v_4 = 5.7, v_5 = 6.1, v_6 = 6.3, v_7 = 6.7, v_8 = 6.9, v_9 = 7.7\}$ as depicted in Figure 4.1.

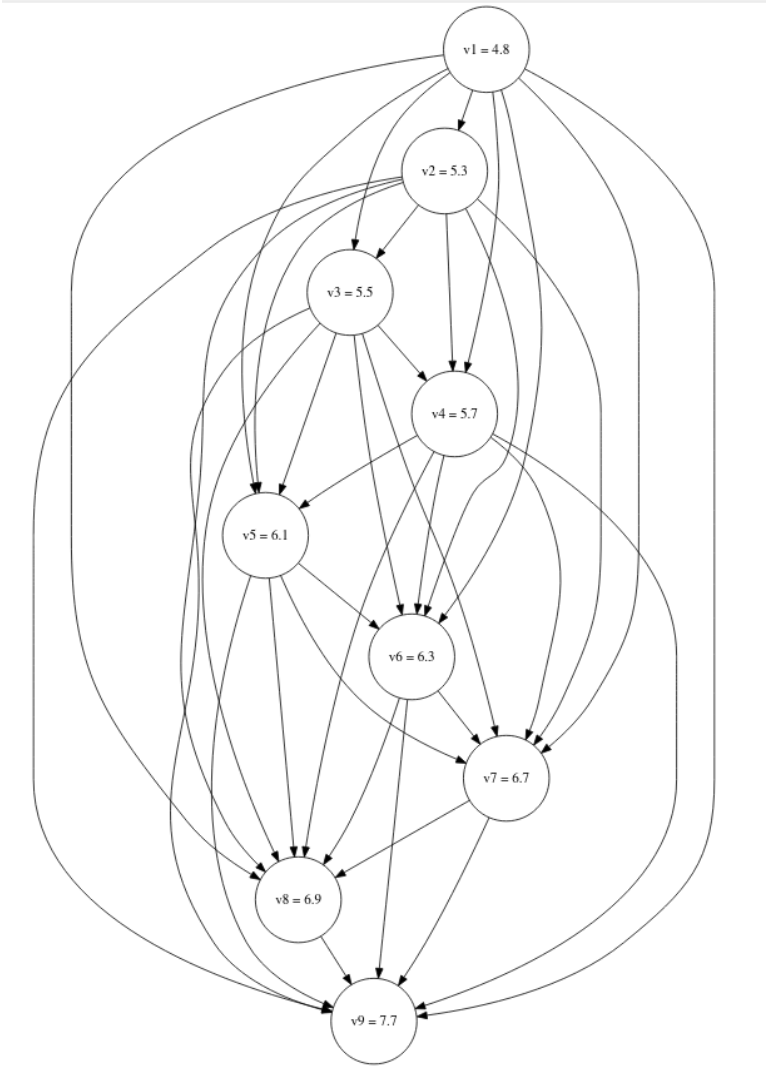


Figure 4.1 An example graph Q generated out of unique values in the attribute to be discretized.

Notice that, any path *P* starting from $v_1 = 4.8$ (source node) and ending at $v_9 = 7.7$ (sink node) in fact corresponds to a discretization scheme. For example the path $P = \{e_{1,4}e_{4,6}e_{6,8}e_{8,9}\}$

establish a discretization scheme $D = \{[4.8, 5.7), [5.7, 6.3), [6.3, 6.9), [6.9, 7.7]\}$ as demonstrated in Figure 4.2 (Path P is marked with red, bold lines).

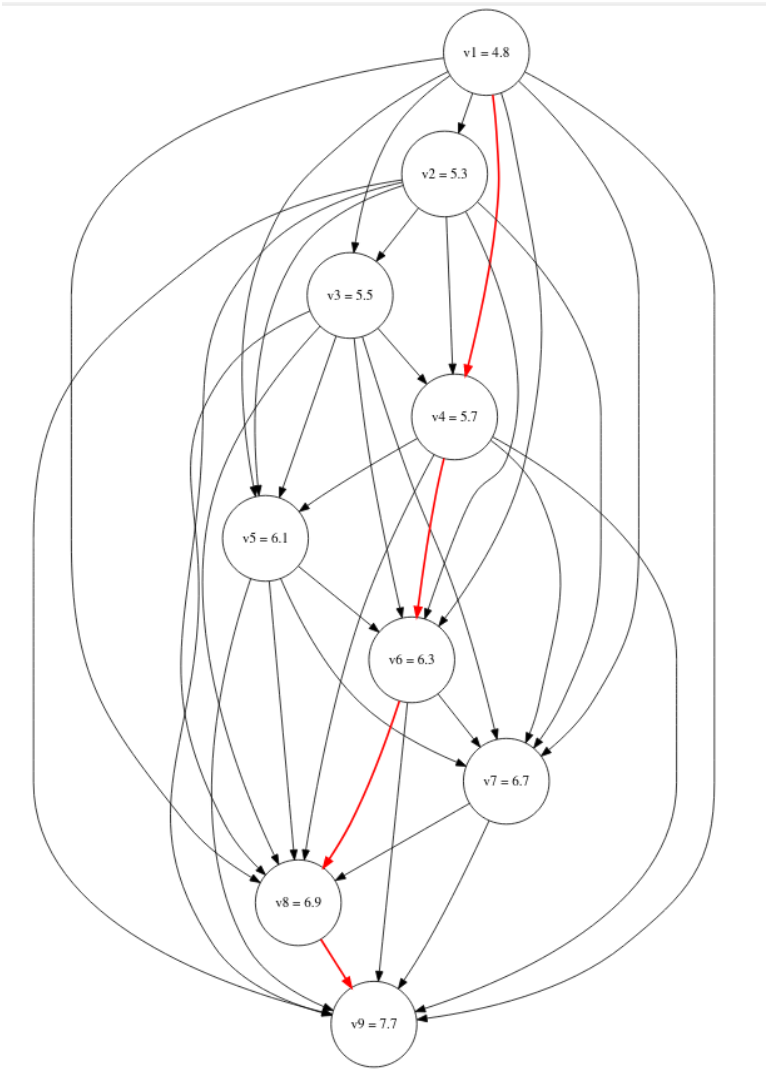


Figure 4.2 An example path in Q (marked with red, bold lines) which corresponds to a discretization scheme.

At this point, our objective is to find an *optimal* path among all possible paths. An *optimal* path should meet the two criteria below:

1. The total cost (the sum of edge costs in the path) is the minimum among all paths.
2. The length (i.e. hop-count) is within interval [*MIN_BINS, MAX_BINS*].

In order to meet the first criterion, a standard shortest path algorithm is sufficient. Nevertheless, our second criterion makes it necessary to perform a *constrained shortest path* procedure. A standard shortest path algorithm (e.g. Dijkstra's algorithm) does not apply bounds on the hop count, as its only objective is to find the path with minimum total edge cost. However, in discretizing an attribute, as the intervals get smaller, the entropy of the corresponding observations with respect to the other attributes gets lower. Therefore, the shortest path almost always consists of the shortest possible edges (intervals of two adjacent unique values). We, therefore, need to put bounds on the path's length to restrict the number of bins for discretization. Hence, we iteratively solve a dynamic programming procedure to find the optimal path within the preassigned length bounds, by gradually modifying the edge costs at each iteration until we find a path with length between the bounds.

Our search for the shortest path does not work in a brute-force manner, rather we apply a customized optimization algorithm for minimizing the number of iterations. During each iteration of the search procedure, we modify the edges by adding the same penalty to the original weights of all the edges. The additional penalty enforces the shortest path algorithm to generate another path with less number of edges than the path found with original edge weights. The main reason is that an increase in each edge's weight adds up to a larger total path cost, therefore the shortest path algorithm tends to find a path with less number of edges to minimize the path cost. Therefore, the relation between path length and penalty is inverse.

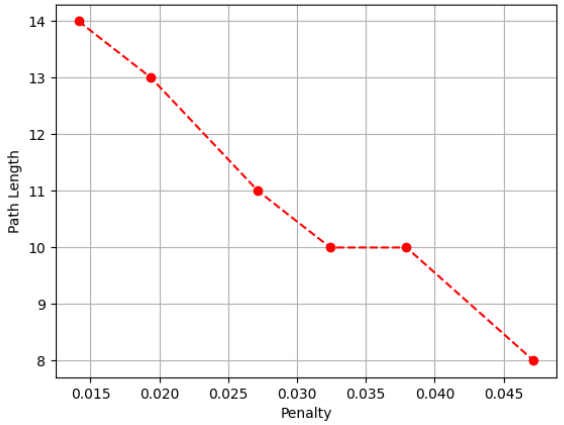


Figure 4.3 The visualization of an example execution of optimal path algorithm.

Iteration	Penalty	Path Length	Path (cutpoints)
0	0.01413	14	$(50, 71, 76, 83, 89, 94, 100, 105, 110, 114, 117, 123, 131, 145)$
1	0.01940	13	$(50, 71, 76, 83, 89, 94, 100, 105, 110, 117, 123, 131, 145)$
2	0.02717	11	$(50, 76, 83, 89, 96, 104, 109, 114, 123, 131, 145)$
3	0.03241	10	$(50, 76, 84, 91, 100, 106, 114, 123, 131, 145)$
4	0.03795	10	$(50, 76, 84, 91, 100, 106, 114, 123, 131, 145)$
5	0.04718	8	$(50, 76, 88, 100, 106, 114, 123, 145)$

Table 4.2 The gradual decrease steps of an example execution of the path optimization algorithm.

As seen in Figure 4.3, adding larger penalties to edge weights caused the computed shortest path to have less number of edges. This search continues until a path that has *MAX_BINS* or less edges is found. In case the path length drops below *MIN_BINS* due to a sharp increase in the penalty, we start gradually decreasing the penalty to produce a path with more edges. For example, as seen in Figure 4.3 as well as Table 4.2, the threshold *MAX_BINS* $= 7$ causes the search procedure to stop once a path of length 8 (with respect to node count) is found.

4.2. Entropy-based Edge Weight Calculation

The weights assigned to the edges in this graph play a critical role in our method, because our final discretization scheme will be selected according to the edge weights by means of the shortest path algorithm. We should assign weights to the edges in such a way that the shortest path can lead us to an optimal discretization scheme among other possible ones. Remember that each edge actually represents an interval for discretization. For example, the edge $e_{3,8}$ from $v_3 = 5.5$ to $v_8 = 6.9$ corresponds to the interval $[5.5, 6.9)$ in our example graph in Figure 4.2.

For the calculation of edge weights, we start with normalizing each attribute in itself for preventing scale based misjudgments in further computations. Since we are willing to establish a multivariate discretization that preserves the correlations among attributes as much as possible, we compute the weight of each edge $e_{i,j}$ from v_i to v_j based on the entropy notion in information theory. *Information entropy*, which was introduced by Shannon [19], can be simply described as the impurity of a set of observations and defined by the following

expression:

$$H(X) = -\sum_{x \in X} p(x) \lg p(x) \tag{1}$$

In our graph, the cost of an edge with respect to attribute A indicates whether the observations within this interval show similar properties ("pure": low entropy) or not ("impure": high entropy) within the other attributes. A lower cost edge denotes a set of observations that are more like each other in terms of attributes other than A. For example, in the Iris dataset (Table 4.1), if an interval defined on sepal length consists of observations that are very similar in sepal width, petal length and petal width, then we call this interval pure as it has very low total entropy with respect to the B, C, and D attributes.

Now, let us consider two paths in Q: P_1 and P_2, each mapping to certain discretization schemes in A. Now, suppose that the total cost of P_1 is less than that of P_2. This in turn means that the intervals corresponding to the edges in P_1 are more pure than those in P_2. Here, a path being more pure than another represents its total entropy-based cost being less. We argue that a good discretization scheme is one that uses intervals having lower entropy levels, and we aim to optimize the total entropy level of all intervals.

Next, we provide the formula for $W_{i,j}$, the entropy-based edge cost calculation (Equation 2) we use. Notice that $C = M - A$ denotes the set of all attributes other than attribute A; $N_{i,j}$ the number of observations within interval $[i, j)$ with respect to A; N the total number of observations in the dataset; and $\mathbb{H}$ denotes the intervals in the equal width discretization of the distinct values in attribute c that falls into the interval $[i, j)$.

$$W_{i,j} = \frac{N_{i,j}}{N} \times \sum_{c \in C} \sum_{x \in \mathbb{H}} -p(x) \lg p(x) \tag{2}$$

Here, note that we use the equal width discretization to determine the intervals of attributes in C while computing the intervals of A with our method. That is because our method is based on the entropy of the other attributes which can only be meaningfully computed if those attributes are already discretized. Therefore, if the other attributes are given categorically,

we simply use the provided categorical values. However, if the other attributes are continuous, we then assume a discretization of them based on the fast equal width discretization. Therefore, the $\mathbb{H}$ in the formula represents the intervals (categories) obtained this way.

4.2.1. Histogram Matrices

In order to calculate the proportion of each interval in $\mathbb{H}$, denoted as $p(x)$ in Equation 2, we use histograms. A histogram consists of the counts of how many values fall into each interval provided that the histogram's intervals are defined as equal-width and non-overlapping. However, repeatedly computing histograms during edge-weight calculation is a costly operation. We prefer an iterative and incremental approach and for each attribute in C, we generate a *histogram matrix* before computing edge weights. A histogram matrix is a special data structure that keeps all histogram information for an attribute in the dataset and reduces the computational cost of histogram calculation.

Row #	A	B	C
1	4.8	1.59	0.77
2	4.8	−0.90	−0.22
3	4.8	−1.22	0.54
4	5.3	−0.16	−0.96
5	5.5	2.25	−0.43
6	5.5	2.03	1.81
7	5.6	1.70	0.15
8	5.7	2.39	1.24
9	6.1	−0.54	0.51
10	6.3	−1.18	0.48
11	6.3	−1.13	−1.61
12	6.7	−0.44	−0.37
13	6.9	−0.20	−0.75
14	6.9	−0.31	−1.08
15	7.7	−1.89	−0.07

⇒

$[-1.89, -0.45)$	$[-0.45, 0.97)$	$[0.97, 2.39]$
0	0	1
1	0	1
2	0	1
2	1	1
2	1	2
2	1	3
2	1	4
2	1	5
3	1	5
4	1	5
5	1	5
5	2	5
5	3	5
5	4	5
6	4	5

Table 4.3 A sample dataset with 15 observations (left) and the histogram matrix of Attribute B with 3-bins (right).

To explain more clearly, see the small dataset consisting of 15 observations in Table 4.3. If we want to calculate histograms with three equal-width bins for attribute B, the intervals will be $[-1.89, -0.45), [-0.45, 0.97), [0.97, 2.39]$ since the minimum and the maximum values

of B are -1.89 and 2.39. For this, we generate a histogram matrix for B (the one on the right in Table 4.3) that holds the counts of observations whose values of attribute B fall into these ranges. Notice that a row in the histogram matrix holds the histogram for only the set of observations up to that row. For instance, 7th row of histogram matrix only shows the histogram calculated with only the first seven observations in the dataset.

A histogram matrix allows us to calculate histograms in constant time. For example, suppose that we want to calculate attribute B's histogram for the observations in the range $[5.5, 6.3)$ with respect to attribute A. We find out that the corresponding observations are with row number 5, 6, 7, 8, and 9 (separated with lines in the table). The histogram of that set of observations can be calculated (in constant time) by subtracting 9^{th} row $[3, 1, 5]$ from 4^{th} row $[2, 1, 1]$ in our histogram matrix. We obtain a histogram $[1, 0, 4]$ with respect to attribute B. It denotes that one observation falls into $[-1.89, -0.45)$ bin, no observations into the $[-0.45, 0.97)$ bin, and four observations into the last bin.

4.3. Main Discretization Algorithm

Algorithm 1 presents our main discretization algorithm. It takes several parameters. S is the main dataset which consists of rows as observations and columns as attributes (whether continuous or non-continuous). A is the column index number of attribute to be discretized. *MAX_BINS* and *MIN_BINS* are parameters setting a threshold in order for the algorithm to search for a discretization scheme having cut-points not more or less than those respective limits. However, among different possible discretization schemes, always the one nearest to the *MAX_BINS* will be selected (see Section 4.4.). *N_PCA* signifies how many attributes should be generated out of attributes by PCA dimensionality reduction algorithm.

Algorithm 1 Main discretization function

FUNCTION *DISCRETIZE*(S, A, MAX_BINS, MIN_BINS, N_PCA):

Apply normalization (or standardization) to all attributes of S other than A.
$S' \leftarrow$ A new matrix comprising A and *N_PCA* new attributes generated by PCA applied to all attributes of S other than A.
Sort rows of S' by attribute A in ascending order.
HM $\leftarrow$ Calculate histogram matrices of each attribute except for attribute A
$U \leftarrow$ unique set of values that exist in A, sorted in ascending order
$V \leftarrow \{v_i | v_i = u_i; u \in U; i = 1 \dots k\}$ ▷ Generate a vertex for each value in U and assign it to vertex set V
$Q \leftarrow$ Build a graph with vertices V

if $|U|/5 \leq$ *MIN_BINS* **then**
    return ▷ A cannot be discretized due to very low uniqueness of values
end if
for $i = 0$ to $|U|$ **do**
    for $j = i + 1$ to $|U|$ **do**
        if $N_{i,j} < N/100$ **then** ▷ $N_{i,j}$: number of observations within interval $[i, j)$ with respect to A
            continue to loop, disregard this interval
        end if
        TotalEntropy $\leftarrow 0$
        for $c \in$ columns of S' except for A **do**
            $\mathbb{H} \leftarrow$ *HM*$[c]$ ▷ Take histogram of observations in range $[u_i, u_j)$ with respect to A, in attribute c's histogram matrix
            EntropyInRange $\leftarrow \sum_{x \in \mathbb{H}} -p(x) \lg p(x)$ ▷ p(x) denotes the ratio of each entry in histogram
            RatioOfRange $\leftarrow \frac{N_{i,j}}{N}$
            TotalEntropy $\leftarrow$ *RatioOfRange* $\times$ *EntropyInRange*
        end for
        Add a directed edge $e_{i,j}$ from v_i to v_j with a weight of *TotalEntropy*
    end for
end for

Path $\leftarrow$ Call *FIND_OPTIMUM_PATH* prodecure and return the path
return *Path*$[1 : -1]$ ▷ Discard first and last values as they are not required for discretization

We explain our main discretization algorithm (Algorithm 1) step by step.

1. (Line 1) We start by scaling each attribute in itself for preventing possible biases for further computations. (either standardization or normalization) Standardization tries to centralize dataset in such a way that mean of numbers become zero and the standard deviation becomes 1. On the other hand, normalization scales all values between 0 and 1. Since both can be useful in different scenarios we left it to the user's preference.

2. (Line 2) We apply a dimensionality reduction technique called PCA (Principal Component Analysis) which takes into account all attributes other than attribute A to produce a reduced number of new attribute set which consists of *N_PCA* attributes. Once PCA provides derived attributes, we merge them (along with our attribute A as the first column) into a new matrix S'.

   The rationale behind this step is as follows: Since we are willing to establish a multivariate discretization which preserves correlations among attributes as much as possible in order to not lose information, the correlation of each attribute with attribute A need to be evaluated. This might result in a high computational cost especially in highly dimensional datasets with tens or hundreds of attributes. Hence, we need to reduce the complexity of dataset by means of PCA in order to minimize the computational cost while not losing too much information that lies behind the correlations between attributes.

3. (Line 3) We sort all rows (observations) according to attribute A in increasing order (we assume that A is not a categorical variable, otherwise it cannot be discretized).

4. (Line 4) For our entropy-based edge weight calculation, we need each column's detailed histogram data, but computing it each time is a costly operation. Thus we calculate *histogram matrix* of each attribute in S' in advance. We define *histogram matrix* to be a special data structure which reduces computational cost during histogram calculation. Thus, other than attribute A, histogram matrices of rest of attributes are computed in advance and stored in memory. The rationale behind was explained in Section 4.2..

5. (Line 6-7) Build a graph with values in U, do not add any edges yet as they will be added after edge weights are calculated.

6. (Lines 9-11) Control whether the attribute is eligible for discretization. If number of unique values in attribute A is less than MIN_BINS $\times 5$, our heuristic implies that such a discretization will not be meaningful because it has very low rate of uniqueness and might be a categorical variable.

7. (Lines 12-13) We will search for all possible intervals in attribute A, thus we perform two nested for-loops.

8. (Lines 14-16) If the number of values in range $[u_i, u_j)$ is too small, we do not want to include this interval to our final discretization scheme.

9. (Lines 17-23) For calculating an edge's weight as formulated in Equation 2, we perform the following steps:

    - We initialize *TotalEntropy* variable with zero.
    - We calculate the ratio of observations in range $[i, j)$ with respect to A to all observations. Denoted as $\frac{N_{i,j}}{N}$.
    - For each column c in S' (the ones produced by PCA) other than attribute A:
        (a) We calculate the histogram of current range by subtracting two rows from histogram matrix (as we explained above).
        (b) We calculate the entropy of resulting histogram with classical entropy formula.
        (c) We multiply the ratio with the entropy and add it to *TotalEntropy*.

10. (Line 24) We connect two nodes v_i and v_j with a directed edge, and assign *TotalEntropy* as its weight.

11. (Lines 28-30) We perform Path Optimization Algorithm which will be explained in detail in Section 4.4., and try to find a shortest path from the minimum value to the

maximum value in our newly constructed graph, suitable to our constraints by performing Algorithm 2. For optimizing performance, we suggest the iteration limit to be 100 and step size to be 0.01.

12. (Line 31) Resulting path consists of the minimum and maximum values in attribute A, and cut-points in between. We discard the first and the last values and only return cut-points.

4.4. Path Optimization Algorithm

Parameters of our procedure is as follows:

- Q: Original graph with unique values in attribute A as nodes
- *SOURCE*: Starting node for our optimal path
- *DEST*: Destination node for our optimal path
- *MAX_LENGTH*: Max. length of optimum path
- *MIN_LENGTH*: Min. length of optimum path
- *STEP_SIZE* is a factor to calculate the magnitude of a step which will be multiplied later with a random number between $[0.0, 0.1)$. The calculated step will be added to (or subtracted from) last found best penalty and candidate penalty will be used to find an optimal path. We advice to use 0.01 as default.
- *ITER_LIMIT* prevents very long executions if it's difficult to find a path with required thresholds. We use 100 as the limit even though our algorithm usually finds an optimum path in no more than 10 iterations.

Algorithm 2 Algorithm for finding an optimal path

FUNCTION *FIND_OPTIMUM_PATH* (Q, SOURCE, DEST, MAX_LENGTH, MIN_LENGTH, STEP_SIZE, ITER_LIMIT):

DoClimb ← False
Penalty ← 0
Path ← NULL
PathLength ← 0
while *Path* = NULL **do** ▷ Generate and evaluate initial penalty
    Penalty ← Pick random number between 0 and 0.1 with uniform distribution
    Q_{P} ← Clone graph Q and add *Penalty* to each edge's weight
    Path, PathLength ← SHORTEST_PATH (Q_{P}, *SOURCE, DEST*)
    DoClimb ← *PathLength* ≤ *MAX_LENGTH*
end while
Iter ← 0
while *Iter* ≤ *ITER_LIMIT* **do**
    Step ← Pick random number between 0 and 0.1 with uniform distribution
    if *DoClimb* **then**
        Step ← − *Step* × *STEP_SIZE*
    else
        Step ← *Step* × *STEP_SIZE*
    end if
    CandidatePenalty ← *Penalty* + *Step*
    Q_{CP} ← Clone graph Q and add *CandidatePenalty* to each edge's weight
    CandidatePath, CandidatePathLength←SHORTEST_PATH(Q_{CP}, *SOURCE, DEST*)
    NewOptimumFound ← False
    if not *DoClimb* **then** ▷ in descending state
        if *MIN_LENGTH* ≤ *CandidatePathLength* ≤ *PathLength* **then**
            NewOptimumFound ← True
            if *CandidatePathLength* ≤ *MAX_LENGTH* **then**
                DoClimb ← True
            end if
        end if
    else ▷ in climbing state
        if *PathLength* ≤ *CandidatePathLength* ≤ *MAX_LENGTH* **then**
            NewOptimumFound ← True
        end if
    end if

```
35:     if NewOptimumFound then                                   ▷ Update current optimum path
36:         Penalty,Path,PathLength←CandidatePenalty,CandidatePath,CandidatePathLength
37:     end if
38:
39:     if PathLength = MAX_LENGTH then
40:         break                                  ▷ Stop iterating, we reached a path with optimum length
41:     end if
42:
43:     Iter ← Iter+1
44: end while
45:
46: if PathLength < MIN_LENGTH or PathLength > MAX_LENGTH then
47:     return NULL                  ▷ the found path is out of aimed interval, it means no optimal path
48: else
49:     return Path, PathLength
50: end if
```

The algorithm shown in pseudo-code in Algorithm 2 is explained below:

- We store our current best found path in *Path* and it's length in *PathLength* and the penalty which causes that path to be generated in *Penalty*.
- The direction of search is stored in *DoClimb* flag variable. If execution is in "descend" state, (i.e. *DoClimb* = *False*), it means the path search will be forward (since the path length-penalty correlation is a decreasing function, see Figure 4.3), that is it will be seeking a shorter path than the last found best path by means of a higher penalty. If the search is in "climb" state (i.e. *DoClimb* = *True*), it means the path search will be backward, that is a longer path than the last found best path is searched by means of a lower penalty.
- (Lines 1-4) We start with initializing our variables.
- (Lines 5-10) We randomly pick a starting point for penalty, and we clone the graph and add the same penalty to each edge of the cloned graph, then calculate the shortest

path. If the resulted path is higher than our *MAX_LENGTH* threshold with respect to length, we go into "descend" state. Otherwise, we enter "climb" state. (by means of setting *DoClimb* flag)

- (Lines 11-12) We enter main loop and continue until the *ITER_LIMIT* is reached if the loop is not exited early in case of finding an optimal path.

- (Lines 13-19) We calculate *Step* by multiplying a random number between $[0, 0.1)$ and *STEP_SIZE*. If we are in "climb" state, we subtract *Step* from current *Penalty* (as we wish to decrease penalty in order to produce a longer path). If we are in "descend" state, we aim at producing a shorter path by increasing penalty, thus we add *Step* from current *Penalty*. The resulting penalty is saved temporarily in *CurrentPenalty* as we will discard it if it does not bring any improvement on our last found best path.

- (Lines 20-21) We clone the graph and add *CandidatePenalty* to each edge of the cloned graph, after which we calculate the shortest path and assign *CandidatePath* and *CandidatePathLength* variables.

- (Lines 23-29) If we are in "descend" state in the current iteration, we check whether *CandidatePathLength* is in between our *MIN_LENGTH* and *PathLength* (most optimal path found so far) because "descend" state's aim is always finding a shorter (or equal) path. If this condition is true, we record this by setting *NewOptimumFound* to *True*. Also, we check whether *CandidatePathLength* is less or equal than *MAX_LENGTH*. If it holds true, we change our state to "climbing". The reason is that we always favor paths as close as possible to the our maximum length. Thus, after obtaining a candidate optimal path within our thresholds, we do not greedily accept this path to be a real optimal. Rather, we continue our search backwards for finding a longer path than current optimal, but shorter than (or equal with) *MAX_LENGTH*.

- (Lines 31-34) If we are in climbing state, check if *CandidatePathLength* is between last found optimal path length and *MAX_LENGTH*. If it is, we record this as a new improvement because we managed to find a longer path than current optimal but also

shorter than (or equal to) *MAX_LENGTH*. As we mentioned earlier, our optimization algorithm always favors the closest path length to *MAX_LENGTH*.

- (Lines 36-38) If we recorded that this iteration brought an improvement on optimum path, we assign all candidate variables to current most optimal path's variables and accept it to be the most optimal path found so far.
- (Lines 40-42) If our most optimal path length is equal to *MAX_LENGTH*, we exit the loop as there remains no need to search for another path.
- (Lines 47-51) If most optimal path found is within our thresholds, return it. Otherwise, return nothing.

5. EXPERIMENTAL RESULTS

We performed two types of experiments: the first is performed on synthetic data whereas the second is performed on real datasets. In our experiments, we compare our method with three common discretization methods: *equal-width*, *equal-frequency*, and *k-means binning* [2]. Unlike our method, all benchmark methods are univariate and they do not investigate the correlations among attributes. Equal-width binning makes intervals of the same width (namely length) from the minimum value to the maximum value. For example, a range of values from 0 to 600 are split into three intervals: $[0, 200), [200, 400), [400, 600]$. Note that this approach is prone to produce unnecessary or nearly empty bins from sparse data or data with outliers. Equal-frequency binning ensures that each interval contains an equal (or close) number of values, given the number of intervals. For example, the sequence containing 12 values $\{3, 4, 5, 4, 6, 7, 7, 8, 9, 9, 10, 9\}$ is split into three intervals such as $[3, 6), [6, 9), [9, 10]$ ensuring that each interval contains four observations. K-means binning runs a k-means clustering algorithm on a single dimension of the data, which separates the data into k clusters while minimizing the mean squared errors within each cluster. Since each cluster maps to a non-overlapping interval, the clusters generated by this method provide a discretization scheme.

On real datasets, in addition to the three above mentioned benchmark methods, we compare our method with Mehta et al.'s CPD method [17], Nguyen et al.'s IPD method, [18], and Bay's MVD method [16]. Since we could not access the implementation codes of those methods, we compared our results with the ones reported in their studies.

Remember that, we define a good multivariate discretization to be one that each interval contains observations with similar properties and different intervals have observations with dissimilar properties with respect to other attributes. We showed that our method is generally more effective in this purpose as compared to the other methods.

[2] We used the implementations in *scikit-learn* library [20] used for machine learning in Python language.

Moreover, note that very large intervals can hide significant patterns in the data, whereas tiny intervals cannot have sufficient observations to capture patterns. This is called the *resolution* problem [16]. In our experiments, we show that the resolution problem occurs with three univariate benchmark methods. On the contrary, we can prevent resolution problem by avoiding paths with tiny intervals or paths with few large intervals by means of the weighted entropy formula (see Equation 2) and path optimization algorithm (discussed in Section 4.4.),

5.1. Evaluation Method

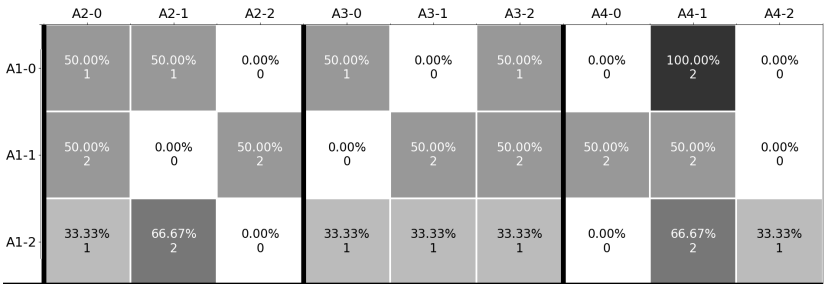
In order to evaluate our method's multivariate discretization performance and compare it to others, we need a numerical metric. Multivariate discretization enables putting observations with similar distributions into the same bins by considering all attributes and correlations within. To this end, we consider the entropy levels as an indicator of similarity among distributions since a low entropy in a group of observations implies that their distributions are not far off. Thus, once all attributes are discretized, we calculate the total entropy of each attribute as well as the total entropy of observations in each bin.

Before presenting our results, we first explain what a *percentage matrix* is, and then, we define our formula for calculating total entropies by taking an example of a small dataset. Observe our example dataset in Figure 5.1a where all attributes are assumed to be discretized into 3-bins and mapped to one of the labels $0, 1,$ or 2.

Percentage Matrix: A percentage matrix displays the frequencies of attribute levels with respect to other attributes' levels. An example percentage matrix and the corresponding attribute discretization is shown in Figure 5.1. In this matrix, each row and column represents a level of an attribute. The set of observations that has the level j in attribute A_i is denoted by $A_{i,j}$. A cell of the matrix is addressed by row $A_{i,j}$ and column $A_{k,l}$ and represent the observations whose A_i value is j and A_k value is l. Within each cell two statistics are given, namely the number of matching observations and their percentage within all of A_i. Moreover, the tone of the cell background also represents the percentage, the darker shades hinting at higher percentages.

A_1	A_2	A_3	A_4
0	0	2	1
0	1	0	1
1	2	1	0
1	0	1	1
1	0	2	0
1	2	2	1
2	0	1	1
2	1	2	1
2	1	0	2

(a)



(b)

Figure 5.1 A small discretized dataset (a) and its corresponding percentage matrix (b).

Note that, a good discretization is one that purifies other attributes with respect to the discretized attribute. This is equivalent to having higher contrasts in the matrix. The more cells we have with very high and very low percentages, the better. In this example, we have many 50% ratios, which means this is a weak discretization. On the other hand, cell $A_{1,0}, A_{4,1}$ has maximum percentage, and that means the first bin of A_1 contains only level-1 observations from attribute 4.

We evaluate the success of discretization with two entropy-based metrics:

1. *The total entropy of the observations in a discretization bin* $A_{i,j}$, calculated with the classical entropy formula and then summed up. It is denoted as $H_{i,j}$ in Equation 3 below. Note that M is the set of attributes in the dataset; $O_{i,j}$ is the set of observations where $A_i = j$; and $H_{k,i,j}$ is the total entropy of observation set $O_{i,j}$ with respect to k-th attribute (only values in A_k used for entropy calculation) based on Equation 1.

$$H_{i,j} = \sum_{k \in M-i} H_{k,i,j} \tag{3}$$

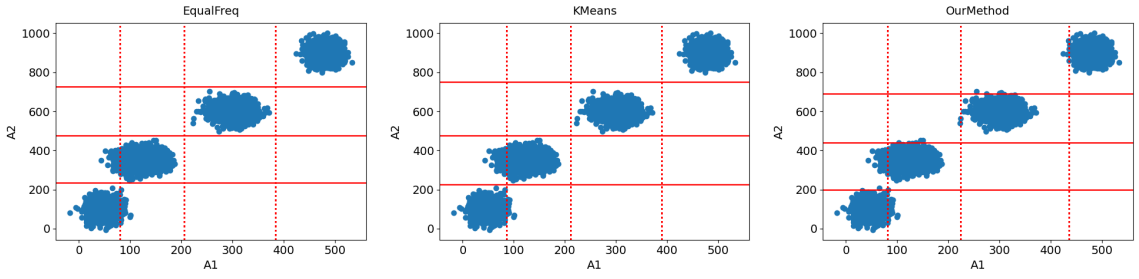
2. *The total entropy of attribute* A_i, which is the weighted sum of each bin's *total entropy* ($H_{i,j}$). Equation 4 provides a formal equation. Note that N is the total number of observations in the dataset and J is the total number of bins in attribute A_i. Note that, the weight coefficient allows us to decrease the contribution of pure but small-sized bins in total entropy.

$$H_i = \sum_{j=1}^{J} \frac{|O_{i,j}|}{N} \times H_{i,j} \tag{4}$$

5.2. Experiments on Synthetic Data

We adopted the approach by Nguyen et al. [18] and Bay [16] and performed our experiments on synthetic data with designed correlations among attributes. Our method excels at discovering these correlations between attributes and uses them for better discretization. Considering that we work on unsupervised data, exposing the multi-modality of the data is the best available option and the synthetic datasets provide an optimal ground for testing this. Next, we explain how we generate the datasets.

5.2.1. Synthetic Data Generation



(a) Equal-frequency (b) K-Means (c) Our method

Figure 5.2 Discretizations on linearly correlated synthetic data by different methods.

For explaining the rationale behind our data generation, we provide a very simple example in Figure 5.2 where we consider only two attributes with a very noticeable linear dependency. In this figure, the discretization intervals computed by equal-frequency and k-means algorithms and our method is shown. Even though equal-frequency and k-means, both of which are univariate methods, do not consider the correlations between the attributes, they still perform fine due to the inherent shape of the data. Our algorithm is designed to handle complicated correlations which can not be exposed trivially by univariate approaches. Therefore, in order

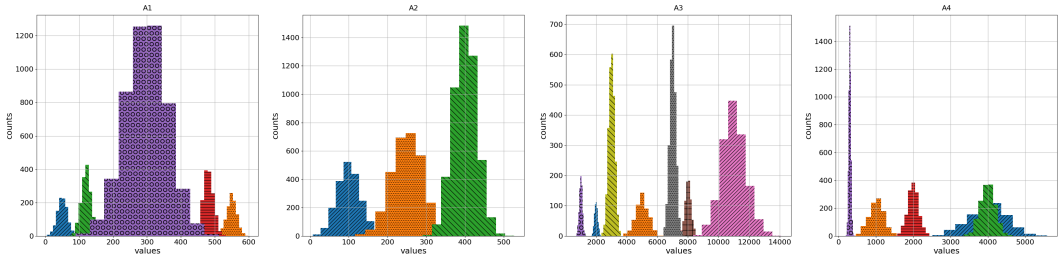
Table 5.1 Configuration of synthetic dataset.

A_1	A_2	A_3	A_4
$X_{1,1} \sim N(50, 15^2)$ size=1000	$X_{2,1} \sim N(100, 30^2)$ size=2000	$X_{3,1} \sim N(2000, 100^2)$ size=500	$X_{4,1} \sim N(4000, 500^2)$ size=1000
		$X_{3,2} \sim N(5000, 400^2)$ size=500	
$X_{1,2} \sim N(550, 15^2)$ size=1000		$X_{3,3} \sim N(1000, 100^2)$ size=500	$X_{4,2} \sim N(1000, 200^2)$ size=1000
		$X_{3,4} \sim N(8000, 200^2)$ size=500	
$X_{1,3} \sim N(120, 15^2)$ size=1500	$X_{2,2} \sim N(250, 40^2)$ size=3000	$X_{3,5} \sim N(1000, 100^2)$ size=750	$X_{4,3} \sim N(4000, 200^2)$ size=1500
		$X_{3,6} \sim N(8000, 200^2)$ size=750	
$X_{1,4} \sim N(480, 15^2)$ size=1500		$X_{3,7} \sim N(11000, 800^2)$ size=1500	$X_{4,4} \sim N(2000, 150^2)$ size=1500
$X_{1,5} \sim N(300, 60^2)$ size=5000	$X_{2,3} \sim N(400, 30^2)$ size=5000	$X_{3,8} \sim N(7000, 200^2)$ size=2500	$X_{4,5} \sim N(300, 30^2)$ size=5000
		$X_{3,9} \sim N(3000, 200^2)$ size=2500	

to test our method's capability, we generated our synthetic data with more noise and non-trivial correlations.

We first generate a dataset with 10000 observations, comprising varying correlations among four attributes, namely A_1, A_2, A_3, and A_4. Each attribute is then sequentially divided into intervals of differing sizes. Each interval is assigned a separate random variable each of which produces values using Gaussian distributions with varying means and standard deviations: $X_{i,j} \sim N(\mu_{i,j}, \sigma^2_{i,j})$. Here, i represents the attribute and j represents the order of the random variable within the attribute.

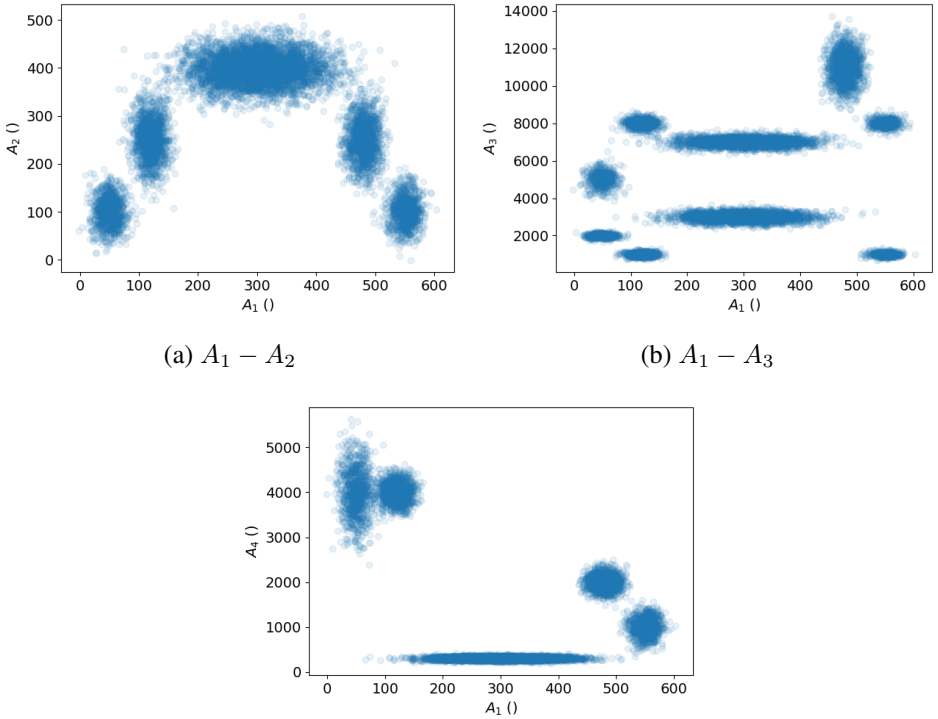
Our synthetic data configuration is given in Table 5.1 and also visualized with histograms in Figure 5.3. For example, $X_{2,1}$ is assigned to observations 1 through 2000, $X_{2,2}$ to observations 2001 through 5000, and $X_{2,3}$ to 5001 through 10000. Also, first 1000 observations are assigned to $X_{1,1}$ and the next 1000 to $X_{1,2}$. That means, two observations both of which are associated to the same random variable within the second attribute may be associated to two different random variables within the first attribute. This introduces a non-linear correlation



(a) A_1 (b) A_2 (c) A_3 (d) A_4

Figure 5.3 Distributions of attributes in the synthetic dataset.

between attributes. However, we assign the random variables to the observations in a sequential manner, and hence although we create variation, we do not produce totally random observations. The resulting dataset and the correlations among attributes can be observed in Figure 5.4.



(a) $A_1 - A_2$ (b) $A_1 - A_3$

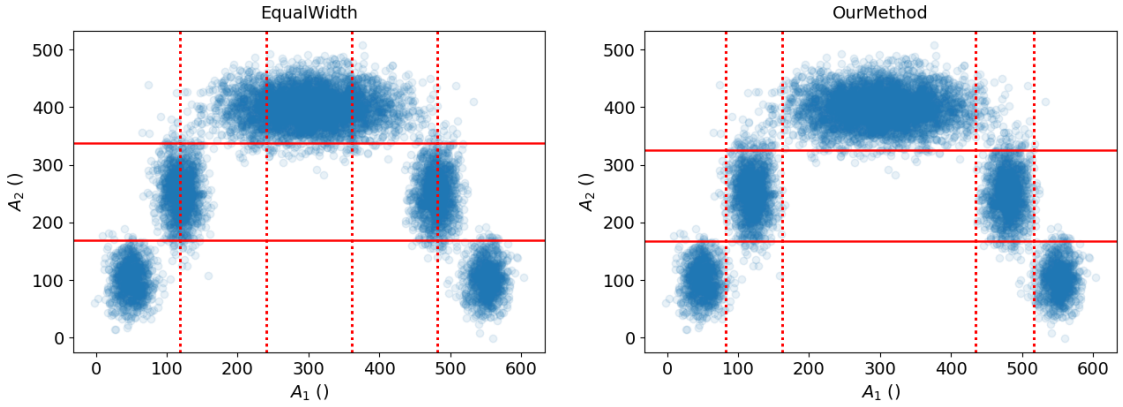
(c) $A_1 - A_4$

Figure 5.4 Correlations between A_1 and other attributes in the synthetic dataset.

5.2.2. Results

We run our algorithm and three benchmark methods on our synthetic dataset to discretize A_1, A_2, A_3, A_4 into 5, 3, 6, 4 bins, respectively. These bin sizes are optimal by the construction of the dataset.

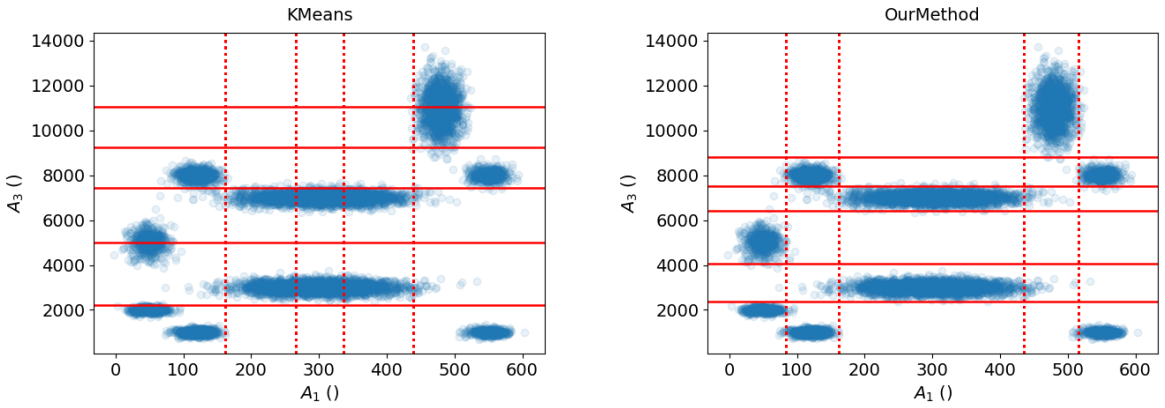
We provide comparisons of our method with the three benchmark methods in Figures 5.5,5.6, and 5.7. Considering that presenting all comparisons visually would take a very large amount of space, here we provide visual comparisons for a limited amount of cases. More detailed comparisons are given numerically in Section 5.2.3..



(a) Equal-width (b) Our method

Figure 5.5 A_1 and A_2's discretization intervals produced by equal-width and our method.

Figure 5.5a demonstrates that equal-width approach could not detect similar clusters of observations since it did not regard A_2 for determining cutpoints of A_1 and vice versa. It only focused on partitioning a single dimension into equal widths. However, our method took all other attributes into account and discovered known ground-truth cutpoints on A_1 and A_2 and separated the clusters of observations properly (as shown in Figure 5.5b).

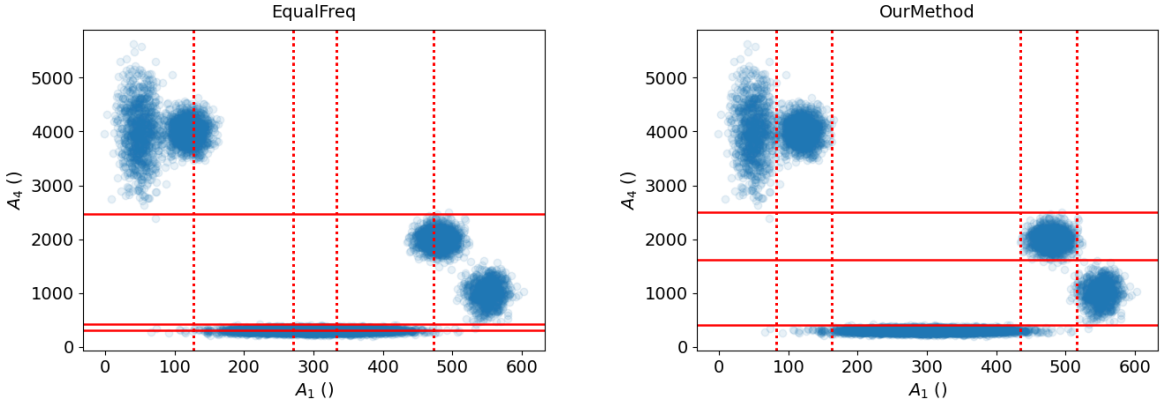


(a) K-Means

(b) Our method

Figure 5.6 A_1 and A_3's discretization intervals produced by k-means and our method.

In Figure 5.6a, k-means binning approach split A_1 into five clusters and minimized the mean square error within clusters using only the values in A_1. Similarly, it split A_3 into six clusters using only the values of A_3. Consequently, the correlations between the discretized attribute and the others are disregarded. On the other hand, our method regards all correlations between attributes while calculating entropy of bins and tries to minimize the entropy within each bin. Therefore, in Figure 5.6b, we see a better separation of clusters and cutpoints closer to the known ground-truth cutpoints.



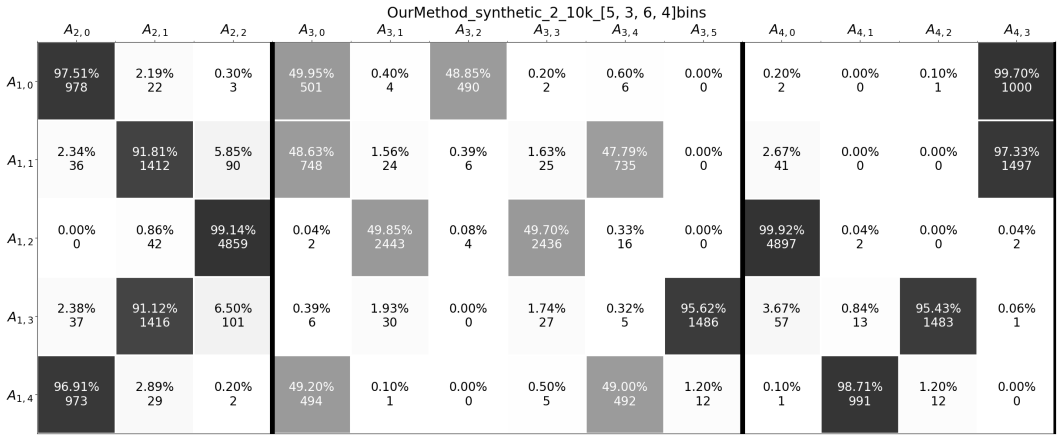
(a) Equal-frequency

(b) Our method

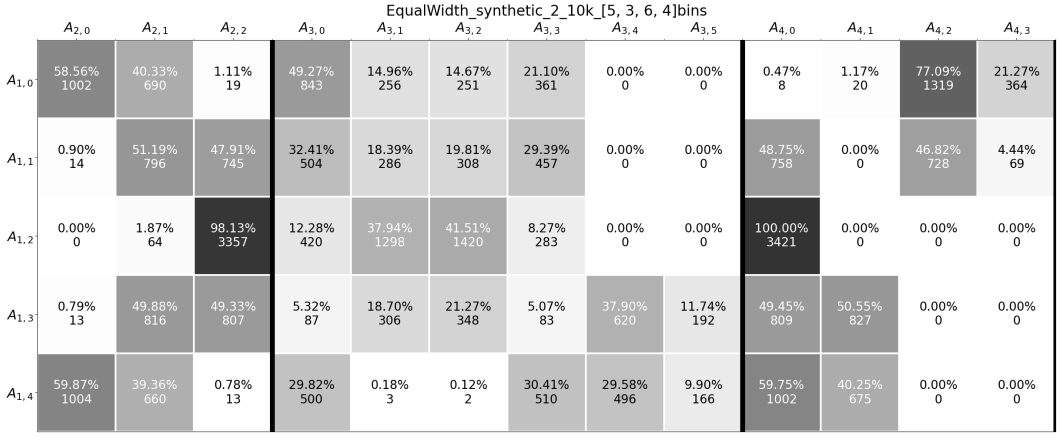
Figure 5.7 A_1 and A_4's discretization intervals produced by equal-frequency and our method.

For attribute A_1 and A_4, equal-frequency binning produced intervals each of which contains a nearly equal number of observations as shown in Figure 5.7a. When those intervals are projected into a scatter plot for $A_1 - A_4$, we see that they are not as successful as that of our method at separating clusters, as demonstrated in Figure 5.7b.

5.2.3. Evaluation

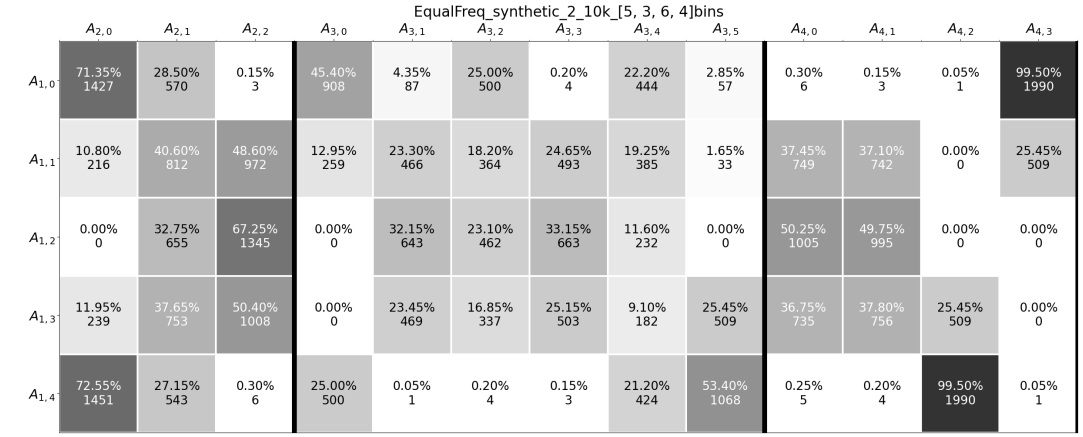


(a) Our method

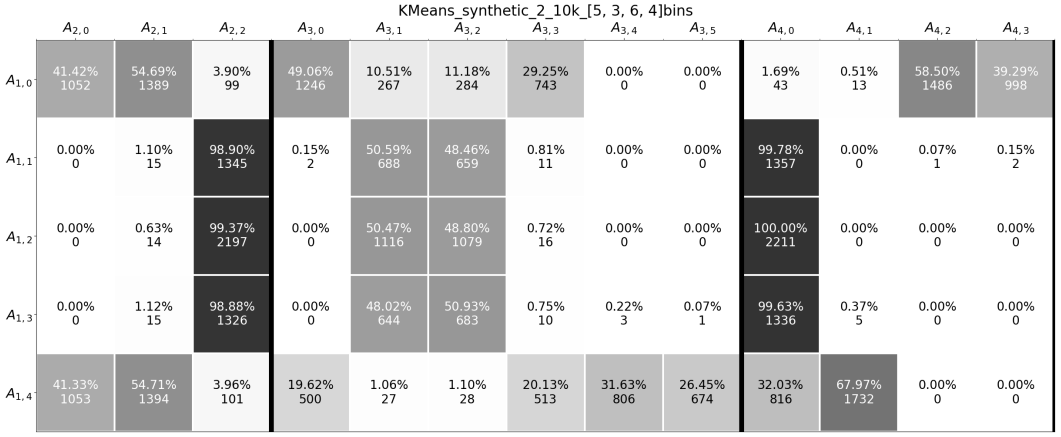


(b) Equal-width

Figure 5.8 Percentage matrices of A_1 after our method and equal width applied.



(a) Equal-frequency



(b) K-Means

Figure 5.9 Percentage matrices of A_1 after equal frequency and k-means binning applied.

In Figure 5.8 and 5.9, the frequency matrices of attribute A_1 after the synthetic dataset was discretized with different methods are shown. This visualization helps us to assess the entropy of bins after discretization. When we examine the percentage matrix of our method (see Figure 5.8a), we see more contrast as compared to the matrices of equal-width (see Figure 5.8b) or equal-frequency (see Figure 5.9a). The k-means algorithm did better than the other two, but failed to identify the bins as clear as our approach. The rest of the percentage matrices for all attributes and methods are listed in Appendix.

Figure 5.10 shows the $H_{i,j}$ values for all attributes and all methods. In this figure, the width of each bar represents the ratio of observations falling into that bin and the height represents

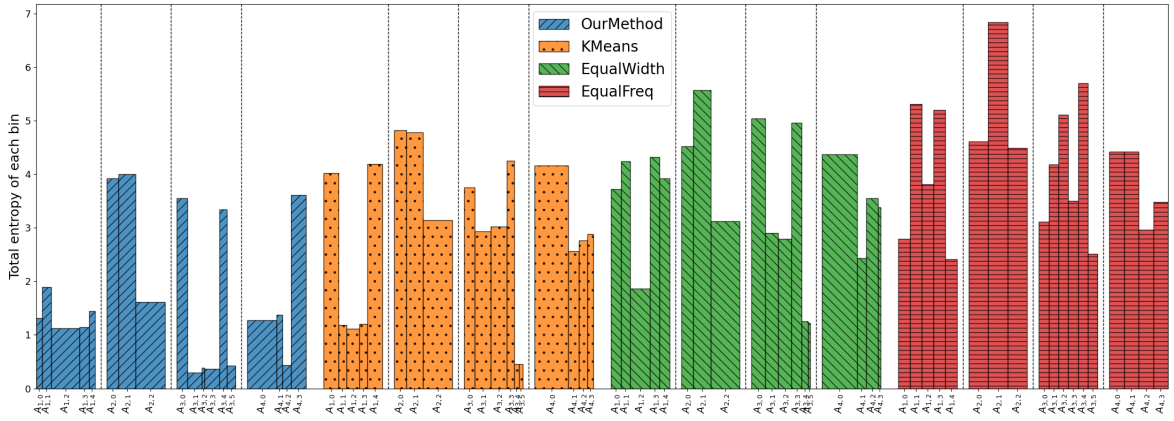


Figure 5.10 Total entropy achieved in each bin via all four methods.

the bin-wise entropy value, $H_{i,j}$. A better discretization should provide thick and short bars, whereas the worst is to have thick and long bars. Overall, our approach provides thicker and shorter bars (larger intervals with smaller entropies). Finally, Figure 5.11 provides a visualization of the attribute-wise entropy values, H_i, for each method. It clearly shows that our approach is superior to all three benchmark algorithms on the synthetic data set.

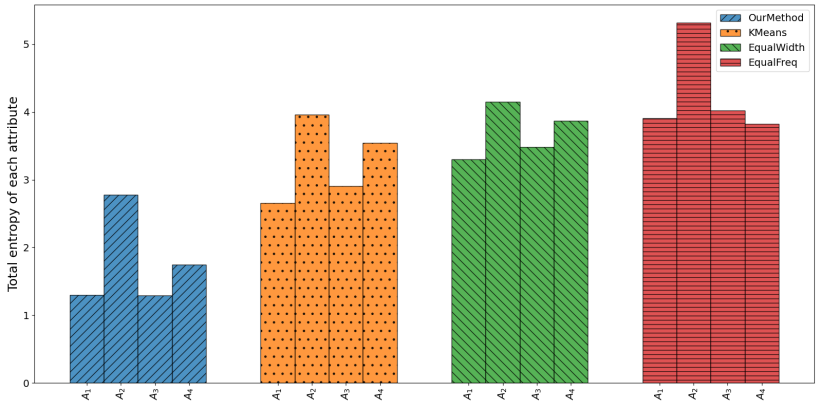


Figure 5.11 Total entropy achieved in each attribute via all four methods.

5.3. Experiments on Real Data

We performed our experiments on four common real datasets from UCI Machine Learning Repository [21] (summarized in Table 5.2). We compare our method against three benchmark methods: equal-width, equal-frequency, and k-means binning. Also, we provide comparisons of our method with other multivariate discretizers such as Mehta et al.'s CPD method

Dataset	Attributes	Observations	Class Labels
Adult-income	14	48842	2
SatImage	36	6435	6
Shuttle	9	58000	2
Ionosphere	34	351	2

Table 5.2 Used real datasets and their properties.

[17], Nguyen et al.'s IPD method, [18], and Bay's MVD method [16] whenever we have data of their results. Since we could not access the implementation of these methods, we compared our results with results reported in the respective papers. For comparison, we consider two evaluation metrics: percentage matrices and total entropy levels, both of which were explained in the previous section.

In addition, we perform classification tests with the random forest classifier to measure how different discretizations affect the performance of a machine learning study. Although random forest is not always the best classifier for prediction, it is enough to show that our method preserves correlations in a predictive task. We also remind that our method's main purpose is to preserve correlations between attributes and to find hidden patterns in order to minimize the accuracy loss in machine learning. Supervised entropy-based discretization methods are reported to be improving the predictive performance of some classifiers such as Naive-Bayes [5]. Nevertheless, Bay argues that supervised discretizers such as ME-MDL harm pattern discovery in exploratory tasks despite the increase in prediction accuracy [16]. For example, Bay reports that using different class variables of UCI Student Admissions dataset radically changed cut points of income attribute although the predictive accuracy was stable in two cases. Hence, supervised discretization may affect the stability of discretization results and may not give confidence to a human expert for discovering patterns.

On the other hand, unsupervised discretization methods do not consider class labels and as a result, the correlations between attributes and the class labels may be lost. Therefore, a small decrease in prediction scores after an unsupervised discretization is not unusual. Our results show that unsupervised benchmark methods resulted in considerable decrease in prediction

accuracy. Nonetheless, our method surpassed two benchmark methods, and also performed very close to undiscretized data.

5.3.1. Experiments on Adult-income Dataset

This dataset is collected from the 1994 Current Population Survey in the US. It has 14 attributes such as age, education level, capital gain and loss, gender, weekly worked hours, marital status, ethnicity, occupation, etc. It has two class labels since it is prepared for a task to predict whether a person's income is above 50,000 dollars or not. We omit class labels as well as nominal attributes like marital status. Thus, we use only five attributes: age, weekly worked hours, educational level (from preschool to PhD), capital gain, and capital loss.

We present the results of the experiment with three benchmark methods. Our bounds for the number of bins are five (maximum) and three (minimum). Our method separated capital loss

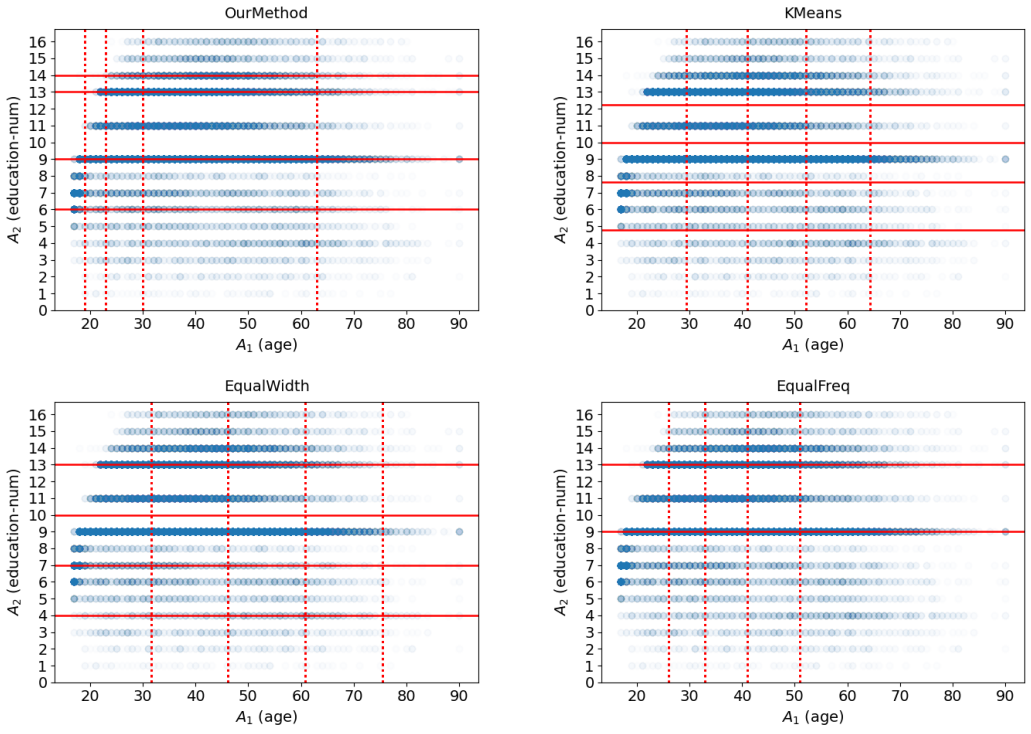


Figure 5.12 Discretization intervals for age and educational level attributes, produced by our method and benchmark methods in *adult income* dataset.

and weekly working hours into four bins, and the remaining attributes into five bins. Thus, we run benchmark methods with that number of bins for a fair comparison.

In Figure 5.12 we observe that our method determined the cutpoints for the age as $[19, 23, 30, 63]$. Note that our intervals for age focused on young ages rather than old ages as opposed to other benchmark methods. Remember that our method discretizes the age attribute considering its relation with educational level, capital gain and loss, and weekly working hours at the same time. Since this dataset is based on demographic and economical properties, it is meaningful to have more detailed intervals at young ages as they contain groups with different economic qualities [16]. Also, most people retire at age 60, thus, detailed intervals at old ages do not make much sense. This allows us to capture more patterns in terms of people's educational and economical qualities.

For example, we can see in Figure 5.12 that until age 19, people have an education level number 9 at most. It corresponds to a high school degree (using the provided education level map in Table 5.3). This pattern can be explored with our method's discretization as it has a cut point at age 19. However, such a pattern between age and education is lost in other methods' intervals as their cut points for age start from the late twenties. This is another example of the resolution problem with too large intervals.

Enumeration	Educational level
1	Preschool
2	$1^{st} - 4^{th}$ class
3	$5^{th} - 6^{th}$ class
4	$7^{th} - 8^{th}$ class
5	9^{th} class
6	10^{th} class
7	11^{th} class
8	12^{th} class
9	High school & some college degree
11	Associate degree
13	Bachelors
14	Masters
15	Professional school
16	Doctorate

Table 5.3 Educational level and their enumerations in *adult income* dataset.

Next, the intervals for educational level (which can be seen in Figure 5.12) are presented in Table 5.4 for easy interpretation. Our method determined cut points for educational levels as 9^{th} class, high school degree, associate degree, bachelor's degree, and master's degree. We consider this a reasonable decision since usually each refers to a different career path. However, all other benchmark methods grouped people with bachelor, master, and PhD. degrees into the same interval. It probably hides many patterns in population with different academic degrees.

Method	Education level intervals	Education level groups (inclusive)
Our method	$[1, 6), [6, 9), [9, 13), [13, 14), [14, 16]$	[preschool-9^{th} class], [1^{th} class-1^{th} class], [high school degree-associate degree], [bachelor's degree], [masters degree, PhD degree]
K-Means	$[1, 5), [5, 8), [8, 11), [11, 13), [13, 16]$	[preschool-8^{th} class], [9^{th} class-11^{th} class], [12^{th} class-high school degree], [associate degree], [bachelor's degree, PhD degree]
Equal Width	$[1, 4), [4, 7), [7, 11), [11, 13), [13, 16]$	[preschool-6^{th} class], [7^{th} class-10^{th} class], [11^{th} class-high school degree], [associate degree], [bachelor's degree, PhD degree]
Equal Frequency	$[1, 9), [9, 13), [13, 16]$	[preschool-12^{th} class], [high school degree-associate degree], [bachelor's degree, PhD degree]

Table 5.4 Educational level intervals produced by different discretization methods.

In Figure 5.13 we can see the discretization of age and weekly work hours together. Our method determined the cut points as $[9, 31, 42]$. Also, Figure 5.18 shows percentage matrices of the *hours per week* attribute (A_5) produced by our method, equal-width and k-means binning. At first look, we can see higher contrast in the cells of our method's percentage matrix. It is an indication that our intervals are purer with respect to other attributes such as capital gain and loss, age, etc.

Notice that the observations of people working under 9 hours per week are mostly under age 20 and above age 60 (see Figure 5.13). It might correspond to the young population who work part-time and the elder population with fewer working hours. We consider this as a significant correlation between age and work hours. Nevertheless, the other three benchmark methods could not capture that correlation since their first cut point for age is around 30.

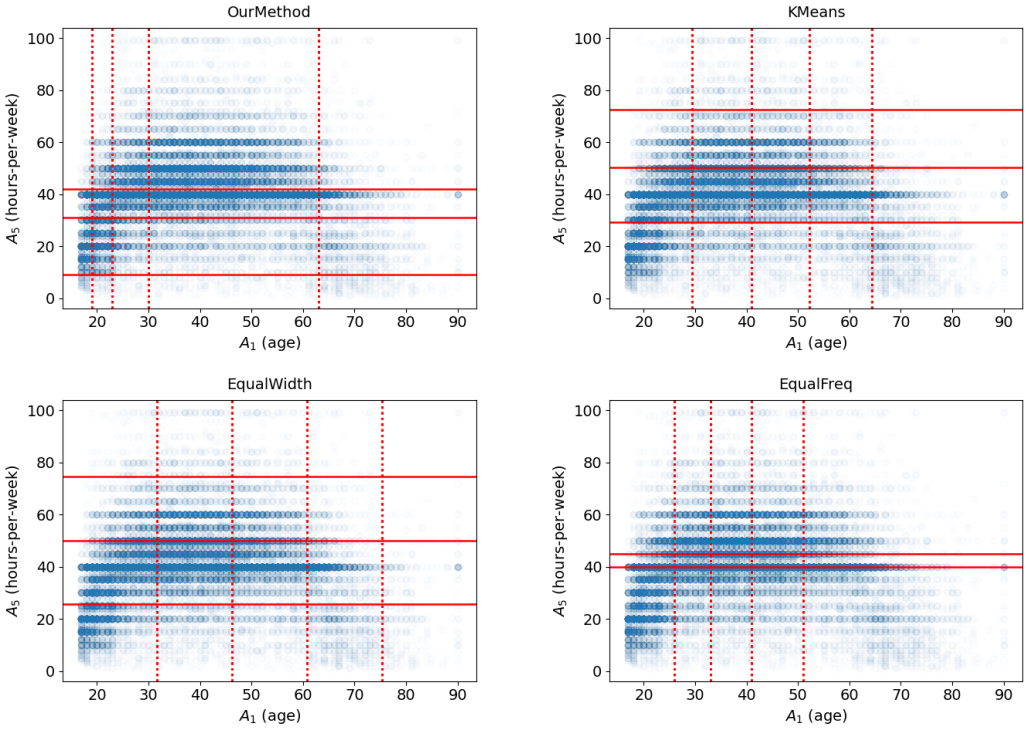


Figure 5.13 Discretization intervals for age and weekly working hours attributes, produced by our method and benchmark methods in *adult income* dataset.

This is an example of a resolution problem, that is, too large intervals for age attribute hid the mentioned pattern.

Moreover, our method placed the last cut point at 42 hours per week, which is generally accepted as the overworking threshold in most countries. Also, the cut point at 42 hours (denoted as $A_{5,3}$) provided a high contrast in the percentage matrix of our method (see Figure 5.18). On the other hand, k-means and equal-width chose the last cut point to be around 80 hours. We cannot contextually make a meaningful explanation of this cut point. In addition, the cut point at 80 (corresponds to $A_{5,4}$) does not have cells with high contrast in the percentage matrix of k-means in Figure 5.18.

We list all percentage matrices of all attributes below in Figure 5.14, Figure 5.15, Figure 5.16, Figure 5.17, Figure 5.18. Since equal frequency could not discretize capital gain and loss attributes, percentage matrices for equal frequency are not possible. However, we can compare our method's discretization to other two benchmark methods (equal width and k-means).

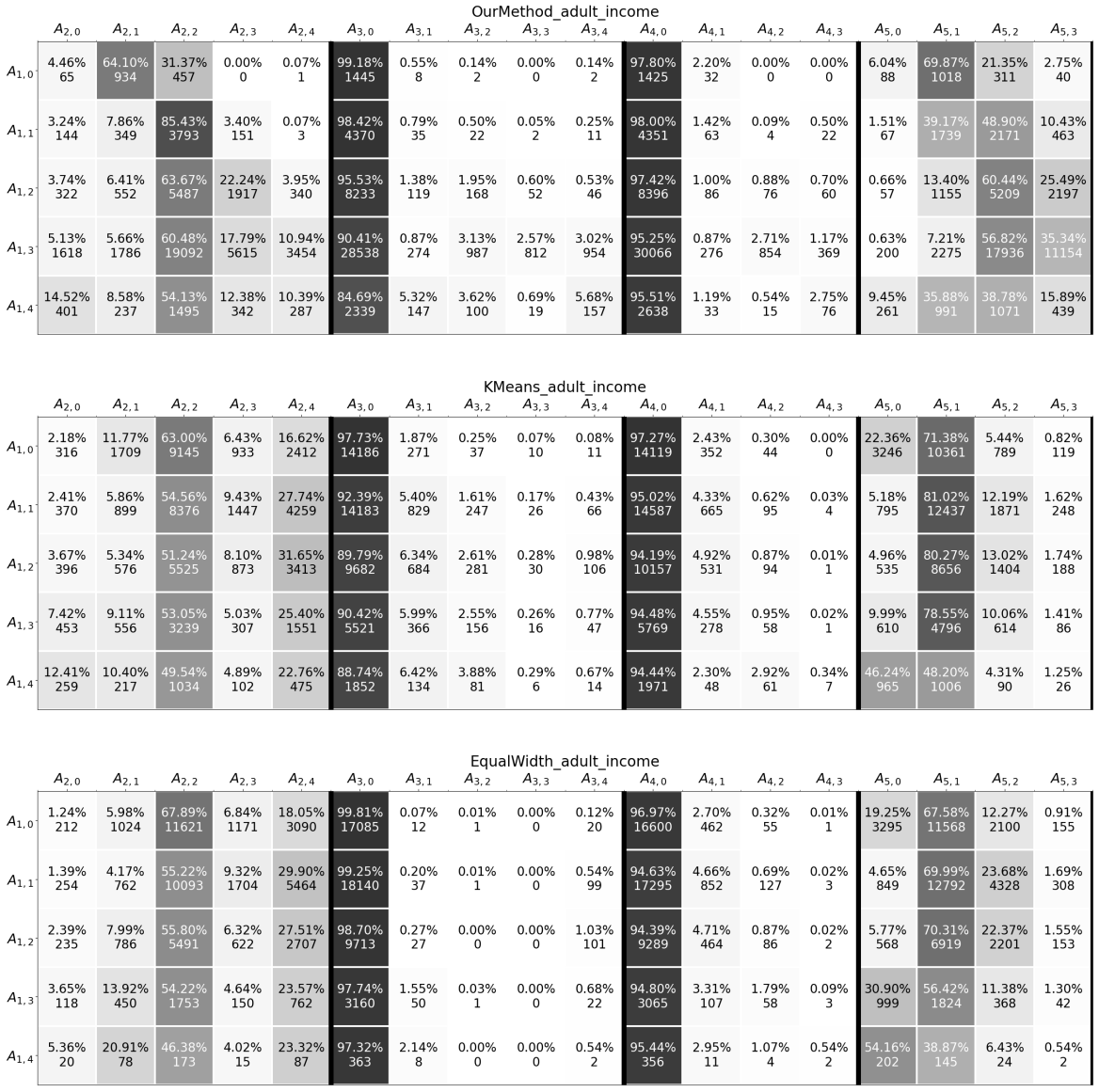


Figure 5.14 Percentage matrices of the *age* attribute (A_1) produced by our method and benchmark methods.

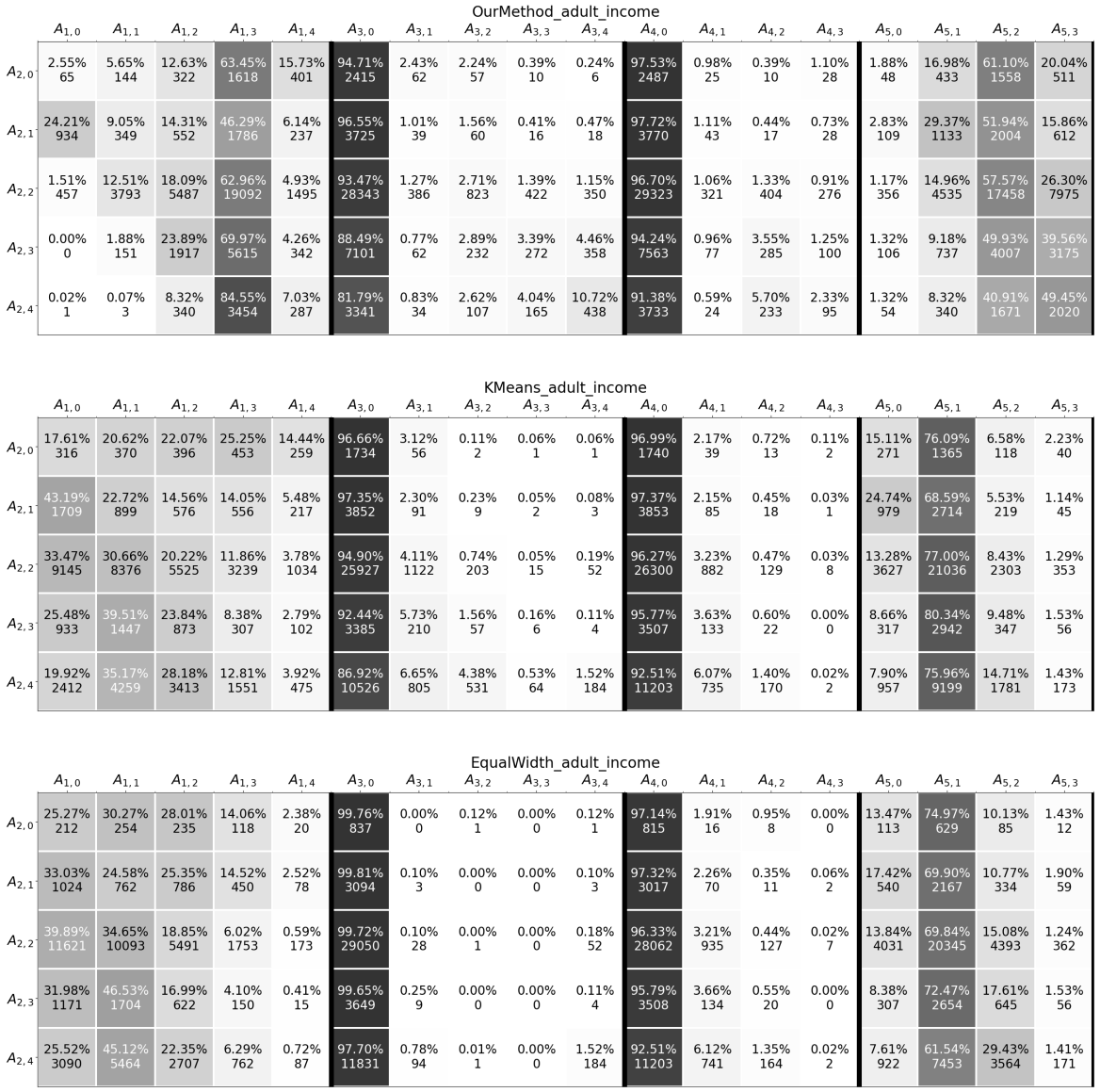


Figure 5.15 Percentage matrices of the *education level* attribute (A_2) produced by our method and benchmark methods.

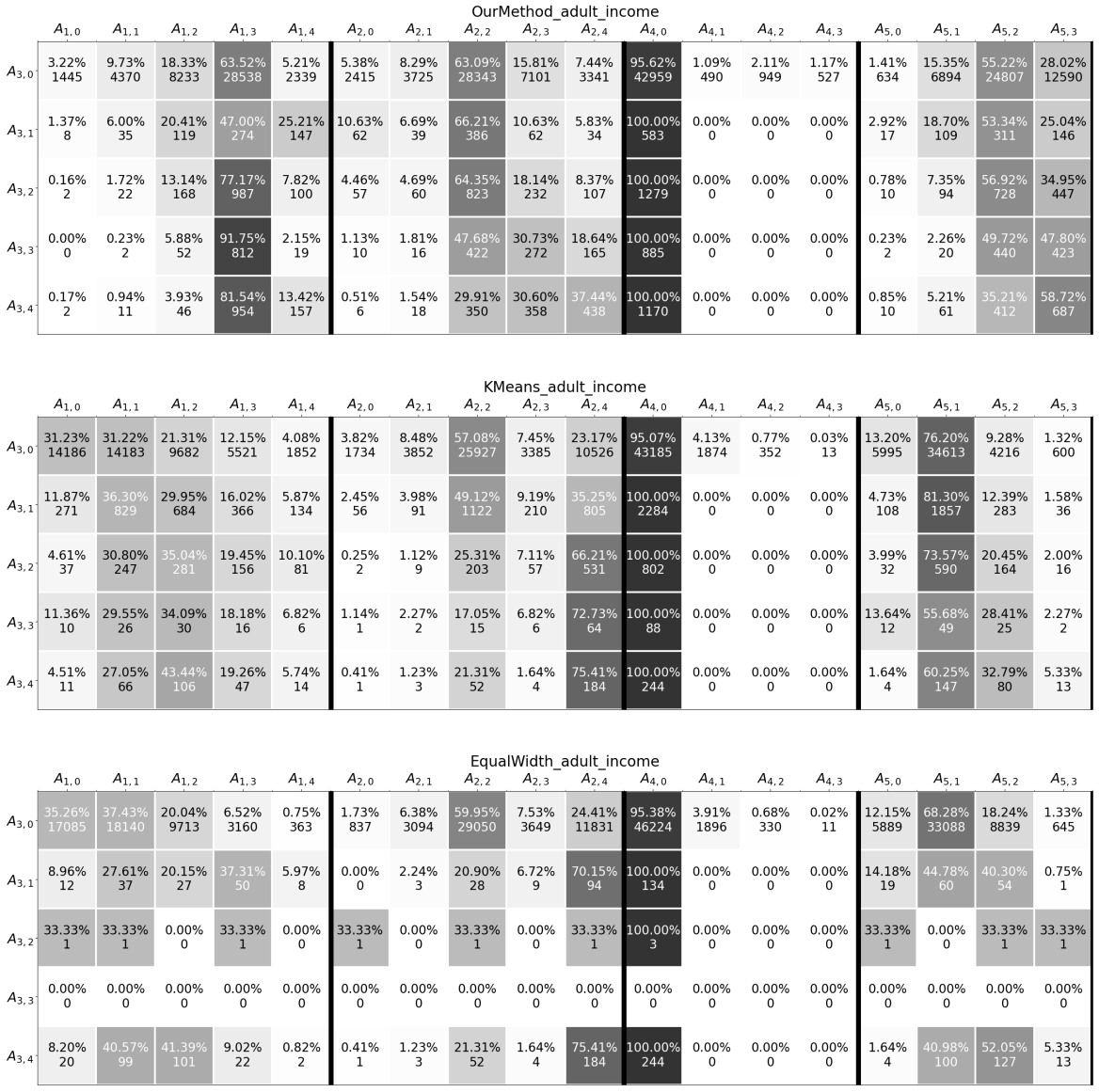


Figure 5.16 Percentage matrices of the *hours per week* attribute (A_3) produced by our method and benchmark methods.

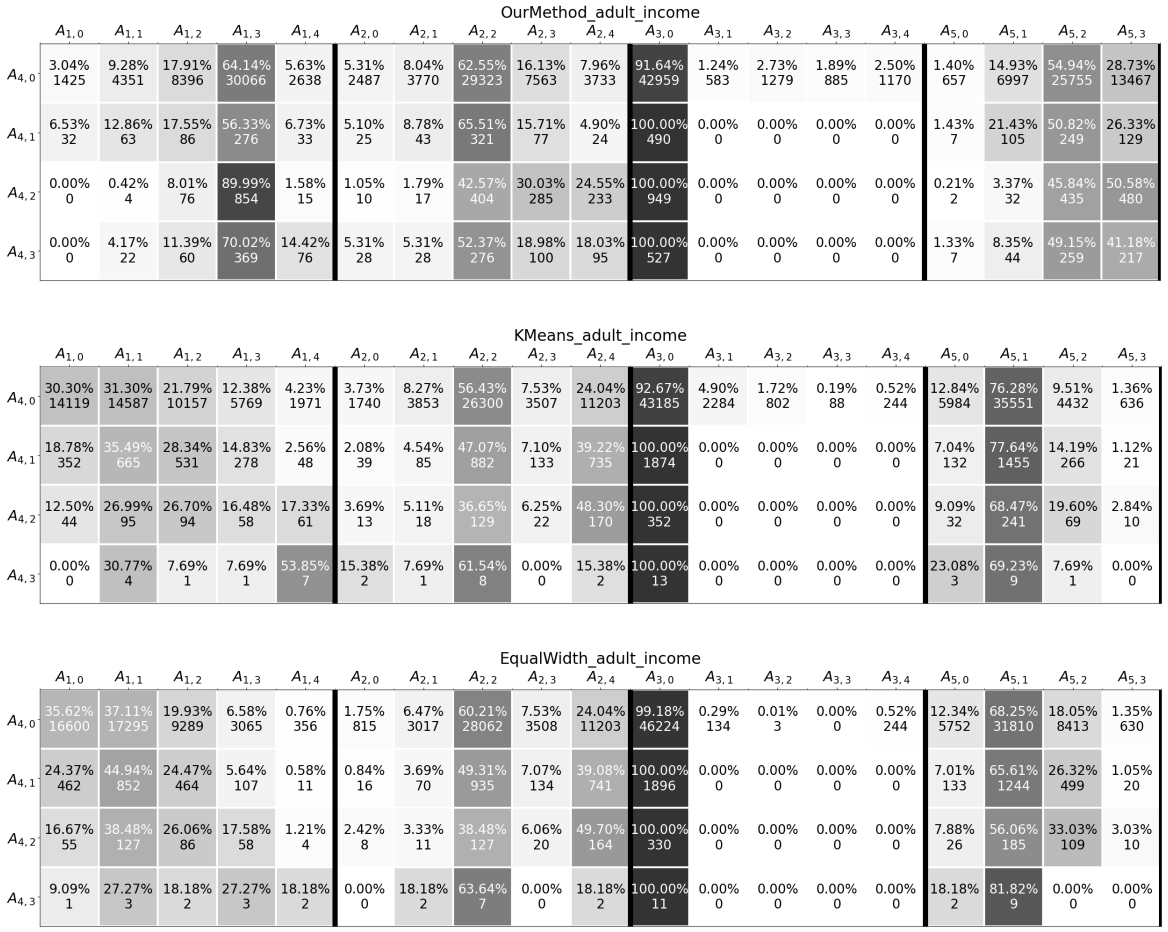


Figure 5.17 Percentage matrices of the *capital loss* attribute (A_4) produced by our method and benchmark methods.

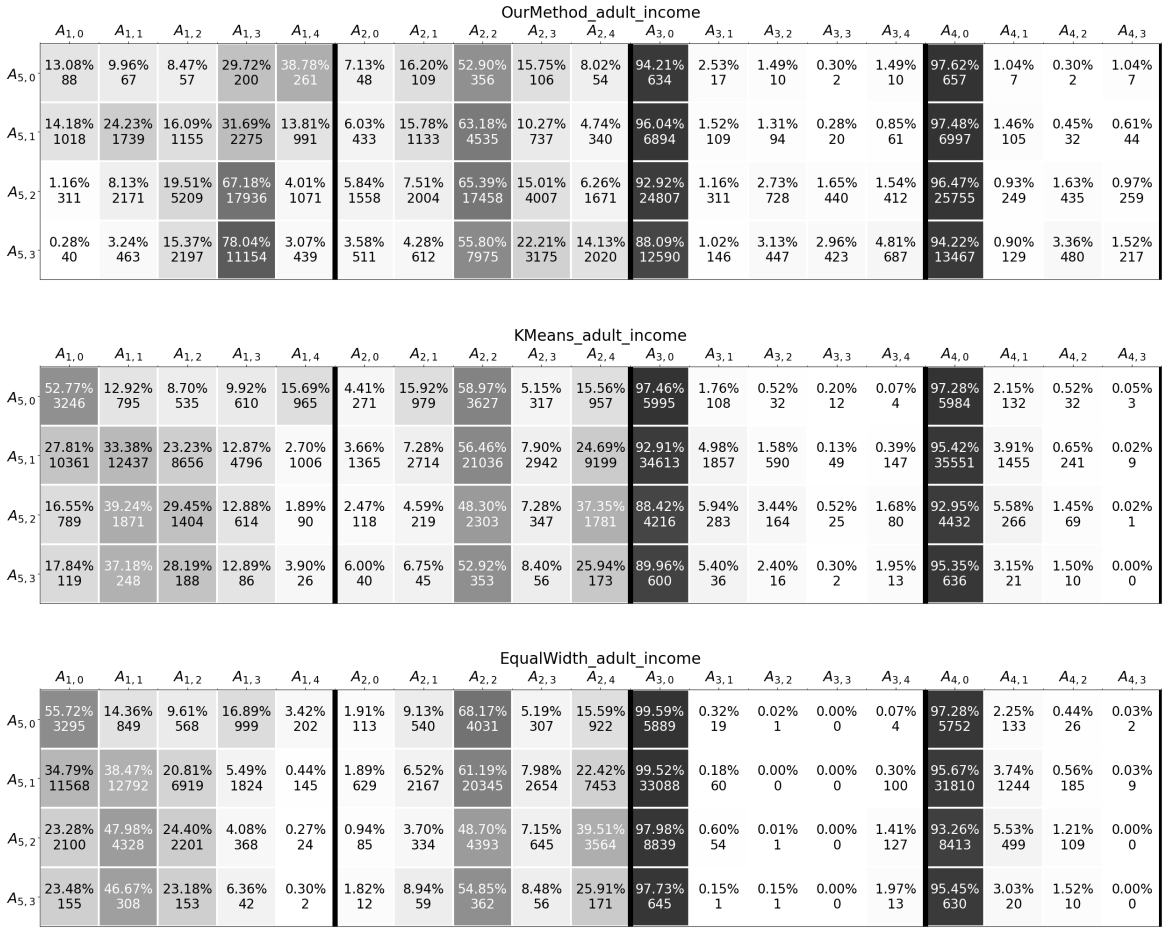


Figure 5.18 Percentage matrices of the *hours per week* attribute (A_5) produced by our method and benchmark methods.

After discussing the cut points in detail, we show the total entropy levels provided by our method and each benchmark method in Figure 5.19. Our method achieved a comparable total entropy level for each attribute. We note that the equal-frequency method could not discretize capital gain and loss attributes into equal-frequency bins since they are highly skewed.

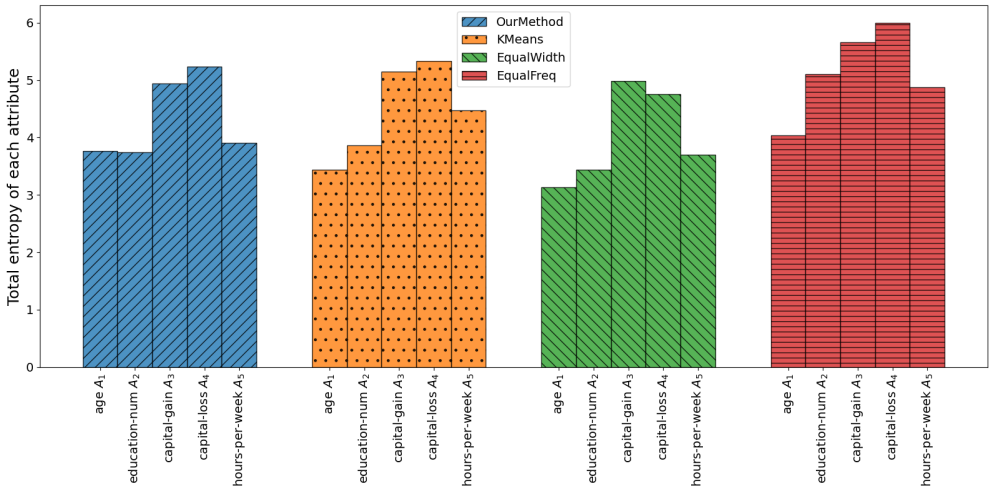


Figure 5.19 Total entropy achieved in each attribute in *adult income* dataset.

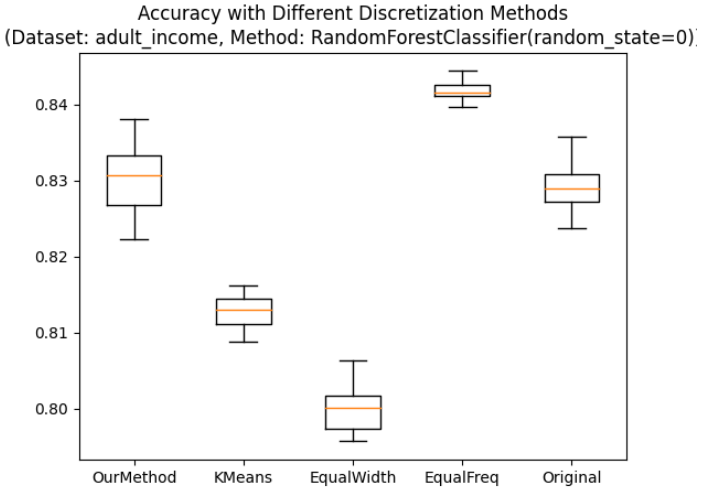


Figure 5.20 Classification accuracy of our method and three benchmark methods in *adult income* dataset.

Finally, we performed a classification experiment using the discretization results. We discard the class labels during discretization, and later we merge them back into the dataset before

model training. All experiments are performed using random forest implementation of scikit-learn library [20] with default parameters and 10-fold cross validation. Since the benchmark methods are univariate and unsupervised, equal width and k-means caused a decrease in prediction accuracy as expected (see Figure 5.20). Note that, the adult-income dataset is highly skewed on gain and loss attributes and the equal frequency binning could not discretize those attributes. As a result, random forest algorithm performed better with equal frequency than other two univariate methods. Our method shows comparable results with a classification performed on undiscretized data.

To conclude, our method produced meaningful and insightful intervals for exploratory tasks, it preserved complex correlations between attributes, and produced lower total entropy values for most attributes. Moreover, the classification experiment showed that the prediction accuracy was nearly the same as undiscretized data, which is remarkable for an unsupervised discretization.

Similar to our study, Mehta et al. [17] used adult income dataset in their experiments for evaluating their CPD method. We do not have access to CPD implementation, nevertheless, we take the discretization intervals of each attribute reported in that study. Then, we calculate percentage matrices and total entropy levels using the cut points produced by CPD and our method. Since the number of intervals produced by CPD is different than our previous experiment with benchmark methods, we conducted a separate experiment. The increase in number of bins result in higher levels of entropy, therefore, we use the same number of bins to prevent unfair comparisons in terms of entropy levels.

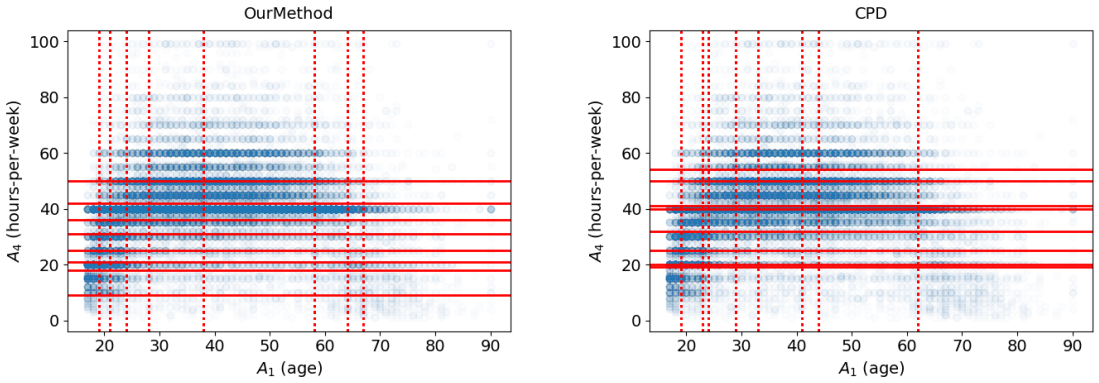


Figure 5.21 Discretization intervals for age and weekly working hours attributes, produced by our method and CPD methods in *adult income* dataset.

In Figure 5.21 we can observe the discretization intervals of CPD and our method for age and weekly working hours. Since CPD and our method are multivariate discretizers, both produced more detailed intervals on younger ages as compared to univariate benchmark methods. This is because younger ages include more diverse socioeconomic groups and more details to be identified. It can be easily observed from the figure that our method provided thicker intervals whereas some CPD intervals are contextually meaningless due to being very thin.

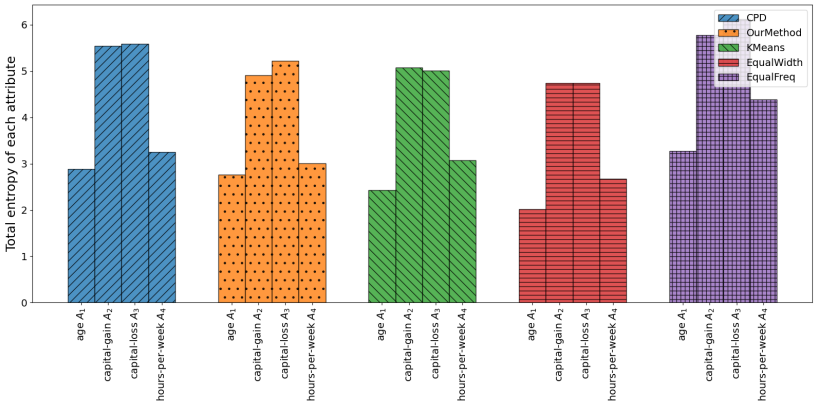


Figure 5.22 Total entropy achieved in each attribute in *adult income* dataset (including CPD).

Total entropy values of each attribute after discretization with respect to CPD's interval counts are presented in Figure 5.22. Both methods could not lower the total entropy of each attribute when compared to univariate benchmark methods. Finally, Figure 5.23 presents

the classification accuracy scores of all methods including CPD. As we stated earlier, unsupervised methods tend to decrease predictive performance in classification. Equal width and k-means caused a decrease in accuracy. Equal frequency showed a better accuracy since it could not discretize capital gain and loss attributes, and undiscretized data can result in higher accuracies. However, CPD and our method performed nearest to the classification on undiscretized data. Also, we can see that our method is slightly better than CPD's accuracy.

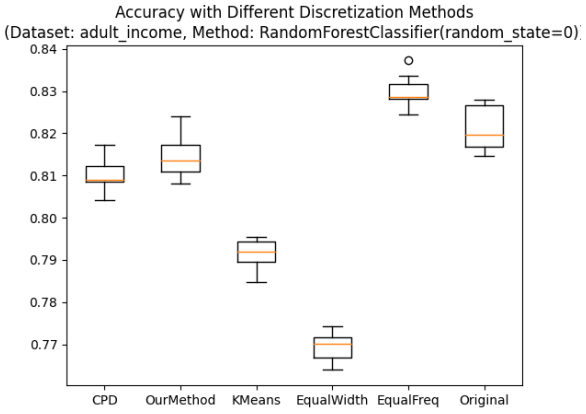


Figure 5.23 Classification accuracy of our method, CPD and three benchmark methods in *adult income* dataset.

5.3.2. Experiments on SatImage Dataset

This dataset is retrieved from Landsat Multi-spectral Scanner images of NASA. It consists of 3x3 pixel values of four spectral bands, which totals 36 attributes. It is used for a classification task to predict the soil type among 6 soil class labels.

We present the results of the experiment with three benchmark methods. Our bounds for the number of bins are five (maximum) and three (minimum). Our method separated all attributes into five bins. Thus, we run benchmark methods with five bins for a fair comparison.

Total entropies of each attribute after discretization are presented in Figure 5.24. Our method could not lower the total entropy of each attribute significantly when compared to univariate benchmark methods.

Next, Figure 5.25 presents the classification accuracy scores of all methods. Again, all experiments used random forest with default parameters and 10-fold cross-validation. Our method's score is 0.86 in the lower quartile (Q2) and 0.90 in the upper quartile (Q3). It is apparent that our method's score is the nearest to the score of classification on undiscretized data. Also, Nguyen et al. [18] performed classification experiments with random forests on the SatImage dataset and they reported that their IPD method has scored 0.89 ± 0.01 whereas Bay's [16] MVD method scored 0.81 ± 0.01 in terms of prediction accuracy. We are aware that their and our configuration for random forest might not be the same, nonetheless, it is sufficient to say that our method is comparable with IPD and MVD in terms of classification scores.

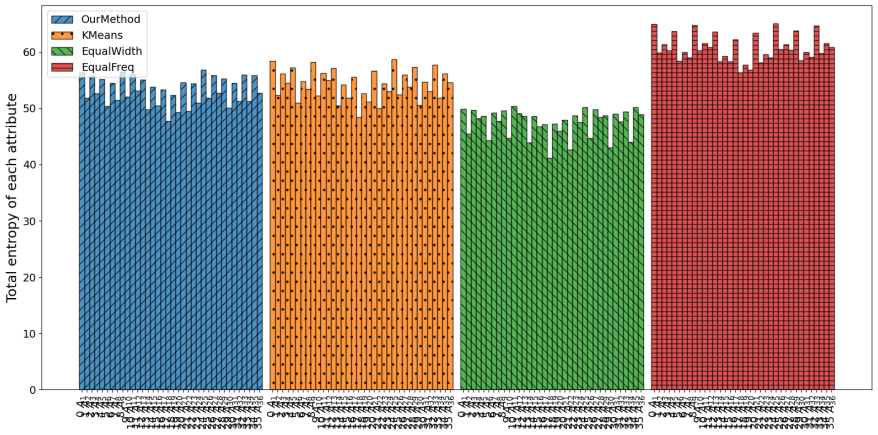


Figure 5.24 Total entropy achieved in each attribute in *SatImage* dataset.

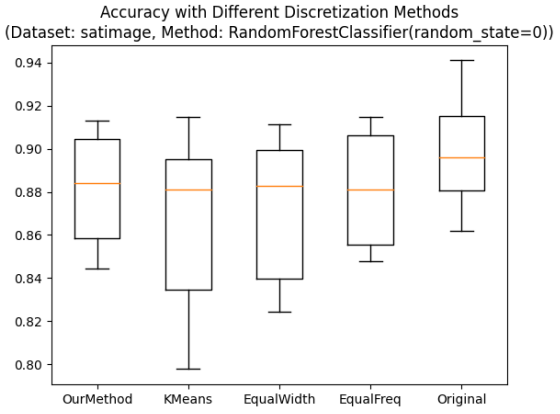


Figure 5.25 Classification accuracy of our method and three benchmark methods in *SatImage* dataset.

5.3.3. Experiments on Shuttle Dataset

This dataset is prepared for classification tasks and contains the data on the positioning of radiators in the Space Shuttle. It consists of 9 continuous attributes and seven class labels. Also, the class distribution is skewed and nearly 80% of the observations belong to class 1. Hence, the donors of the dataset noted that the default accuracy score is 80% and the goal of classification is achieving 99-99.9% accuracy.

The discretization has the same configuration as the experiment on SatImage dataset. Total entropies of each attribute after discretization are presented in Figure 5.26. We see that equal-width and k-means binning surpassed our method in terms of the total entropy.

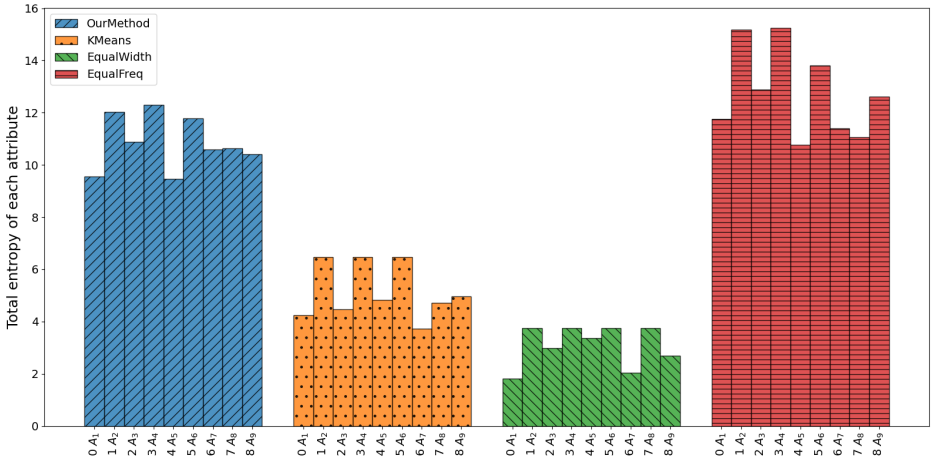


Figure 5.26 Total entropy achieved in each attribute in *Shuttle* dataset.

Next, Figure 5.27 presents the classification accuracy scores of all methods. All experiments used random forest with default parameters and 10-fold cross-validation. Again, our method has the nearest accuracy score to undiscretized data. Besides, we mentioned that this dataset is aimed at achieving at least 99% accuracy, and our method is the closest to reach that score. This experiment shows that although k-means and equal-width approaches benefit from the shape of the dataset and end up with lower entropy levels, their intervals are contextually incorrect and therefore they lose significant accuracy points in prediction tasks.

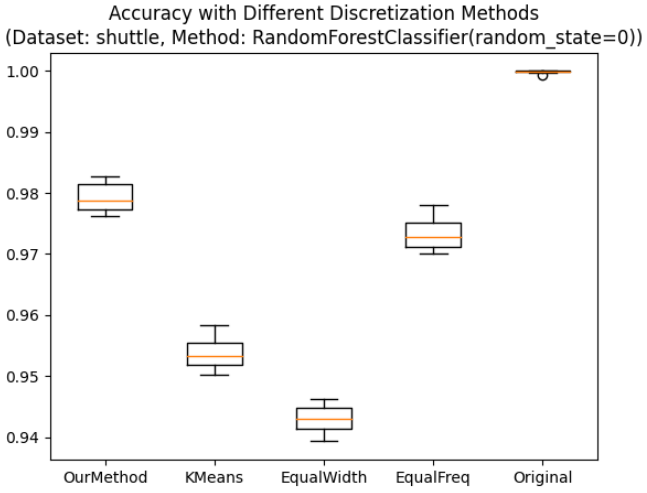


Figure 5.27 Classification accuracy of our method and three benchmark methods in *Shuttle* dataset.

5.3.4. Experiments on Ionosphere Dataset

This dataset contains the signal data collected by radars. It consists of 34 continuous attributes and two class labels. The class labels are either "good" which indicates the evidence for the existence of a structure in the ionosphere, or "bad" which shows that radar signals pass through the ionosphere.

The discretization has the same configuration with the experiment on SatImage dataset. The total entropies of each attribute after discretization are presented in Figure 5.28. Our method could not lower the total entropy of each attribute significantly when compared to univariate benchmark methods.

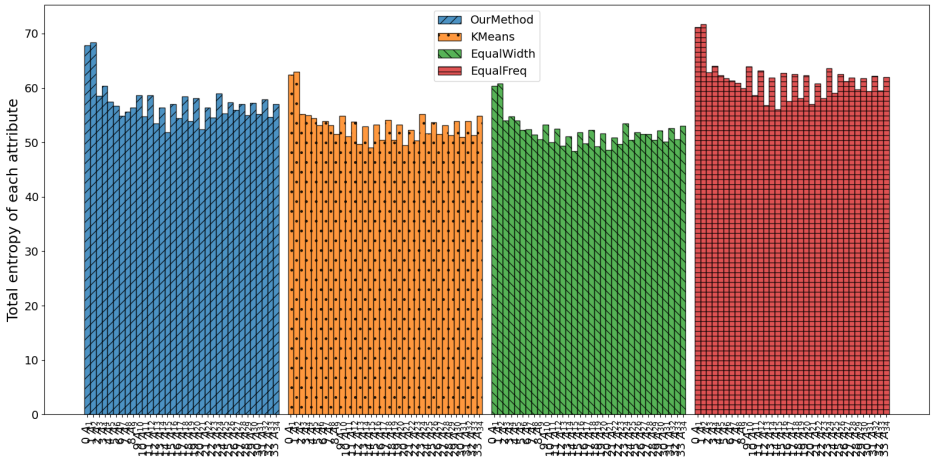


Figure 5.28 Total entropy achieved in each attribute in *Ionosphere* dataset.

Next, Figure 5.29 presents the classification accuracy scores of all methods. All experiments used random forest with default parameters and 10-fold cross validation. We can see that our method showed comparable scores with other methods. Interestingly, the three benchmark approaches provided better classification accuracy not only with respect to our model but also with respect to the original dataset. Further testing and investigation is necessary to expose the causes of this result which we did not have the time to do in this study.

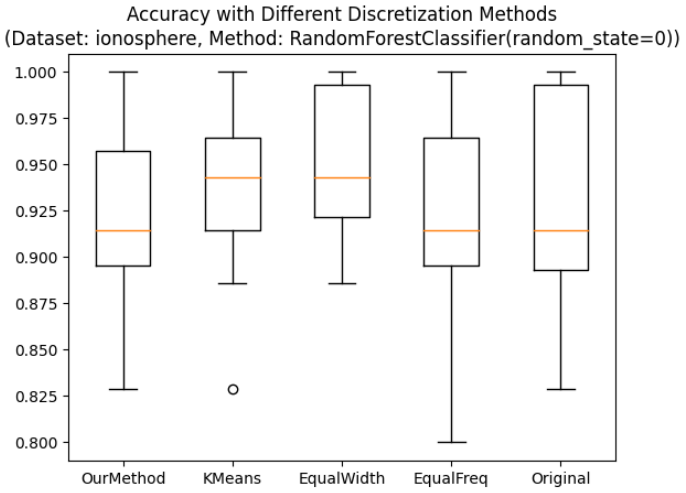


Figure 5.29 Classification accuracy of our method and three benchmark methods in *Ionosphere* dataset.

5.4. Execution time

Our method evaluates all attributes at once to calculate graph edge weights via entropy values, and then, solves a graph shortest path problem with a dynamic programming approach. This increases the complexity of overall procedure. We've run different tests with different sized synthetic datasets in order to demonstrate the effect of increasing sizes. Our setup is a laptop with AMD Ryzen-7 5700U processor and 40 gigabytes of RAM. Notice in Figure 5.30 that increasing the data size 10-fold (from 10,000 to 100,000) increased the running time only 2-fold. That is due to the fact that we perform sub-sampling on unique values of an attribute and limit the maximum number of nodes in the graph to 1,000. In our experiments, we did not notice a significant effect of this on discretization intervals and prediction accuracy. Moreover, we benefit from PCA dimensionality reduction method to cope with high dimensional datasets with tens or hundreds of attributes. It is worth noticing that increasing the bin size of discretization caused a little increase in running time because additional operations are performed to find a lengthier path during graph path search phase.

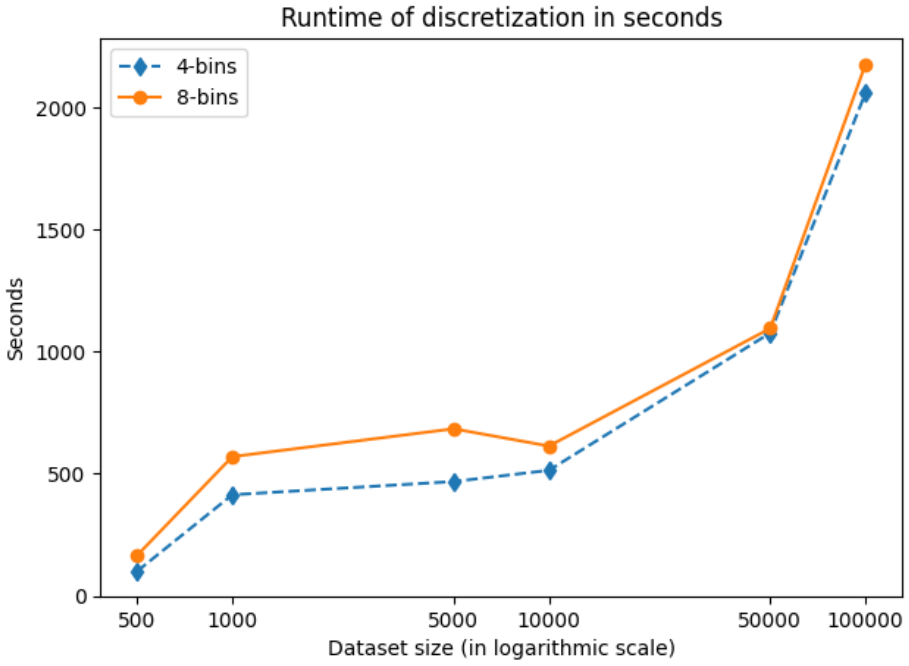


Figure 5.30 Runtime of discretization (4-bins and 8-bins) on synthetic datasets.

6. CONCLUSION

In this study, we established a static, multivariate, unsupervised and global discretizer by making use of data entropy and constrained shortest path search. We tested the proposed method using a synthetic dataset where the attributes are non-trivially correlated, to expose its capability of distinguishing related observations by considering all attributes and correlations within.

Also, we made experiments on well-known real datasets and demonstrated that our method's discretization intervals are more meaningful for exploratory tasks and more capable of capturing hidden patterns as compared to the intervals of univariate benchmark methods. In addition, on classification tasks, our method significantly generally surpassed univariate benchmark methods and showed comparable results with other multivariate discretizers such as MVD [16], CPD [17], and IPD [18].

In the future, we are expecting to decrease the running time of our algorithm, which is significantly more than our benchmark algorithms. For this purpose, we expect to improve our penalty value searching routine which is the main bottleneck in the algorithm. We also expect to make it more autonomous by discarding the bin count thresholds.

Finally, we observed that in some datasets with unique distribution shapes our method performs worse than the univariate benchmark approaches. The underlying reasons of this requires further investigation, which we expect to conduct in the future. We are also interested in discovering modifications to our algorithm to alleviate this issue.

REFERENCES

[1] Salvador Garcia, Julian Luengo, and Francisco Herrera. Discretization. In *Data Preprocessing in Data Mining*, pages 245–283. Springer, **2015**.

[2] J. Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 16. Elsevier, **1993**. doi:10.1007/BF00993309.

[3] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, Philip S Yu, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, **2008**.

[4] Sergio Ramírez-Gallego, Salvador García, Héctor Mouriño-Talín, David Martínez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benítez, and Francisco Herrera. Data discretization: taxonomy and big data challenge. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(1):5–21, **2016**.

[5] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Machine learning proceedings 1995*, pages 194–202. Elsevier, **1995**.

[6] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1022–1027. **1993**.

[7] Huan Liu and Rudy Setiono. Feature selection via discretization. *IEEE Transactions on knowledge and Data Engineering*, 9(4):642–645, **1997**.

[8] Son H. Nguyen and Andrzej Skowron. Quantization of real value attributes-rough set and boolean reasoning approach. In *Proceedings of the Second Joint Annual Conference on Information Sciences*. Citeseer, **1994**.

[9] Randy Kerber. Chimerge: Discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 123–128. AAAI Press, **1992**. ISBN 0262510634.

[10] Petri Kontkanen and Petri Myllymäki. Mdl histogram density estimation. In Marina Meila and Xiaotong Shen, editors, *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pages 219–226. PMLR, San Juan, Puerto Rico, **2007**.

[11] Marco Vannucci and Valentina Colla. Meaningful discretization of continuous features for association rules mining by means of a som. In *ESANN*, pages 489–494. **2004**.

[12] Gabi Schmidberger and Eibe Frank. Unsupervised discretization using tree-based density estimation. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 240–251. Springer Berlin Heidelberg, Berlin, Heidelberg, **2005**. ISBN 978-3-540-31665-7.

[13] Marenglen Biba, Floriana Esposito, Stefano Ferilli, Nicola Di Mauro, and Teresa M. A. Basile. Unsupervised discretization using kernel density estimation. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 696–701. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, **2007**.

[14] Artur J. Ferreira and Mário A.T. Figueiredo. An unsupervised approach to feature discretization and selection. *Pattern Recognition*, 45(9):3048–3060, **2012**. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2011.12.008.

[15] Yoseph Linde, Andrés Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, **1980**. doi: 10.1109/TCOM.1980.1094577.

[16] Stephen Bay. Multivariate discretization for set mining. *Knowledge and Information Systems*, 3:491–512, **2001**. doi:10.1007/PL00011680.

[17] Sameep Mehta, Srinivasan Parthasarathy, and Hui Yang. Toward unsupervised correlation preserving discretization. *Knowledge and Data Engineering, IEEE Transactions on*, 17:1174–1185, **2005**. doi:10.1109/TKDE.2005.153.

[18] Hoang-Vu Nguyen, Emmanuel Mueller, Jilles Vreeken, and Klemens Boehm. Unsupervised interaction-preserving discretization of multivariate data. *Data Mining and Knowledge Discovery*, 29:296–297, **2015**. doi:10.1007/s10618-014-0353-2.

[19] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, **1948**. doi:10.1002/j.1538-7305.1948.tb01338.x.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, **2011**.

[21] Dheeru Dua and Casey Graff. UCI machine learning repository, **2017**.

APPENDIX

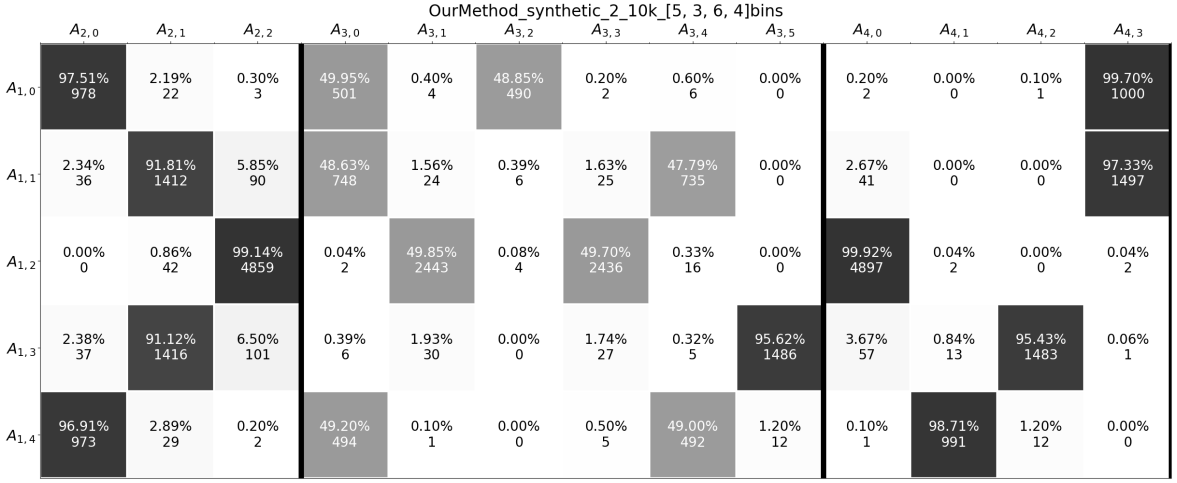


Figure 6.1 Percentage matrix of A_1 from synthetic dataset after discretized with our method.

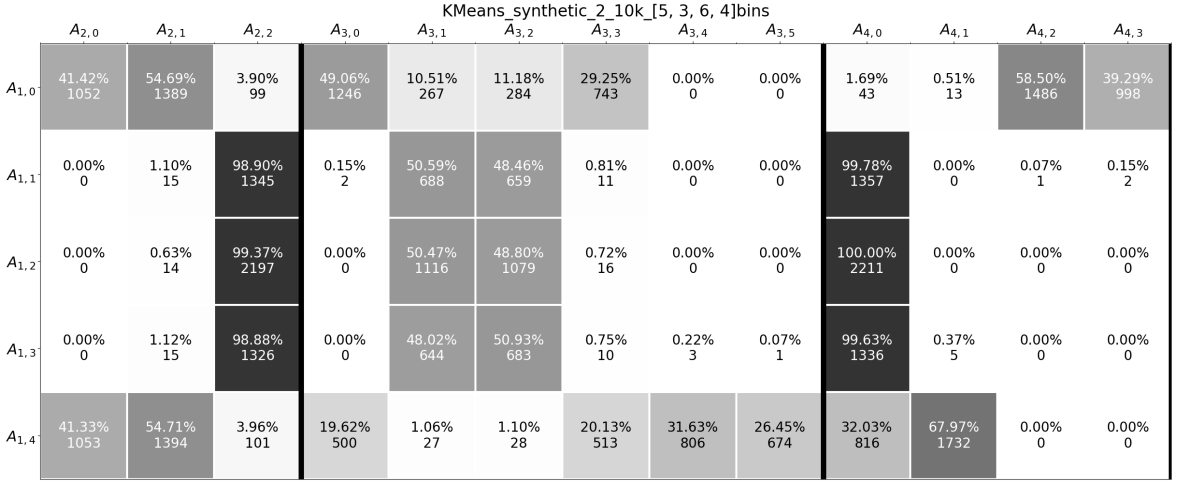


Figure 6.2 Percentage matrix of A_1 from synthetic dataset after discretized with k-means.

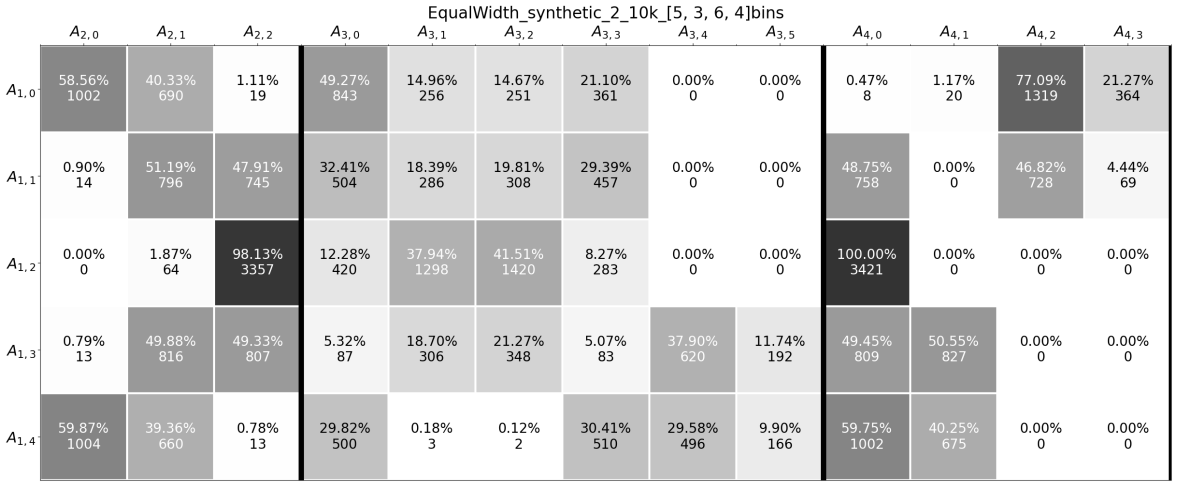


Figure 6.3 Percentage matrix of A_1 from synthetic dataset after discretized with equal-width.

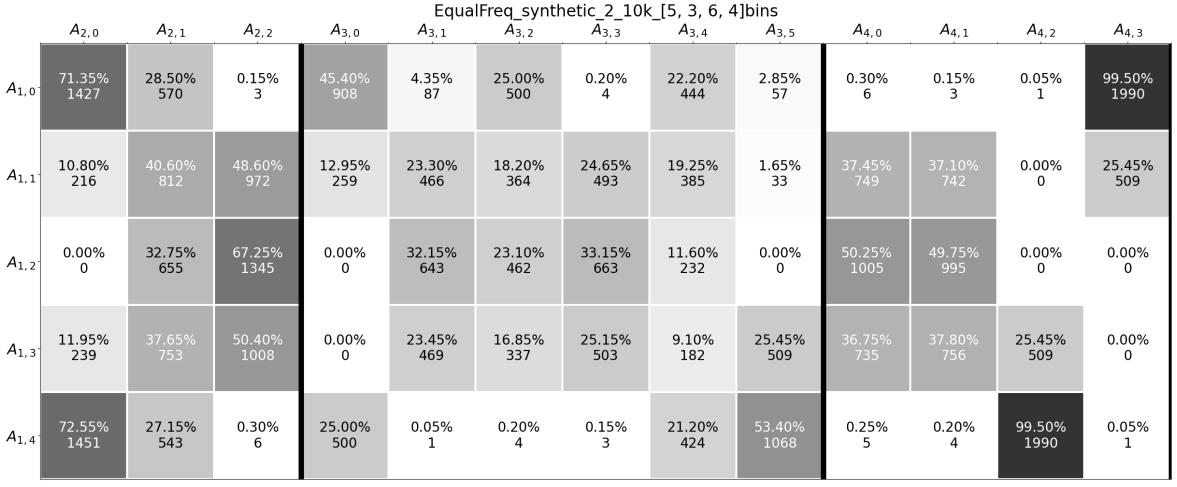


Figure 6.4 Percentage matrix of A_1 from synthetic dataset after discretized with equal-frequency.

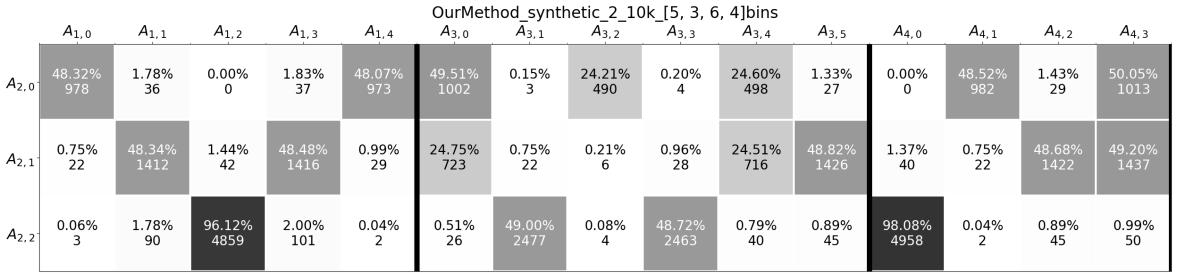


Figure 6.5 Percentage matrix of A_2 from synthetic dataset after discretized with our method.

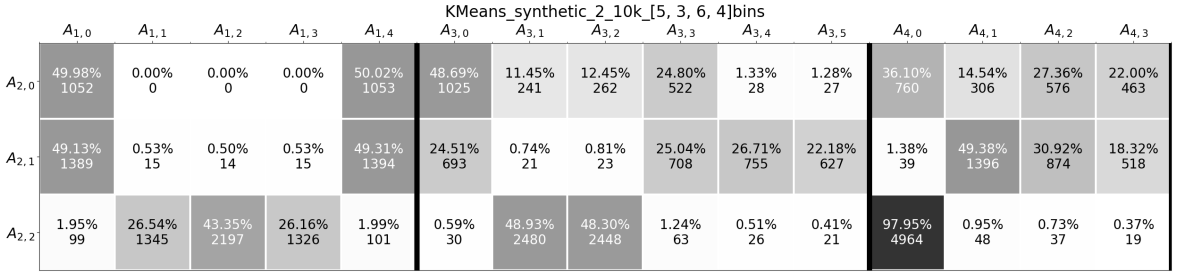


Figure 6.6 Percentage matrix of A_2 from synthetic dataset after discretized with k-means.

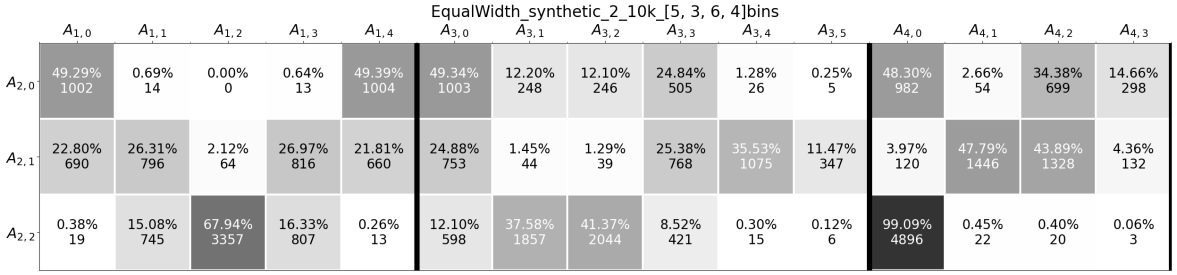


Figure 6.7 Percentage matrix of A_2 from synthetic dataset after discretized with equal-width.

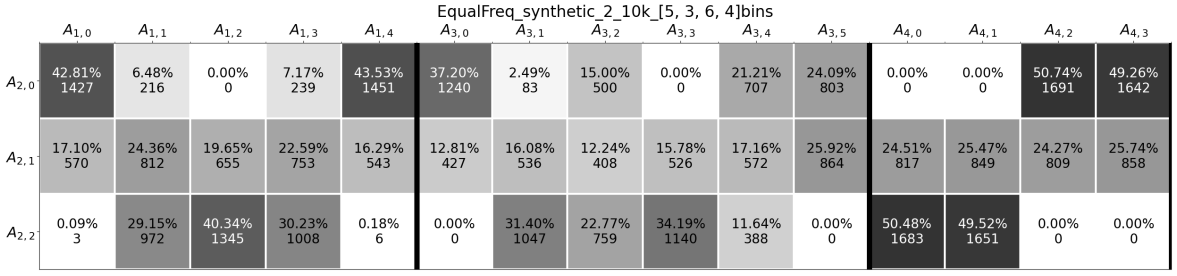


Figure 6.8 Percentage matrix of A_2 from synthetic dataset after discretized with equal-frequency.

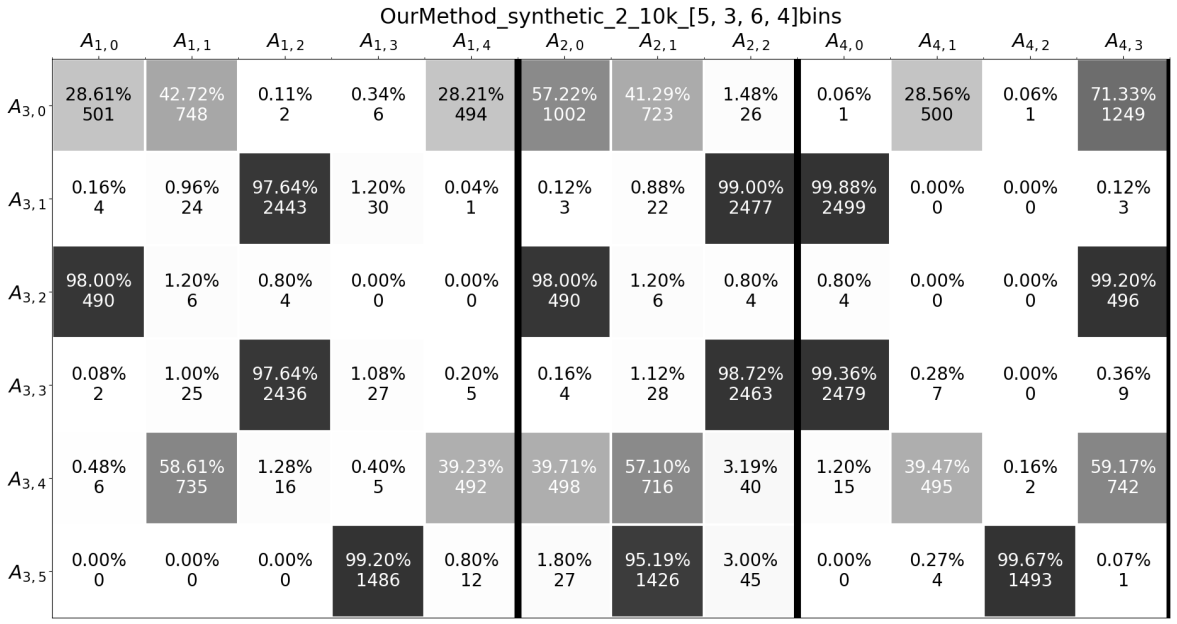


Figure 6.9 Percentage matrix of A_3 from synthetic dataset after discretized with our method.

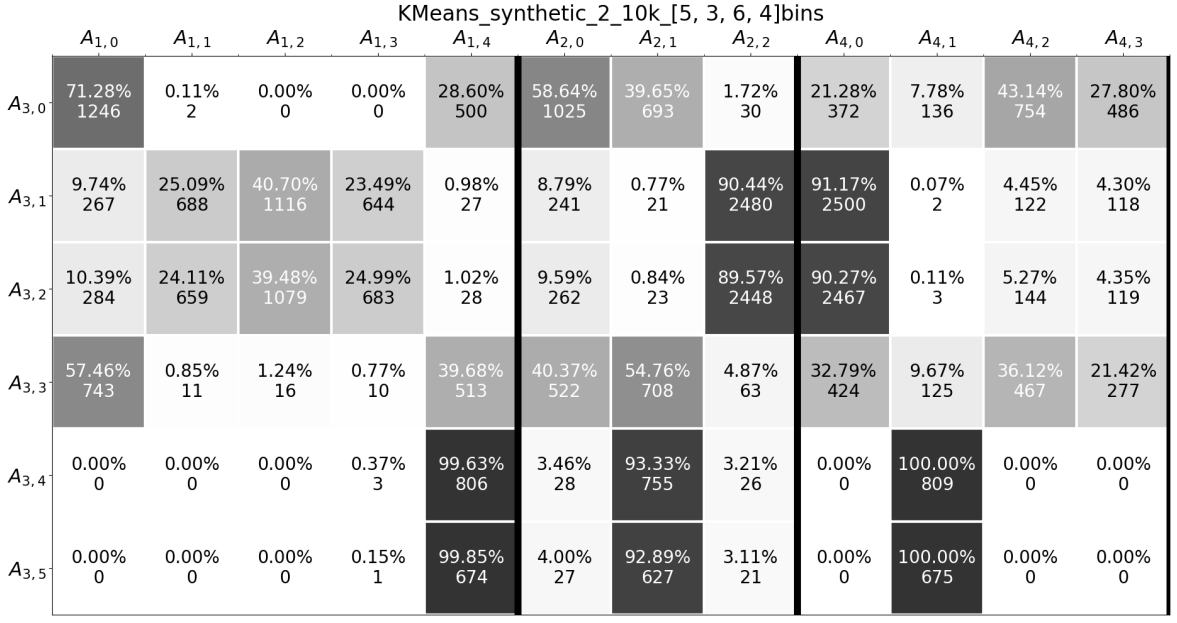


Figure 6.10 Percentage matrix of A_3 from synthetic dataset after discretized with k-means.

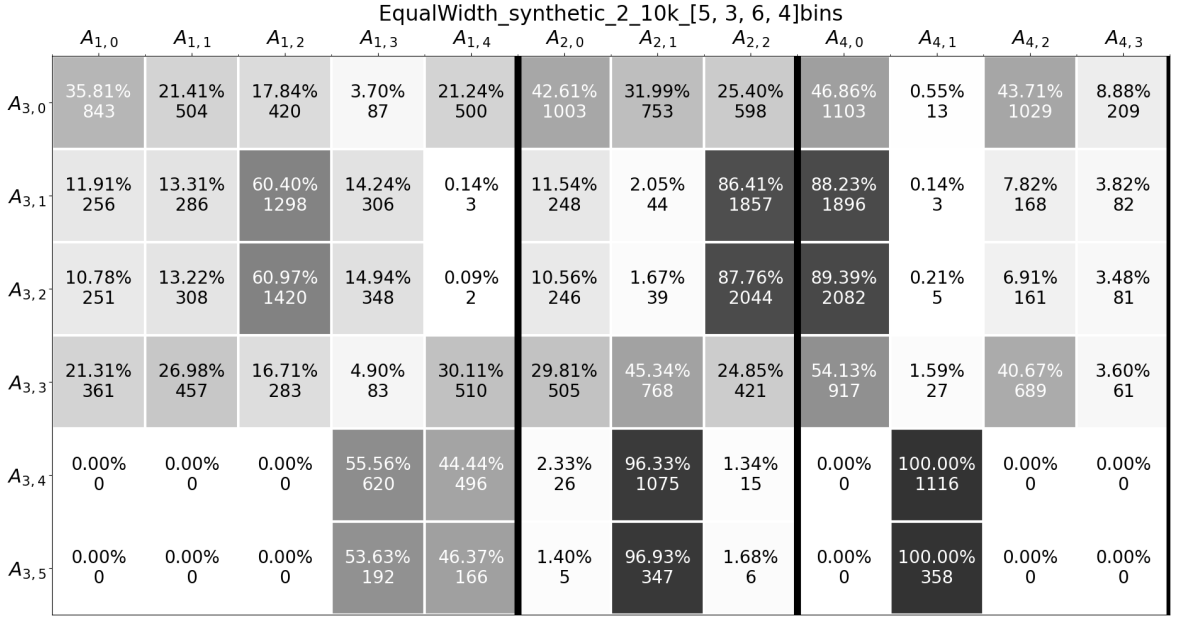


Figure 6.11 Percentage matrix of A_3 from synthetic dataset after discretized with equal-width.

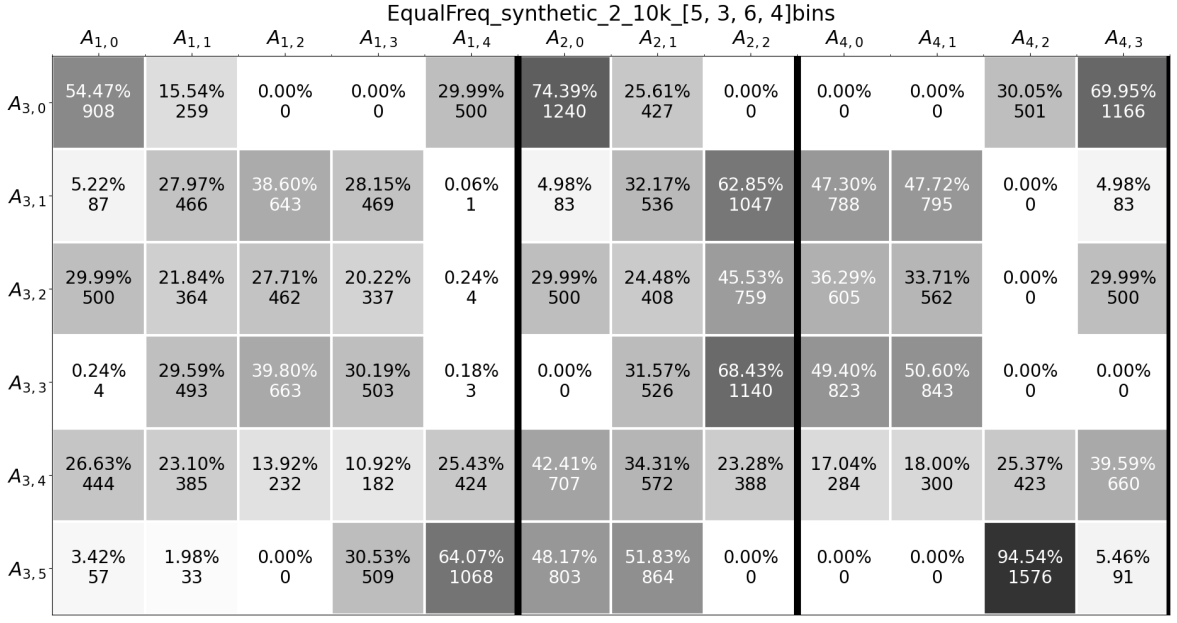


Figure 6.12 Percentage matrix of A_3 from synthetic dataset after discretized with equal-frequency.

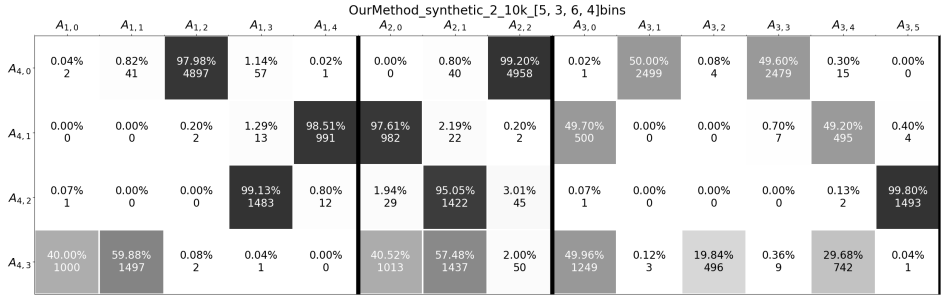


Figure 6.13 Percentage matrix of A_4 from synthetic dataset after discretized with our method.

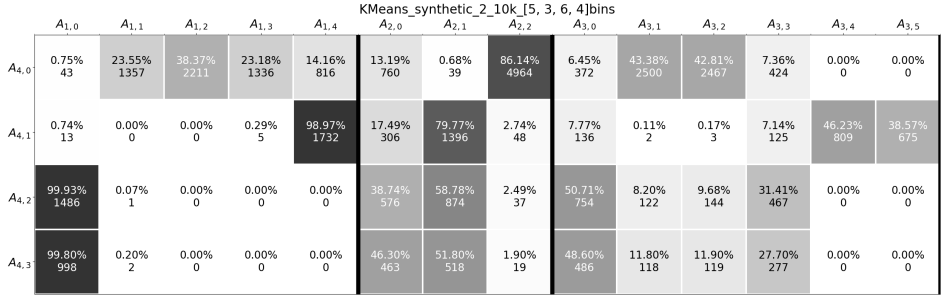


Figure 6.14 Percentage matrix of A_4 from synthetic dataset after discretized with k-means.

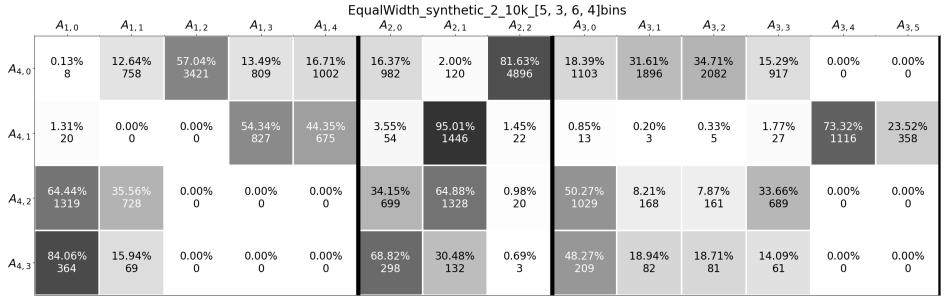


Figure 6.15 Percentage matrix of A_4 from synthetic dataset after discretized with equal-width.

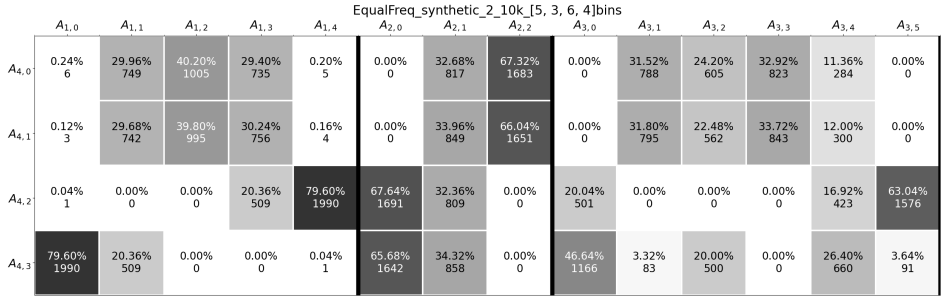


Figure 6.16 Percentage matrix of A_4 from synthetic dataset after discretized with equal-frequency.