# Automatic WoE Re-binning Algorithms

Eric Liu

For many reasons, an automatic woe re-binning tool is needed in our daily job. It can save the modeler lots of time when there are hundreds or thousands of variables. It can find an optimal solution given certain constraints, which is quite difficult for our human brains. With its help, we can divide a variable into much more bins than ever before, in order to get higher IV and more accurate predictors.

## Problem Definition

First we’ll introduce some notations that will be used in this document. And we’ll define the re-binning problem with these notations.

* : dataset. We usually have 2 datasets: for DEV, for OOT.
* : variable. Each variable has some bins over all the datasets. We can denote a variable as . The number of bins and the bin boundaries are decided by other criteria which will not be covered in this document.
* : bin. The minimum unit of a variable in the whole re-binning process. Each bin has its high boundary and low boundary . And these boundaries are applied on each dataset. For each dataset, a bin has its own attributes related to the re-binning process including:

1. Bad percentage () :
2. Good percentage () :
3. Information value () :
4. Weight of evidence () :

Given these definitions, the re-binning problem can be defined as:

Given two datasets and , and a variable that exists on both datasets, try to find a series of combination of bins so that in all datasets, either

or

holds (monotony constraint), where , and (boundary constraint), and all of are included in (integrity constraint). And the function of reaches the possible maximum (information value constraint).

## Problem Abstraction

First I want to introduce the famous knapsack problem:

*Given the max volume of a bag to fill in, and a pile of goods with different volumes and values to choose from, what’s the solution to have the most valuable bag of goods?*

In our circumstance, the re-binning problem for each single variable can be treated as an abstract knapsack problem with item-dependency constraint:

1. The number of the unmerged bins is the volume of our abstract bag.
2. All possible combination of bins are the goods we can choose from, with the volume of , and the value of .
3. The item-dependency constraint refers to that if the combination is chosen, all other combinations involving cannot be chosen any more.
4. And the monotony constraint, boundary constraint and integrity constraint mentioned before also hold.
5. The target is to fill the abstract bag with the maximum of

## The First Algorithm

Dynamic programming and search tree are commonly used in knapsack problems. But in our special case, search tree is a better method because pruning can be done effectively considering the monotony constraint.

The search tree is built as the following figure:



The algorithm is to do a back order walk through the whole tree from the root node, and do pruning whenever it is possible.

The visiting sequence of the back order walker is:

1. Visit the most left child node.
2. Visit the sibling child node.
3. Repeat step 2 if there are un-visited child nodes.
4. Visit the current node.
5. Return to the parent node, if there is one.

The actions taken when visiting a node includes:

1. Get the next child node. If there is one, go to next step, otherwise, go to step 3.
2. Calculate the of the child node, if the monotony constraint cannot hold, prune this child node, then go to step 1. Otherwise, save the for reducing future computation and go to step 4.
3. If the current node includes the last bin , calculate the . If it is larger than the current max, update the best solution to current one.
4. Visit the next node. It is the child or parent node of current node.

The algorithm starts from a virtual bin , and when the back order walker returns to , the best solution has been found.

## Convex Attribute of Information Value Function

First we denote the action of merging two adjacent bins and into a larger bin as

After 10 billion times of random experiment, we have a hypothesis of convex of function:

when and only when , the equality holds.

## Algorithm Optimization

The search tree algorithm is simple in terms of idea but very complex in terms of computation. The major reason to this is when finding the possible child nodes for , , all of the , will be considered. This results into pairs of possible combination.

Figure below illustrates the idea of when , , and will all check whether there is any solution existing starts with each of , if the monotony constraint holds for all of the combination of and .



Here the convex attribute of information value function comes to rescue. We can reduce the combination at each layer from to .

In the ideal case, if we have , , >, which matches the given trend, and , , and , then we can directly prune the nodes of , , and.

Of course the later constraint won’t hold in the most cases, so that we cannot prune any nodes directly. But we still can prune the combination of (parent, child) nodes pairs, in other words, the corresponding link in the tree. In fact, we alter our algorithm to a tree building algorithm.

1. For each of the child nodes belong to layer , which refers to , , we calculate its and its .
2. For each child node, we find its eligible parent nodes , that qualify the monotony constraint.
3. Because each has only one path to the root note, with the total IV of , we append to the path with the max .
4. Go to next level, and restart from step 1.

The algorithm also starts with a virtual bin , which belongs to layer 0. And it ends with paths with different total IV of , . The one with the maximum should be our answer.

## Algorithm Analysis

In the average case, there are times of merging computation, times of computation when building up the tree, and in the final solution comparison.

In practice, the current implemented version is the search tree algorithm with a minor optimization using the convex attribute. It takes less than 3 minutes to compute the WOE bins for 1000+ variables. Each variable could have less than 100 bins before the re-bin process, in each bin there are 2% of the good ones or the bad ones.

## Alternative Greedy Algorithm

Before designing the algorithm above, a simpler greedy algorithm was designed and implemented with complexity, as a compromise, the maximum of IV cannot be achieved.

The algorithm can be simply put in the following way:

1. Starts from DEV dataset.
2. From the first bin, look for the first pair of anti-trend bins.
3. If found, merge these 2 bins. If not, move to the next dataset. Repeat step 2.

## Data Preparation for Categorical Variables

We cannot apply the monotony constraint to categorical variables naturally. So a sorting algorithm must be applied to these variables before the re-binning process.

What is usually done includes:

1. Sorting by the of the DEV dataset.
2. Sorting by the of the combined dataset of DEV and OOT.