Continuous Variable Binning Algorithm to Maximize Information Value Using Genetic Algorithm

Nattawut Vejkanchana(✉) and Pramote Kuacharoen

National Institute of Development Administration, Bangkok, Thailand
nattawut.vej@stu.nida.ac.th, pramote@as.nida.ac.th

Abstract. Binning (bucketing or discretization) is a commonly used data pre-processing technique for continuous predictive variables in machine learning. There are guidelines for good binning which can be treated as constraints. However, there are also statistics which should be optimized. Therefore, we view the binning problem as a constrained optimization problem. This paper presents a novel supervised binning algorithm for binary classification problems using a genetic algorithm, named GAbin, and demonstrates usage on a well-known dataset. It is inspired by the way that human bins continuous variables. To bin a variable, first, we choose output shapes (e.g., monotonic or best bins in the middle). Second, we define constraints (e.g., minimum samples in each bin). Finally, we try to maximize key statistics to assess the quality of the output bins. The algorithm automates these steps. Results from the algorithm are in the user-desired shapes and satisfy the constraints. The experimental results reveal that the proposed GAbin provides competitive results when compared to other binning algorithms. Moreover, GAbin maximizes information value and can satisfy user-desired constraints such as monotonicity or output shape controls.

Keywords: Binning · Genetic algorithm · Data pre-processing · Information value · Constrained optimization

1 Introduction

1.1 Binning

In predictive modeling, binning or discretization is performed to transform a continuous variable into intervals. Each binned interval can be assigned by a value that represents its interval characteristic. The purpose of binning is to (1) increase the stability of the predictive continuous variables, (2) reduce statistical noises and complexities in the variables, (3) reduce the influence of outliers, and (4) standardize both categorical and continuous variables by replacing each binned range with a standardized representative value.

For predictive models in applications that use scorecards such as credit scoring or predictions in healthcare industry, the final scorecard can be used and understood easily since there are less levels in each variable.

The original version of this chapter was revised: The name of the Author has been corrected as "Pramote Kuacharoen". The correction to this chapter is available at https://doi.org/10.1007/978-3-030-32475-9_39

A well-known statistic to measure independent variables predictive power is the Information Value (IV) [1]. The larger IV indicates the higher predictive power of a variable. Therefore, IV can be used to measure the predictive power of a binned variable. A good binning algorithm should bin a variable to have a good IV after binned. We discuss IV in more details in the next section.

There are several binning algorithms. However, to our knowledge, none of them directly aims to maximize the IV. The existing algorithms are further discussed in more details in the related work section.

In this paper, the authors propose a new binning algorithm using a genetic algorithm for continuous variables in binary classification problems. It can achieve good binning characteristics [2] which are (1) missing values are binned separately, and (2) each bin should contain at least 5% percent observations. In addition, the new algorithm can satisfy user-desired constraints such as monotonicity or output shape controls and maximize a desired statistic, IV.

1.2 Information Value (IV)

One of the main statistics we use to analyze predictive variables is the Information Value (IV). This statistic evaluates the association between possible predictors and target variable (e.g., bad loan in credit scoring or churn in churn prediction). The IV measures how much a predictor can differentiate between good and bad instances. The higher IV indicates a stronger relationship between the predictors and the target. Before we calculate IV, we need to separate data into bins (i.e. categories or groups). The following formula is used to calculate the IV.

$$\mathrm{IV} = \sum\nolimits_{\mathrm{i}} (\%\mathrm{Good}_{\mathrm{i}} - \%\mathrm{Bad}_{\mathrm{i}}) \times \ln\left(\frac{\%\mathrm{Good}_{\mathrm{i}}}{\%\mathrm{Bad}_{\mathrm{i}}}\right) \tag{1}$$

where

$\%\mathrm{Good}_{\mathrm{i}}$ = number of good instances in ith bin out of total number of good instances
$\%\mathrm{bad}_{\mathrm{i}}$ = number of bad instances in ith bin out of total number of bad instances

The log component in the IV formula is the Weight of Evidence (WoE) which measure the strength of a grouping for separating good and bad instances.

$$\mathrm{WoE}_{\mathrm{i}} = \ln\left(\frac{\%\mathrm{Good}_{\mathrm{i}}}{\%\mathrm{Bad}_{\mathrm{i}}}\right) \tag{2}$$

The higher WoE value in absolute term, the stronger the separation between good and bad instances. WoE value can be interpreted as follows:

- WoE = 0: Good and bad instances distributed equally in a particular bin
- WoE < 0: Distribution of bad instances is greater than the distribution of good instances in a particular bin
- WoE > 0: Distribution of good instances is greater than the distribution bad instances in a particular bin.

Generally, the values of the IV statistic can be interpreted as in Table 1 [1]:

Table 1. IV interpretation

IV	Predictive Power
<0.02	Not useful
[0.02, 0.1)	Weak predictor
[0.1, 0.3)	Medium predictor
≥ 0.3	Strong predictor

2 Related Work

There are commonly used binning methods. We review eight well-known methods and categorize them into four groups. This first group is an unsupervised method i.e. it does not use the target variable's information to bin an input variable. On the other hand, the rest are supervised.

1. Unsupervised binning. The equal-width and equal-size binning [2] are in this group. For the equal-width, a continuous variable is divided into a user-defined number of equal-width intervals. Therefore, an equal-width binned output is obtained. For equal-size, the output has roughly the same number of observations. This group does not handle any desired constraints, nor does it aim to optimize any statistics.
2. Supervised, iterative merging. The optimal-binning [3] and Chi-Merge [4] methods belong to this group. These algorithms aim to find cut points for intervals. The number of output bins is not pre-defined. The number of output bins depends on a user-defined threshold. For the optimal-binning, it uses chi-square test's p-value. The other one uses the chi-square test statistic directly. The algorithms repeatedly merge pairs which contain two similar bins according to the statistics into a bin. Again, the group does not handle any desired constraints, nor does it aim to optimize any statistics.
3. Supervised, tree-based greedy search. This group has the multi-interval discretization [5], conditional inference tree-based binning [6], CAIM [7], CACC [8], and Ameva discretization [9] as examples. These algorithms aim to find cut points for intervals by adopting the greedy best first search algorithm. They start by finding a cut point on the whole range and creating two children nodes. Then, they continue to recursively split children nodes until a stopping criterion is reached. The first method uses entropy as heuristic. The second uses a two-sample permutation test statistic. The other three methods use Class-Attribute Interdependence Maximization (CAIM), Class-Attribute Contingency Coefficient (CACC), and Ameva coefficient respectively. This group does not directly aim to handle constraints nor optimize the output's statistics.
4. Supervised, monotone merging. There are two methods that could be categorized to this group which are the Maximum Likelihood monotonic coarse classifier (MLMCC) [2] and monotone optimal binning [10]. These algorithms aim to obtain

monotonic binned outputs. The main idea is that they split the input variable into many intervals and try to merge bins into a fewer number of monotonic bins. After obtaining monotonic bins, the algorithms iteratively merge adjacent bins based on a statistic. The cumulative bad rate is used for the MLMCC. For the latter, it uses the two-sample z-test's p-value statistic. In addition, the latter can handle the minimum observation in each bin constraint by forcefully merging violated bins to their adjacent. However, both methods do not directly aim to maximize output's statistics.

The algorithms incline to use supervised methods. The existing algorithms split a variable into intervals using a statistic (e.g., p-value or bad rate) to find cut points. Cut points separate two adjacent bins that are considerably different in term of the statistic used. The results from these algorithms are bins which have different characteristics based on their chosen statistics. Moreover, a main drawback of existing methods is that they ignore business knowledge. Business rules can be viewed as constraints and should be handled by binning algorithms. These algorithms do not facilitate adding new constraints. This leads us to implement a new method which we can easily choose output shape, constraints and a statistic to be optimized.

3 Implementation

3.1 Fine and Coarse Classing

In general, the binning process can start by firstly bin a continuous variable into many small bins. This is called fine classing. For example, a continuous variable from the HELOC (home equity line of credit) dataset from FICO [11], income can be binned into fine bins. Figure 1 shows WoE values of each bin. The bins with large WoE values contain more good instances than bad ones. The output in Fig. 1 is strange and difficult to explain why the bin [2424, 2435] is better than [1510, 2418].

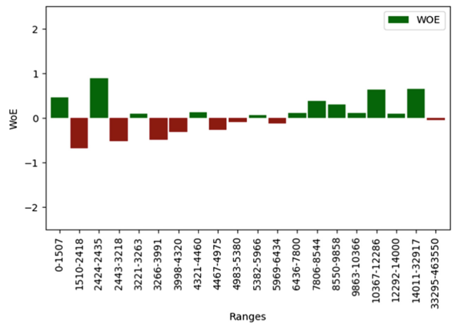

Fig. 1. Fine classing bins

A reason that could cause the fluctuation is insufficient samples in the bins. To stabilize the variable or to make it more explainable, practitioners may apply a domain knowledge that the larger the income, the higher chance of good instances. We use the knowledge to group the small adjacent bins together to obtain an output as in Fig. 2. This is called coarse classing. The result is more explainable. We can further use the binned output variable as an input variable for a classification problem. One way to do that is to simply use the WoE of each bin as a representative value.

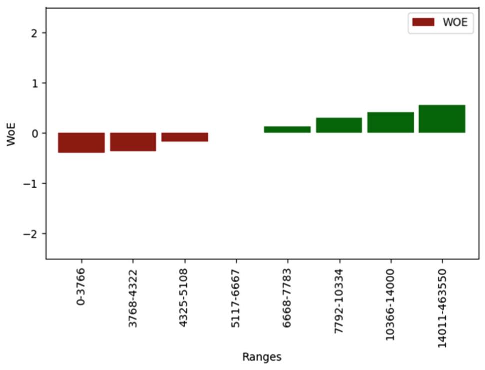

Fig. 2. Coarse classing bins

From the example above, binning can be viewed as a process to merge adjacent bins together or find cut points to obtain a good set of output bins. We use the criteria of good binning as described in the introduction section. The criteria are constraints. It is also a good idea to incorporate other constraints such as monotonicity, allowing only one local maximum, and minimum for output shape controlling. For a variable in the binning problem, there can be many output solutions that pass all the constraints. To decide which solution is the best one, we use IV as a heuristic. At this stage, we view the binning problem as a constrained optimization problem.

3.2 Search Space

Let us consider the search space of the problem. As an example, we assume the starting number of bins is four, the expanded search tree is shown in Fig. 3. The numbers 1, 2, 3 and 4 represent the nth bins. The root level shows that we start with four individual bins. The second level or the level that each node has three bins. The first node means

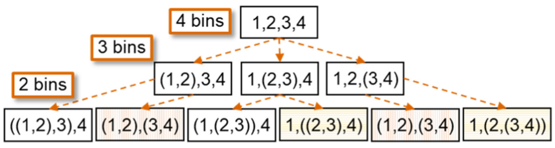

Fig. 3. Exhaustive search without pruning

we group the 1st and 2nd bins of the parent node together and leave the 3rd and 4th bins as they are. The bins in parentheses are in the same binned group. The number of leaf nodes in the expanded tree is factorial of N-1 when N is the starting number of bins. It is a huge number when N is big. Fortunately, at the last level or the 2-bins level, we can see that the second and fifth nodes are identical. This also happens to the fourth and sixth nodes. It means that we can prune the search tree to reduce the search space.

The problem can also be viewed this way. Starting with four separated bins, we group them into three bins by selecting a pair of adjacent bins which are (1, 2), (2, 3) and (3, 4) and grouping them together. The result is illustrated in Fig. 4. Therefore, the number of nodes generated in this level is 3, i.e. choosing a pair from three available pairs or $\binom{3}{1}$. The same approach can be applied to generate the two-bins nodes by choosing two from the starting three pairs and getting $\binom{3}{2}$ or 3 nodes. In total, with the starting bins of four, we generate seven nodes. As a result, if we have the starting bins of N, we generate $2^{N-1} - 1$ nodes. To summarize, the actual search space is an exponential function with base 2. Assuming we have an upper bound of the starting bins of 20 then the search space is 524,287 nodes. The 20 or N starting bins could be initially transformed from a continuous range into bins from the equal-size or equal-width binning algorithm. We could finish running it in a timely manner with a parallel brute force program with a commodity CPU. However, with the assumption, it is hardly believed that the solution has the best possible IV.

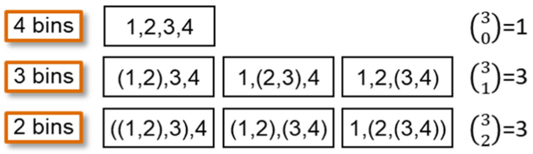

Fig. 4. Output nodes after pruning

3.3 Binning Algorithm Using Genetic Algorithm

This paper proposes a new binning algorithm using a genetic algorithm which we call "GAbin" to solve the constrained optimization problem. The success of using a genetic algorithm depends on two key components: (1) gene representation and (2) fitness function [12].

Gene Representation. We use cut points as the gene representation. With a binned output e.g., 1, (2, 3), 4, we have three output bins. The cut points are written as gene representation of 101. The gene representations are called individuals or chromosomes. Solutions produced from a genetic algorithm called individuals. An individual contains a series of 1s and 0s. A single bit is a gene. The "1" and "0" state cut and no cut points respectively. The "0" means we do nothing. The "1" means we group all prior consecutive 0s (if any) and itself together into a bin. In this case, we group bin 2 and bin 3 together. N bins have N-1 cut points because the last bin is always a cut point (end). We cut the four initial separated bins at the first and third positions. The below figures depict on how the gene representation is used in a genetic algorithm. In Fig. 5, we create a hypothetical continuous variable and initially bin it into 5 bins. We assume that it should be increasing monotonic. We start with five bins and its gene representation is 1111 (every bin is a cut point). The output contradicts the prior knowledge of monotonicity.

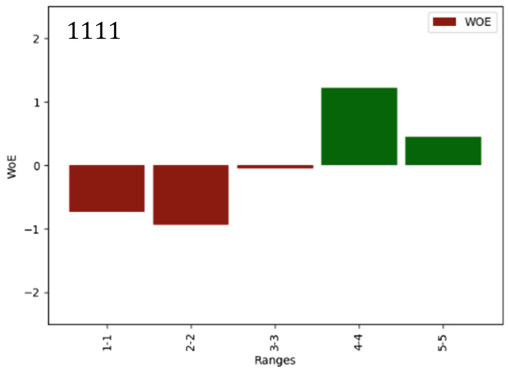

Fig. 5. Five bins of a hypothetical continuous variable

In Fig. 6, we bin the 1st and 2nd bins together and its gene representation is 0111 (the 1st bin is not a cut point). We could see that the output represents a good monotonicity in the range from one to four (the first three output bins).

In Fig. 7, we bin the 4th and 5th bins together and its gene representation is 1110. We could see that the output represents a good monotonicity except the first bin in the output. We could see that the two outputs are good at some bins. The 0111 and 1110 outputs are good for the bins on the left and right sides respectively. Combining them and producing new good outputs could be naturally done by the genetic algorithm's crossover operation. The crossover operation randomly picks two individuals at a time.

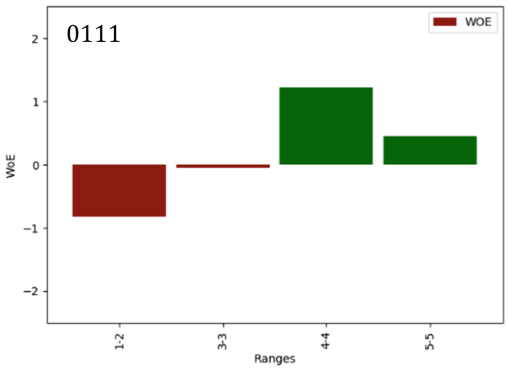

Fig. 6. 0111 gene representation

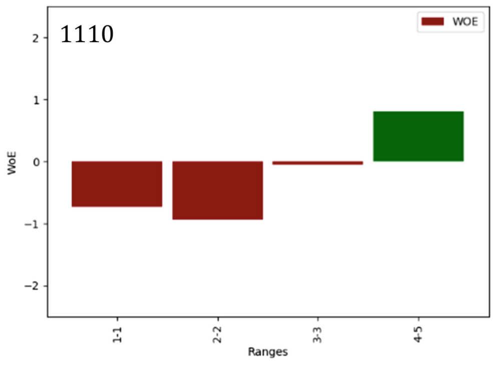

Fig. 7. 1110 gene representation

For each picked individual, cut it into two chunks at a random position (gene). Combine chunks from each individual to form a new individual (offspring). In this case, assume that we cut the first 0111 individual to 011 and 1. Cut the second 1110 and get 111 and 0. By combining the genetic codes (the first's 011 and the second's 0), we obtain 0110 and it is an increasing monotonic output as illustrated in Fig. 8. This shows that using cut points as a genetic representation in a binary-coded genetic algorithm could bin and control the shape of the binning problem. In our GAbin, an individual is randomly initialized to N-1 binary digits where N is the number of starting bins. The maximum number of 1s in an individual is 19 since it is said that in every output bin should contain at least five percent of samples. Therefore, the maximum number of output bins is 20.

Starting Bins. To bin a continuous variable, it is suggested that we bin its continuous range into starting bins using the equal-size or equal-width binning algorithm. However, deciding on how many starting bins is problematic. The output of the binning process by combining bins using cut points depends on starting bins. We could not change the so-called cut points inside the starting bins. This leads us to use unique

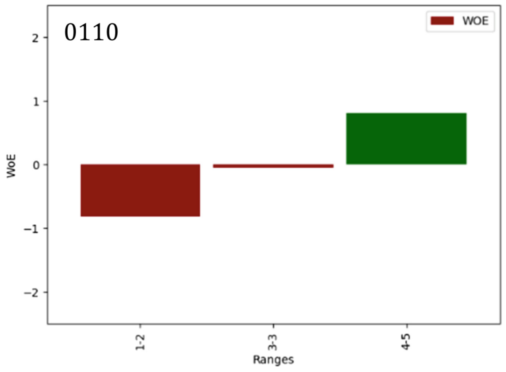

Fig. 8. 0110 gene representation

values of the continuous value as our starting bins. For example, for the FICO dataset, the age values are from 19 to 99. There are 81 unique values and they are our starting bins. This gives the GAbin full control over selecting cut points.

Fitness Function. GAbin's objective is to maximize IV subject to given constraints. The guidelines suggest that each bin should contain at least 5% percent observations. This could be treated as a constraint. We also add other constraints such as (1) each bin should contain at most two times of average samples in each bin to avoid having too many samples in a bin, (2) output's shape controlling constraints (monotonicity, one or two local maxima). The constraints in GAbin are optional. Users can select which constraints to use. Moreover, business rules can be also viewed as constraints and easily incorporated to GAbin. To summarize, our GAbin use IV as the objective function subject to two main groups of constraints which are (1) number of samples in each bin or constraint group 1, and (2) shape controlling constraints or constraint group 2.

The GAbin uses the static penalty approach to handle constraints in the genetic algorithm. This means the fitness function is independent of the current generation number [13].

This is an example of a binning option. With a continuous variable, the constraints are (1) each bin contains at least 5%, (2) at most two times of average samples in each bin, and (3) monotonicity.

We have three main penalty types:

1. Death penalty. We give a death penalty (−10,000) to an individual with more than 19 cut points to avoid too many useless output bins after crossover.
2. Constraint group 1's penalty. The penalties for the first and second constraints are how many samples are more or less than the acceptable threshold. Let us consider an example. A variable has a total number of samples of 400. The 5% threshold is 20 samples. A small bin has only five samples then the penalty is 15.
3. Constraint group 2's penalty. The penalty for the shape controlling constraint is the total number of samples in the bins that violate the constraint. When we try to control the monotonicity, there is a bin of five samples that violates the monotonicity while other bins are satisfied. The penalty for this case is 5.

A bin cannot violate the first and second constraints at the same time. However, a bin can violate both the number of samples in each bin and shape controlling constraints. We use a double-counting approach. For example, a small bin of five samples violates the first and third constraints. This means it has too few samples and breaks the monotonicity. With the 5% threshold of 20, the total penalty for this bin is 15 + 5 = 20. The mentioned penalties reflect two key properties of a good constraint handling penalty function [13]. The first property is to represent the number of violated constraints. This demonstrates in the double-counting approach. The penalty function counts the violated samples for each constraint. The second property is to represent the distance to feasibility. This is the reason we count how far away from the acceptable threshold. Therefore, we create the fitness function for GAbin as Eq. 3. In addition, to control the output shape of one local maximum or minimum (excluding the first and last bins), we use a similar penalty taking care of the monotonicity. If the constraint is violated, the penalty function counts numbers of samples in each violated bin.

$$\text{fitness}(X) = IV - w_1 \sum_{j=1}^{c} \sum_{i=1}^{k} N_{1i} - w_2 \sum_{i=1}^{k} N_{2i} \tag{3}$$

where

IV	Information Value described in Eq. 1.
w_1	penalty coefficient of constraint group 1 (default: 1)
w_2	penalty coefficient of constraint group 2 (default: 2)
c	number of group 1 constraints
k	number of output bins
N_{1i}	samples that violate the acceptable threshold for the jth group 1 constraint in ith bin
N_{2i}	samples in ith bin that violate the constraint group 2

Algorithm. We summarize the algorithm for the GAbin in Algorithm 1. The default values of hyper parameter are from experiments.

Algorithm 1 GAbin: Binning Algorithm to Maximize Information Value using genetic Algorithm

Inputs:

1. A continuous variable with a binary target variable $D = \{(x_i, y_i)\}_{i=1}^{N}$
2. f(x), A fitness function according to Equation 3.
3. $|P|$, size of the population in each generation (default: 3,000)
4. PB_{CX}, the probability with which two individuals are crossed (default: 0.5)
5. PB_{mu}, the probability for mutating an individual (default: 0.1)
6. PB_{i_mu}, the probability for mutating a gene (default: 0.05)
7. $|T|$, tournament size (default: 3)
8. G_{ND}, maximum number of generations with no development (default: 20)
9. G_{max}, maximum number of generations (default: 300)

Initialization

Initialize:

1. g ← 0 #generation number
2. Count unique values for x_i and its associated numbers of good and bad samples. These unique values become the starting input bins.
3. Create an initial population, P_g. The individuals in the population are from a generator that randomly initialized to N-1 binary digits where N is the number of starting bins. The maximum number of 1s in an individual is 19.

Evaluate:

4. Evaluate the entire population using f(x).

```
Main loop
  while g < $G_{max}$ and the no f(x) develops < $G_{ND}$ do
    1. Select: individuals in $P_g$ to offspring set, $S_g$ based on its f(x) and the $|T|$.
    2. Crossover: Randomly select pairs two individuals in $S_g$ with the probability of
       $PB_{CX}$ and crossover them. This is a processed $S_g$ called $S_g'$.
    3. Mutate: individuals in $S_g'$ with the probability of $PB_{mu}$ and $PB_{i\_mu}$. This is a
       processed $S_g'$ called $S_g''$.
    4. Evaluate: Evaluate the entire $S_g''$ using f(x).
    5. Update: Replace $S_g''$ to $P_{g+1}$, g ← g + 1 and count the no development genera-
       tions.
  end while
  return individual with the largest f(x) in $P_g$.
```

The time complexity for a genetic algorithm could be considered by numbers of fitness function evaluations. It is $O(|P| \times G_{max})$. For GAbin, it is 3,000 × 300 or 900,000 regardless the number of starting bins.

4 Experiment and Results

For demonstration purpose, we implemented the GAbin and used it to bin the HELOC (home equity line of credit) dataset from FICO [11]. It was a binary classification problem. The target variable stated a defaulted or non-defaulted loan. We binned a continuous dependent variable, LTV (Loan to value ratio). We believed that LTV could have a decreasing monotonic relationship with the default. Larger LTV means more money to repay. We first split the dataset into training and testing datasets. We used the training dataset to bin the variable using three binning options:

1. Each bin contains at least 5% and at most two times of average samples in each bin. Furthermore, the adjacent bins must have minimum WoE difference of 5 percent of the maximum WoE. (Basic constraints).
2. The basic constraints plus monotonic (increasing or decreasing).
3. The basic constraints plus allowing only two local maxima (excluding the first and last bins).

After binned the training data, we applied the binned ranges to the test data and calculate its IV. The summarized result is shown in Table 2. As expected, output with

Table 2. Binning result for LTV

Option	IV in training	IV in testing	Output's bins
1	0.2346	0.1971	8
2	0.2183	0.1551	5
3	0.2249	0.1621	10

only basic constraints had the highest IV. It was because the genetic algorithm could find any solution to maximize IV with fewer restrictions. The same reason is applied to the other options. Option 2 had tighter restrictions than option 3.

We placed outputs for each option in training and testing datasets in Figs. 9, 10 and 11. Figure 9 had the loosest constraints, so the shape in test data was different from train data. This could mean unstable output bins. Figure 10 shapes were stable. This could mean that the monotonic assumption suits LTV well. Figure 11 shapes had roughly one minimum and one maximum point or two peaks in both train and test data. This demonstrated that GAbin could produce user-desired output shapes.

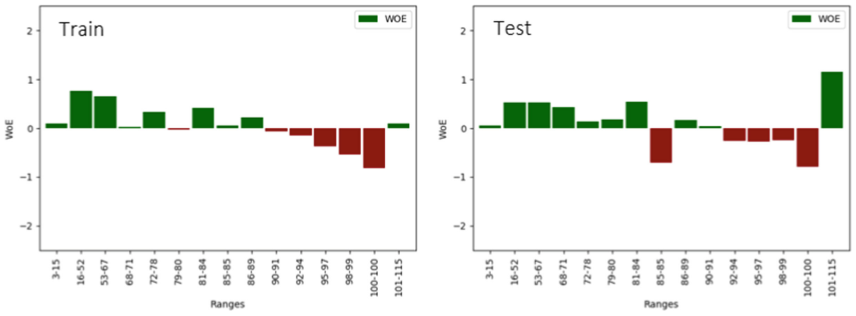

Fig. 9. Option1 result for train and test datasets

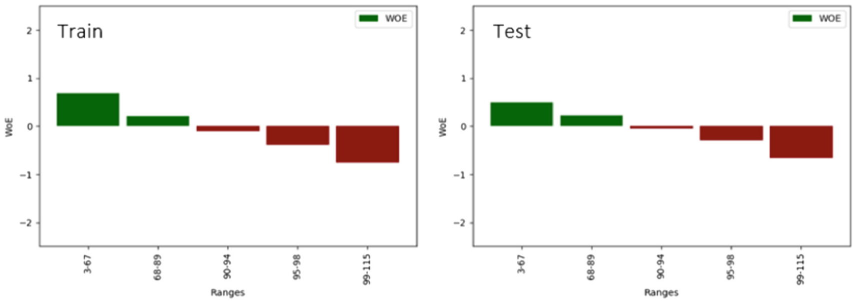

Fig. 10. Option2 result for train and test datasets

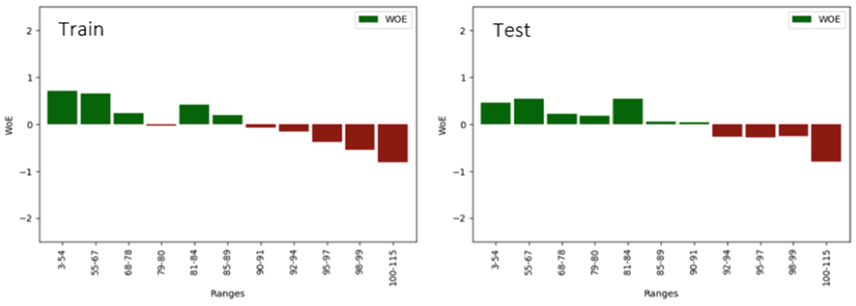

Fig. 11. Option3 result for train and test datasets

Comparing results with other binning algorithms mentioned in the related work section are shown in Fig. 12. The GAbin with only basic constraints achieved the best IV in both train and test set. It showed that the GAbin can truly optimize the target function, IV. However, looking at only IV can be misleading. The result shape should be explainable and non-fluctuating. Therefore, it was dubious to use the first option in real-world applications. Its output shape could not sustain in the test set. This was also the case for MLMCC. The method aimed and was able to obtain monotonic bins in the train set but not in the test. The GAbin with 2 peaks (option 3) achieved high IV in train and test set. In additional to that, the output shape in the test set was roughly similar. Also note that the outputs for GAbin monotonic and monotone optimal binning are identical for this variable.

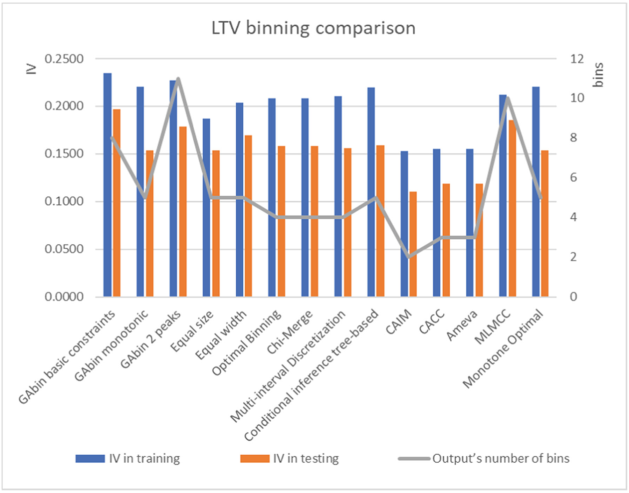

Fig. 12. LTV binning comparison

Another example to show that output shape controlling was useful. We binned the prior custom score variable and compared with other methods. The comparison is showed in the Fig. 13. The Chi-Merge had the highest IVs in train and test set, but the output shapes were different and fluctuating. While others' outputs are monotonic, we found that allowing the variable to have a peak achieved higher IVs. The output bins from GAbin were shown in Fig. 14. It showed that the output shape sustains in the test dataset. This led us to believe that the prior custom score variable is more suited the one-peak shape than monotonic.

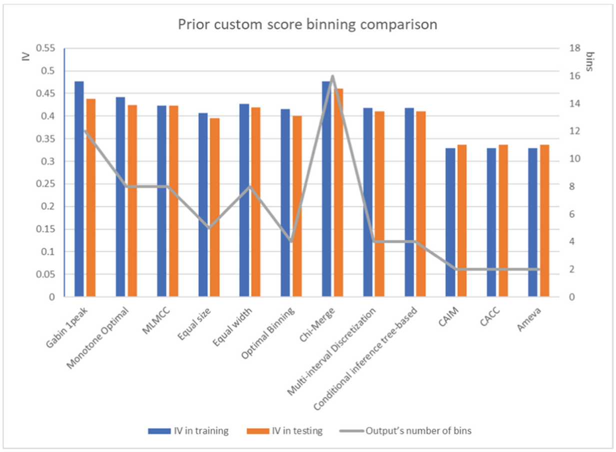

Fig. 13. Prior custom score comparison

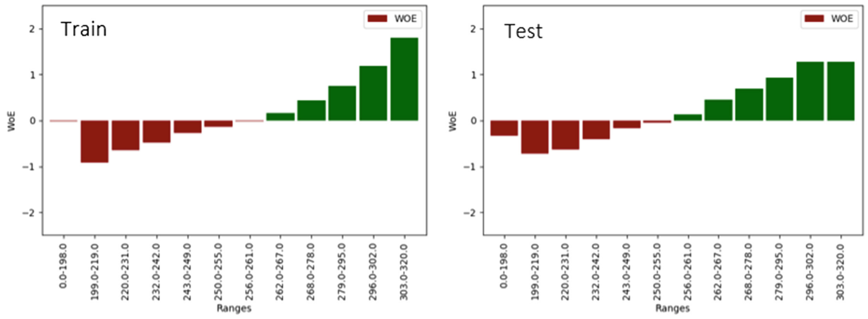

Fig. 14. GAbin's prior custom score in train and test datasets

5 Conclusion

GAbin is a supervised binning algorithm. Three main advantages of the GAbin are (1) users obtain IV-optimized results for classification problem, (2) users can choose the output shapes, and (3) all constraints are handled. Experiment results show that output bins can achieve high IVs, satisfy constraints and produce good results in test data. Using a genetic algorithm in GAbin demonstrates its powerfulness. GAbin can constrained-optimize IV while it does not require any number of starting bins, output bins or predefined statistic thresholds. However, it does require hyper parameters for the genetic algorithm. Tuning hyper parameters is possibly an area for improvement. At this stage, it can be used for only a binary classification because of its fitness function, IV. GAbin's structure can support other supervised learning by changing its fitness function to a suitable one.

Acknowledgments. This research was partially supported by Taskworld Inc.

References

1. Siddiqi, N.: Credit Risk Scorecards, pp. 79–82. Wiley, Hoboken (2013)
2. Thomas, L., Edelman, D., Crook, J.: Credit scoring and its applications, pp. 131–139. SIAM, Society for industrial and applied mathematics, Philadelphia (2002)
3. Refaat, M.: Credit Risk Scorecards: Development and Implementation Using SAS. Lulu.com, Raleigh (2011)
4. Kerber, R.: ChiMerge: discretization of numeric attributes. In: The Tenth National Conference on Artificial Intelligence, San Jose, California (1992)
5. Fayyad, U.M., Irani, K.B.: Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In: IJCAI (1993)
6. Jopia, H.: Scoring Modeling and Optimal Binning. (2019). https://cran.r-project.org/web/packages/smbinning/smbinning.pdf. Accessed April 2019
7. Kurgan, L., Cios, K.: CAIM discretization algorithm. IEEE Trans. Knowl. Data Eng. **16**(2), 145–153 (2004)
8. Tsai, C., Lee, C., Yang, W.: A discretization algorithm based on class-attribute. Inf. Sci. **178** (3), 714–731 (2008)
9. Gonzalez-Abril, L., Cuberos, F., Velasco, F., Ortega, J.: Ameva: an autonomous discretization algorithm. Expert Syst. Appl. **36**(3), 5327–5332 (2009)
10. Mironchyk, P., Tchistiakov, V.: Monotone optimal binning algorithm for credit risk modeling. Researchgate (2017). https://www.researchgate.net/publication/322520135_Monotone_optimal_binning_algorithm_for_credit_risk_modeling. Accessed April 2019
11. FICO: Home Equity Line of Credit (HELOC) Dataset. FICO. https://community.fico.com/s/explainable-machine-learning-challenge?tabset-3158a=2. Accessed April 2019
12. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn, pp. 126–129. Prentice Hall, Upper Saddle River (2010)
13. Coello, C.A.C.: Constraint-handling Techniques used with evolutionary algorithms. In: The Genetic and Evolutionary Computation Conference Companion, Kyoto, Japan (2018)