AN IMPLEMENTATION FOR A HEURISTIC RANGE-BASED CLASSIFICATION METHOD

PHUONG ANH TRAN

MSc Computing

A DISSERTATION SUBMITTED FOR THE DEGREE OF MSC COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

CARDIFF UNIVERSITY

2016

Declaration

I hereby declare that this dissertation is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the dissertation.

This dissertation has also not been submitted for any degree in any university previously.

Phuong Anh Tran

June 2016

Acknowledgments

To my supervisor, Dr Jinhua Shao, whom inspires me with confidence and beyond.

Abstract

Contents

List of Figures

List of Tables

List of Algorithms

Section 1

Introduction

The ability to extract meaningful information from a set of data has been a crucial problem within data mining and machine learning research. One approach is to recognise certain patterns within a dataset, then transform these patterns into rules. In the future, these rules could be used to predict an classified outcome for any given instance of that dataset. The ability to transform such patterns is thus a deciding factor in producing accurate predictions. Subsequently, its accuracy relies on the choices of methods and algorithms in learning and deriving useful and relevant rules.

The goal of this project is to investigate an algorithm to derive classification rules for range-based data. Ranges pose a more challenging problem to assess since there are more possible patterns to deduce than that of categorical values. Therefore, before patterns can be learnt, the data must be split into 'right intervals' [src] so that the rules devised will be more fitting. The process of deducing relevant subset of data thus plays a significant part in deciding the rule accuracy, and an important step before actual classification can take place.

In details, this project analyse and implement a method that aims to solve the above issues i.e. to classification rules for range-based numerical dataset, developed in a previously published paper by Shao, J. and Tziatzios, A. [src]. The original method withholds two distinctive features: firstly, the data trimming process is done by an algorithm similar to Kadane's [src] solving maximum-array problem, which is highly efficient due to its linear time complexity (logN). Secondly, in order to mimic that max-sum method, chosen ranges are analysed by

assigning binary tags upon their covered items depending on the items' class values. By using class values, the given ranges can be narrowed down to to be more relevant to the rules derived. These two features make for a fast and efficient way to classify range-based dataset. This method is then analysed further in order to produce an improved[?] version that is implemented as a final software using Java.

What was improved?

Outcome? Your own work?

SECTION 2

Background

The project implements an algorithm originated from [SRC]. The algorithm utilises a similar approach to association rule mining: it searches for associated ranges in each attribute, then combine these ranges to form a larger rule. To make this search more efficient, class values are used to guide the search i.e. only ranges that have relevant class tags are considered. The algorithm is explained in the following sections.

2.1 Preliminaries

Assuming that we have a dataset that could be presented as a table $T(A_1, A_2, A_3, ..A_m, C)$, where $A_{j,m}$ as $1 \leq j \leq m$ are range attributes and C is the categorical class value. A single tuple within T is denoted as $t_k = (v_1, v_2, v_3..v_m, C)$ where v_m is a value under A_m.

A sample dataset served as an example for this section can be shown in Table 2.1 below:

Table 2.1: A sample dataset as table

T	A_1	A_2	A_3	A_4	C
$t1$	0.75	1.45	2.13	4.56	c_1
$t2$	0.64	1.62	2.64	4.75	c_2
$t3$	0.71	1.21	3.11	3.97	c_1
$t4$	0.57	1.23	2.75	4.24	c_1
$t5$	0.80	1.53	2.34	4.11	c_2

Definition 1 (range) Assume within the domain of attribute A exists two values a and b that represents a continuous range over A_j, denoted $r = [a, b]_{A_j}$. This range covers a set of values in A that lies between a and b.

Example 1: From Table 2.1, a range of $[0.64, 0.75]_{A_1}$ would cover a set of values 0.64, 0.71 and 0.75 in A_1.

Definition 2 (cover) Assume $r = [a, b]_{A_j}$ to be a range over attribute A. This range r covers a set of tuples where their values are between a and b where $a \leq v_j \leq b$. This set of tuples that is covered by r is denoted $\tau(r)$.

Example 2: From Example 1, a range of $[0.64, 0.75]_{A_1}$ would cover a set of tuples t_2, t_3, t_1 hence its cover is t_2, t_3, t_1.

Definition 3 (associated ranges) Assume $r_1 = [a_1, b_1]_{A_1}$ to be a range over A_1 and $r_2 = [a_2, b_2]_{A_2}$ to be a range over A_2. Those ranges are associated ranges if $\tau(r_1) \cap \tau(r_2) \neq \varnothing$

Example 3: Assume $r_1 = [0.64, 0.75]_{A_1}$ and $r_2 = [1.21, 1.45]_{A_2}$. Table 2.1 shows that r_1 covers t_1, t_2, t_3 and r_2 covers t_1, t_4, t_3, which means $r_1 \cap r_2 = \{t_1, t_2, t_3\} \cap \{t_1, t_3, t_4\} = \{t_1, t_3\}$. Since there are mutual tuples between these ranges, r_1 and r_2 are associated ranges.

Definition 4 (range-based classification rule) Assume c to be a class value from C and $r_1, r_2, r_3..r_h$ be a set of associated ranges. $r_1, r_2, r_3..r_h \rightarrow c$ is a range-based classification rule.

By definition, any tuple t_k can form a rule. However, this will not be accurate nor useful. To determine whether a rule is qualified, there are criteria to be used, which are to be discussed in the sub-section below.

2.2 Criteria

There are three criteria used in this algorithm: support, confidence and density. This sub-section presents the formal definitions for those measures, while examples are to be provided in the next sub-section where the actual algorithm is reviewed.

Definition 5 (support) Assume T to be a table and $r_1, r_2, r_3..r_h \to c$ to be a range-based

classification rule derived from T. The support for r, provided $| \, . \, |$ being the size of a

set, is:

$$\sigma(r) = \frac{| \, \tau(r_1) \cap \tau(r_2) \cap ... \cap \tau(r_h) \, |}{| \, T \, |}$$

Definition 6 (confidence) Assume similar settings from Definition 5, the confidence for r

in T is:

$$\delta(r) = \frac{| \, \tau(r_1) \cap \tau(r_2) \cap ... \cap \tau(r_h) \cap \tau(c) \, |}{| \, \tau(r_1) \cap \tau(r_2) \cap ... \cap \tau(r_h) \, |}$$

Definition 7 (density) Assume similar settings from Definition 5, the density for r in T

is:

$$\gamma(r) = \frac{| \, \tau(r_1) \cap \tau(r_2) \cap ... \cap \tau(r_h) \cap \tau(c) \, |}{| \, \tau(r_1) \cup \tau(r_2) \cup ... \cup \tau(r_h) \, |}$$

2.3   Original algorithm

As mentioned, the algorithm provides a more efficient way to discover associated ranges than
the traditional brute-force solution. Instead of traversing through all possible combinations
of ranges among all attributes, this algorithm first divides the data into sub-data pools
in accordance to its class value. Then, under each attribute, it searches for sub-ranges
that could pass certain user-defined thresholds (confidence, support and density). These
sub-ranges are then collected and ready to be tested for possible combinations between each
other that passes certain thresholds similar to the previous step, which are then collected
and ready to be tested for larger combinations. Additionally, after larger combinations are
found, an extra adjustment step is done on class values of relevant tuples within the new
associated ranges. Afterwards, new associated ranges are ready to be analysed again to
find more possible sub-ranges and new combination of sub-ranges. This iteration continues
until no possible combinations can be found, so that the last standing associated ranges are
translated into the concluding classification rule for that class value.

To get a clearer structure of the algorithm, its proceedings can be divided into two phases:

Phase I (Analyse) This phase has its primary concern as to analyse ranges. Based on the algorithm description, it includes:

1. Partition data into sub-data pool using class values.

2. For each attribute in sub-data pool, find a maximum and minimum range within that attribute and use it as a starting range for next step.

3. From min-max starting ranges, search for sub-ranges that passes certain user-specified thresholds such as confidence, support and density. Sub-ranges are collected and passed onto Phase II to generate larger associated ranges.

Phase II (Generate) This phase has its primary concern as to generate larger associated ranges for the next iteration. Based on the algorithm, it includes:

1. For each sub-range obtained from last phase, check for possible combination of that range with the remaining ones.

2. Adjust class values of relevant tuples $r_1, r_2, r_3..r_h \rightarrow c$ under the new combined sub-range.

Details for each phase are as follows:

2.3.1 Original Phase I (Analyse)

Given a table $T(A_1, A_2, A_3, ..A_m, C)$, the algorithm attemps to seek for classification $r_1, r_2, r_3..r_h \rightarrow c_j$ by finding, among instances that are under class $c_j \subset C$, a number of associated ranges $r_1, r_2, r_3..r_h$ that have at least a minimum support (σ_{min}), density (δ_{min}) and confidence (γ_{min}) that had previously been specified by the user. This process can be explained in Phase I (Analyse) in which the algorithm seeks for associated ranges and checks if thresholds are satisfied. From [SRC], this phase is outlined as below:

A step-by-step explanation of this algorithm is as follows:

Algorithm 1 *R-CARM*

---

input: $T(A_1, A_2, \ldots, A_m, C)$, where each A_i is numerical
and C is categorical
output: R, a set of range-based classification rules

1. **for each** $c_j \in C$ **do**
2. $T_{c_j} \leftarrow select(T, c_j)$
3. $CR_1 \leftarrow \{[v_{min}, v_{max}]_{A_1}, [v_{min}, v_{max}]_{A_2}, \ldots,$
        $[v_{min}, v_{max}]_{A_m}\}$ from T_{c_j}
4.   **for** $(k = 1,\ CR_k \neq \varnothing,\ k{+}{+})$ **do**
5. $LR_k \leftarrow \varnothing$
6.   **for each** $cr \in CR_k$ **do**
7. $S \leftarrow analyse\_range\ (cr, c_j)$
8.   **for each** $s \in S$ **do**
9.   **if** $\sigma(s) \geq \sigma_{min}$ **and** $\gamma(s) \geq \gamma_{min}$ **then**
10. $LR_k \leftarrow LR_k \cup s$
11.   **if** $\delta(s \Rightarrow c_j) \geq \delta_{min}$ **then**
12. $R \leftarrow R \cup \{s \Rightarrow c_j\}$
13. $CR_{k+1} \leftarrow generate\ (LR_k)$
14. **return** R

Figure 2.1: Original Phase I (Analyse) algorithm

Step 1-2 As the beginning steps of the algorithm, they selects a set of tuples $t_1, t_2, t_3..t_N$ from T that has class value $c_j \subset C$ and store them in T_{c_j}.

Example: Using Table 2.1 from last section, for class c_j where $j = 1$ (class 1), tuple set T_{c_1} under this class are:

Table 2.2: A subset of tuples under class 1

T_{c_j}	A_1	A_2	A_3	A_4	C_j
t1	0.75	1.45	2.13	4.56	1
t3	0.71	1.21	3.11	3.97	1
t4	0.57	1.23	2.75	4.24	1

Step 3 In this single step, from T_{c_j}, for each attribute A_i the algorithm derives a set of ranges $[v_{min}, v_{max}]_{A_i}$ for that attribute. These ranges are formed by seeking for the

minimum and the maximum values of A_i in T_{c_j}. This is denoted as candidate ranges cr, since they are still subjected to a check to see whether they have the required minimum thresholds needed to form the rules. *Example:* From Table 2.2, for each attribute a candidate range can be generated as follows:

$A_1 = [0.57, 0.75]$

$A_2 = [1.21, 1.45]$

$A_3 = [2.13, 3.11]$

$A_4 = [3.97, 4.24]$

Step 4-13 In these remaining steps lie the core analysis of Phase I, which could be separated into three main components: sub-range derivation, threshold check and larger range generation. Firstly, for each range in cr, an analysis is performed to derive possible sub ranges from this range. The reason for this analysis is because for continuous ranges, $[v_{min}, v_{max}]_{A_i}$ could cover more than necessary tuples that consequently reduces the rule accuracy if not trimmed down. *Example:* Consider the example $A_1 = [0.57, 0.75]$ from the previous step. Using this range on the original set of tuples T in Table 2.1, the tuples that the range covers are:

Table 2.3: A subset of tuples under class 1 and range r of attribute 1

T	A_1	C
t1	0.75	1
t2	0.64	2
t3	0.71	1
t4	0.57	1

Under this range $A_1 = [0.57, 0.75]$ which is a candidate rule, its cover of t_1, t_2, t_3, t_4 gives a $\frac{3}{4}$ correct classification. However, by adjusting the range to $A_1 = [0.68, 0.75]$ which covers t_1, t_3, t_4, it now gives $\frac{4}{4}$ correct classification. Although the support for this range is reduced, higher classification accuracy is achieved. Therefore it shows a need to prune(?) these candidate ranges before proceeding to the next step. The

pruning process lies within step 7.

Step 7 The pruning process is represented by the *analyse_range* function, where a method is implemented to find sub-ranges. This method is based on the solution to the max sum problem [SRC], which is for finding a maximal gained sub-array within a given array of discrete numbers (its details will be discussed further in the next section). To apply the similar approach to finding maximum sub-ranges, for class c_j, each tuple in T_{c_j} that is covered by range $[v_{min}, v_{max}]_{A_i}$ is scored with 1 if its class is c_j and -1 otherwise. This is to convert continuous values into discrete (in this case, binary) scores of numbers, that are then used to calculate the max sum scores. Those max sum scores represents the candidate sub-ranges, which could be the desired sub-ranges once they satisfy the support confidence and density constraints. There can be multiple sub-ranges and thus all sub-ranges are collected and store the result in set S. Sub-ranges in set S is checked in the following step 8-12 for threshold requirements.

Example: From Table 2.3, T_{A_1} where $A_1 = [0.57, 0.75]$ and the class value c_1 is 1, the tuples t_1, t_2, t_3, t_4 are covered under this rule and can be presented in ascending order as:

Table 2.4: A subset analysed by max-sum method for sub-ranges

	t_2	t_4	t_3	t_1
Attribute value	0.64	0.68	0.71	0.75
Class value	c_2	c_1	c_1	c_1
Binary values	-1	1	1	1

Assume the max-sum approach requires 100% confidence i.e. correct classification $(c = 1)$ for each covered tuple, the resulted optimal binary array will be 111 which represents a sub-range $s = [0.68, 0.75]$ that covers tuples t_4, t_3, t_1

Step 8-10 These steps perform a support and density threshold check on sub-ranges in

set S. For each sub-range s in S, a threshold check is performed to see whether it has sufficient support and density specified (Step 9). If sub-range s does, it is kept in LR_k where candidate sub-ranges are stored to form larger associated ranges in the next iteration (Step 10). However, if it does not satisfy the specified thresholds, sub-range s is discarded as it cannot be used to form larger associated ranges due to support and density measures being monotonic [SRC]. All checked ranges s in LR_k are then passed on for a final check on confidence threshold in the following step 11-13.

Example: From Table 2.4 where sub-range $s = [0.68, 0.75]_{A_1}$ is found for attribute A_1, the support and density can be computed as follows:

$$\sigma_s = \frac{\{t_3, t_4, t_1\}}{\{t_2, t_4, t_3, t_1\}} = \frac{3}{4} = 0.75$$

$$\gamma_s = 1$$

Assume the condition specified is $\sigma_s \geq 0.6$ and $\gamma_s \geq 0.9$, then sub-range $s = [0.68, 0.75]_{A_1}$ satisfies the required support and density threshold and therefore accepted. Assume, with the similar rule, another sub-range $s = [1.21, 1.45]_{A_2}$ is found for attribute A_2 from an initial range $[1.21, 1.45]_{A_2}$. The support and density can be computed as follows:

$$\sigma_s = \frac{\{t_3, t_4, t_1\}}{\{t_3, t_4, t_1, t_5, t_2\}} = \frac{3}{4} = 0.75$$

$$\gamma_s = 1$$

With the same support and density measures, sub-range $s = [1.21, 1.45]_{A_2}$ is also accepted. These two sub-ranges are stored in LR_k for the next steps, hence $LR_k = \{[0.68, 0.75]_{A_1}, [1.21, 1.64]_{A_2}\}$.

Step 11-12 These steps perform the last check for confidence threshold on ranges in LR_k. Using class value c_j, it checks whether sub-range s in LR_k can be used to form range-based classification rules with sufficient confidence (Step 11-12). If s satisfies the confidence test, sub-range s is then kept in LR_k which is then used to form larger association of ranges. The process of forming larger associated ranges is in Phase II

(Generate) in Step 13. However, in cases where s fails to pass the confidence threshold, it is discarded and no longer considered in the next iteration. *Example:* In Step 8-10 example, sub-range $s = [0.68, 0.75]_{A_1}$ is shown to have passed threshold for support and density. It is then checked for confidence measure, which is as follows:

$$\delta_s = \frac{\{t_4, t_3, t_1\}}{\{t_4, t_3, t_1\}} = 1$$

Assume the condition specified is $\delta_s \geq 0.8$, sub-range $s = [0.68, 0.75]_{A_1}$ passes the test and therefore accepted. Similarly for sub-range $s = [1.21, 1.45]_{A_2}$:

$$\delta_s = \frac{\{t_3, t_4, t_1\}}{\{t_3, t_4, t_1\}} = 1$$

Hence $LR_k = \{[0.68, 0.75]_{A_1}, [1.21, 1.64]_{A_2}\}$ is accepted and passed onto Phase II (Generate) in Step 13 to form larger associated ranges.

Step 13 This step, which represents Phase II (Generate), perform the generation of larger associations of ranges CR_{k+1} by a generate function. The process is the core in Phase II (Generate) is described in Phase II sub-section below.

2.3.2 Original Phase II (Generate)

In this phase, the algorithm prepares CR_{k+1} for the next iteration by combining two relevant ranges in LR_k, thus extending the current ranges. The combining process mirrors the A-priori algorithm in generating frequent itemsets in association rule mining. Although A-priori applies on categorical values, it is adapted to this range-based classfication algorithm by using an additional step on range adjustment. The algorithm is denoted in Figure 2.2 below:

Step 1-5 Given LR_k from previous phase, these steps traverses through all sets of associated ranges in LR_k and for each set of associated ranges ar, it looks for another remaining set ar as a possible set to combine to make a larger range. The condition to combine is in the following Step 6-7.

Algorithm 2 *Generate*

---

input: LR_k, a set of associated ranges that have sufficient
support and density
output: CR_{k+1}, a set of candidate associated ranges

1. $CR_{k+1} \leftarrow \varnothing$
2. **for each** $ar \in LR_k$ **do**
3. $A \leftarrow getAttributes(ar)$
4.   **for each** $ar' \in LR_k$ **do**
5. $A' \leftarrow getAttributes(ar')$
6.   **if** $first_{k-1}(A) = first_{k-1}(A')$ **and**
                $last(A) < last(A')$ **then**
7. $cr \leftarrow [v_{min}, v_{max}]first(A), \cdots ,$
                    $[v_{min}, v_{max}]last(A), [v_{min}, v_{max}]last(A')$
8. $cr \leftarrow adjust\_range \ (cr)$
9. $CR_{k+1} \leftarrow CR_{k+1} \cup cr$
10.   **break**
11. **return** CR_{k+1}

Figure 2.2: Original Phase II (Generate) algorithm

Step 6-7 These steps decide whether ar and ar can be combined by searching whether ar

and ar differs by only one last attribute. $first_{k-1}$ is a function to retrieve the first

attribute in each set, and $last$ is a function to retrieve the last attribute in the same

set. So if the last function of ar returns an attribute that is larger than the that which

is returned by the last function of ar, this pair is accepted. A new extended set of

associated range is formed based on this pair, in which the starting range is the same

$first_{k-1}$ range and the ending range is the larger $last$ range between the two ranges

(Step 7). *Example:* From Phase I example, $LR_k = \{[0.68, 0.75]_{A_1}, [1.21, 1.64]_{A_2}\}$ is

found. Assume another $LR_k = \{[0.68, 0.75]_{A_1}, [2.75, 3.11]_{A_3}\}$ is found. A pair between

these LRs can be formed since $A_1 = A_1$ and $A_3 > A_2$. A new extended set of assciated

ranges cr is $cr = \{[0.68, 0.75]_{A_1}, [1.21, 1.64]_{A_2}, [2.75, 3.11]_{A_3}\}$

However, before this set is passed onto the next iteration, an adjustment is necessary

since similar issue is encountered from Phase I : A combination of two continuous

ranges may cover more than necessary set of tuples. For instance, it occurs when a tuple might be covered in range 1, but not covered in range 2 and vice versa. As the new extended set is passed onto the next iteration, those tuples that fail to be covered in all specified ranges shall be set aside. In this algorithm, the algorithm preserves a strict policy of selection: it presumes those tuples would produce wrong classification at the end, thus will change their classes to $Null$ which signifies a possible eliminiation on the next iteration (recall maxsum process in Phase I where it might eliminates tuples that are not under the same class value to trim down the candidate range). *Example:* An extended set of associated ranges $cr = \{[0.68, 0.75]_{A_1}, [1.21, 1.64]_{A_2}, [2.75, 3.11]_{A_3}\}$ is found from the last step. This set cr covers tuples under A_1, A_2, A_3 respectively as:

$$cr = \{[0.68, 0.75]_{A_1}, [1.21, 1.64]_{A_2}, [2.75, 3.11]_{A_3}\}$$

$$\{t_4, t_3, t_1\}\{t_3, t_4, t_1\}\{t_4, t_3\}$$

It can be seen that tuple t_1 is not covered under A_3 range. Initially, t_1 has class c_1, but since it was not covered completely under this set, the algorithm will convert c_1 to $Null$ in t_1. From Table 2.1, t_1 is now presented as:

Table 2.5: A tuple turned Null in class value during generate phase

T_{c_j}	A_1	A_2	A_3	A_4	C_j
t1	0.75	1.45	2.13	4.56	Null

Its class will be kept for all iterations under class c_1.

Once Phase II (Generate) is completed, it repeats Phase I (Analyse) but on the new extended set of associated ranges instead. The iteration continues until no possible pair can be formed, which is when the algorithm is concluded. The remaining pair become the classification rule for that class.

However, it must be noted that this null-class converting process is replaced in this

project algorithm, as this project adopts a stricter policy that eliminates partly-covered tuples instead of leaving it for consideration in the next iteration. Explanation and reasoning for such approach is discussed in the next section where the project algorithm is reviewed.

SECTION 3

Project Algorithm

Based on the original algorithm discussed in the previous section, initially this project only aimed to implement the exact algorithm into hard code. However, during the implementation process, several adjustments were agreed to be made. Some adjustments (e.g additional steps in scanning data input, display final rules to screen) are for the neccessary code adaption i.e. pre-processing data for input and output, at beginning and end respectively. Others (e.g max-sum, threshold check, null-class converter) are for possible improvements in algorithmic in the programme.

Respectively, for the technical-related changes, section 4 will discuss them in depth where the actual code implementation is concerned. For the algorithmic changes, they are to be discussed in the remaining sections below.

3.1 Similarities

As the projectâĂŹs main goal is to implement the algorithm, the programme structure is entirely similar to the initial algorithm i.e. Phase I (Analyse) and Phase II (Generate) is kept in order. Details are explained in the following sub-sections.

3.1.1 Project Phase I (Analyse)

Phase I contains the range analysis method (max-sum) and the threshold checker (for support, density and confidence). In this project, similar steps are implemented: Firstly, the table

of data is divided into sub-data by class value (step 1-2). Secondly, within this sub-data, a $[v_{min}, v_{max}]$ range is derived for each of the attribute (step 3). Thirdly, for each of these ranges within an attribute, a max-sum analysis is conducted so that sub-ranges can be deduced from the original range (step 4-7). Finally, each of these sub-ranges is then checked to see whether it passes the specified thresholds of support, confidence and density (step 8-12). The accepted sub-ranges are collected and sent to Phase II (Generate) (step 13).

However, in this project, several steps are done differently in terms of algorithm compared to the initial algorithm. Firstly, the max-sum analysis is adapted to fit in this range-based classification problem. While it was not discussed in details in the initial paper, there is a difference in this max-sum compared to the original max-sum solution. That is, this max-sum method will not seek for one maximal array but multiple ones depending on a set of pre-specified conditions. Secondly, in the threshold checking part, the order of threshold tested are no longer support - density - confidence, but support - confidence for first iteration and density - support - confidence for the second iteration onwards. It is to adapt to the different structural as well as technical requirements of different iteration e.g. density is not relevant in the first iteration where no extended ranges have been made yet. Details are to be discussed in the next section.

3.1.2 Project Phase II (Generate)

Phase II contains the range combination method (a-priori) and the additional adjustment step (null-class converter). In this project, the first method is implemented but the second part is replaced by an elimination step (eliminator): Firstly, the accepted sub-ranges from Phase I are considered and grouped with each other under the condition that one of them has an extendable range (step 1-7). Secondly, according to the original algorithm, an adjustment method is required by changing the class value of partly-covered ranges to Null (step 8).

However, instead of turning tuple's class value to *Null* only, this project adopts a stricter policy: all partly-covered ranges are eliminated and not considered for the next iteration. It is due to the idea taken from the paper that those tuples are deemed to badly influence the rule accuracy in the long run. While they might as well get eliminated in the next

max-sum iteration, the algorithm in this project does not at all consider those tuples to save on processing time and enhance the rule accuracy in exchange for less support. Hence the null-class converting process (step 8) is replaced with an elimination process. [SUPPORTINGPAPER?] Details will be discussed in the next section.

3.2 Differences

As mentioned in the previous section, there are two main algorithmic changes in this project compared to the initial algorithm. They are: [FIG?]

3.2.1 Project Phase I (Analyse)

- Max-sum method → maxsum() in detail

  Original version: Max-sum on multiple ranges - not elaborated on paper

  Project version: Max-sum on multiple ranges in details - elaborated

- Threshold checking method → separated check()

  Original version: Threshold check consists of three orderly checks: support-density and confidence

  Project version: Threshold check consists of two orderly checks for first iteration and three orderly checks for second iteration onwards. The order of each threshold check is also altered (for first iteration: support-confidence; for second iteration: density-support-confidence)

3.2.2 Project Phase II (Generate)

- Null-class converting method → adjust()

  Original version: a conversion where partly covered tuples have their classes change to Null

Project version: a total elimination where partly covered tuples are complelely removed and not considered during the next iteration

3.3   Final Construction