Monotone optimal binning algorithm for credit risk modeling

Pavel Mironchyk
Rabobank
pavel.mironchyk@rabobank.com

Viktor Tchistiakov
Rabobank
viktor.tchistiakov@rabobank.com

September 6, 2017

Abstract

Overview of commonly used algorithms for credit score binning is given. A new binning algorithm especially suitable for credit Scorecard modeling and showing superior performance is proposed. The performance of the proposed algorithm and commonly-used algorithms is compared on open-source dataset HELOC.

I. Introduction

Binning or bucketing is a technique purposed to reduce impact of statistical noise. An interval with all observed values is split in smaller sub-intervals, bins or groups, each of them gets assigned the central value characterizing this interval. All observation that lay on particular sub-interval form an associated bin. Obviously central estimates depend on how you choose split points. From other side, the binning is also a form of discretization, a way to cut a continuous value range into a finite number of sub-ranges, each getting some categorical value associated. In such way a continuous value could be replaced by discrete categorical value - to reduce overall complexity. Binning is widely used in credit risk modeling. For instance, it is essential for credit scorecard modeling to separate low and high risk observations (Mays, 2001). It is used to segment a risk driver values to statistically coherent groups, so central tendencies of those could be used to train models and thus reducing influence of statistical noise or outliers. There is a number of algorithms for binning used in credit risk modeling, as descrived in literature. The most popular methods for binning include the following: equal-width binning, equal-size binning, optimal binning (Zeng, 2014), multi-interval discretization (Fayyad and Irani, 1993), MLMCC (Thomas, 2002), Chi-merge (Kerber, 1992), conditional inference trees binning (Hothorn and Hornik, 1999). Further in this article we discuss each of these algorithms in more details. In this paper, the authors propose new simple and fast heuristic algorithm for binning that is specially designed for problem of scorecard modeling - monotone optimal binning. The authors performed companion of binning algorithm against widely used measures and found that the proposed algorithm satisfies better typical requirements for binning. This paper is organized as follows. *Section 2* describes briefly commonly used binning algorithms. *Section 3* defines criteria for good binning algorithm. *Section 4* describes proposed monotone binning algorithm. *Section 5* reveals performance comparison of algorithm on criteria defined in *Section 3*. In the last section, the conclusions are presented. The appendix contains full reference implementation in MATLAB language of proposed algorithm.

II. Commonly-used binning methods

In this section we give a brief description for each method mentioned in Introduction.

i. Equal-width

This is an obvious and straightforward approach to binning (Thomas, 2002). The whole range of predictor values is divided into a pre-specified number of equal-width intervals. Each interval defines borders for corresponding bin. Thee number of bins is predefined.

ii. Equal-size

Here, the range of predictor values is split in intervals in such way, that bins formed in result contain more or less equal number of observations. The width of bins varies depending on density of observations (Thomas, 2002). The target number of bins is predefined. Both equal-width and equal-size binning algorithms do not employ relationship with the target variable.

iii. Optimal Binning

This is algorithm is an evolution of the previous two algorithms. It may be seen as optimization on top of the previous two, as in this case a predictor variable is used to define cut-points for intervals. The goal of algorithm is to define bins have sufficiently different statistical mean estimates of predictor value (Siddiqi, 2006). The algorithm consists of the following steps:

1. We start with sufficiently big number of small buckets
2. For each neighboring pair of buckets we computes p-value
3. Find the largest p-value of all pairs; if it is above some threshold merge corresponding pair of buckets then repeat step from 1, otherwise exit.

iv. Maximum-likelihood Monotone Coarse Classifier

This is a well known algorithm (Thomas, 2002), it is also also known under name "Monotone Adjacent Pooling Algorithm". It is implemented in MATLAB Financial Toolbox (2014b) as default algorithm for credit scorecards predictors binning.

Assume that bad rate is going down as characteristic value increases. (If not, apply the following procedure starting from the other extreme of characteristic). Start at the lowest characteristic value and keep adding values until the cumulative bad rate hits its maximum. This is the first coarse classification split point. Start calculating the cumulative bad rate from this point until it again hits maximum. This is the second split point. Repeat the process unlit all the split points are obtained.

v. Multi-interval Discretization

Multi-interval discretization binning is based on entropy minimization heuristic search for recursively splitting of continuous range into sub-intervals, and recursively define the best bins (Fayyad and Irani, 1993). This is method purposed to separate classes based on observation frequencies. The entropy function is defined as

$$Ent(S) = -\sum_{j=1}^{k} P(C_i, S) log(P(C_i, S))$$

where C_i are predictor classes in input dataset S. We aim at maximizing information entropy of the partition induced by T : $S_1 \in S$, $S_2 = S - S_1$,

$$E(T, S) = \frac{|S_1|}{|S|} Ent(S_1) + \frac{|S_2|}{|S|} Ent(S_2)$$

Once cut point T is found for compete interval of S, the process is repeated for sub-intervals recursively and so on, until there is no substantial improvement in entropy.

vi. Chi-Merge

The algorithm is somewhat similar to optimal binning merging (Kerber, 1992). The difference that χ^2 is used to test similarity of adjacent

bins. The algorithms consists of the following steps:

1. First the input range is initialized by splitting it into sub-intervals with each sample getting own interval.
2. For every pair of adjacent sub-intervals a χ^2 value is computed.
3. Merge pair with lowest χ^2 into single bin.
4. Repeat 1 and 2 until maximum χ^2 is less then some predefined threshold.

χ^2 is defined as following:

$$\chi^2 = \sum_{i=1}^{m}\sum_{j=1}^{k}\frac{(A_{ij} - E_{ij})^2}{E_{ij}}$$

where $m = 2$ (the 2 intervals being compared), k= number of classes = number of observations in ith interval, jth class. A_{ij} = number of examples in i-th interval $= \sum_{j=1}^{k} A_{ij}$, R_i = number of examples in j-th class $= \sum_{i=1}^{m} A_{ij}$, N = total number of examples $= \sum_{j=1}^{k} C_j$, E_{ij} = expected frequency of $A_{ij} = \frac{R_i C_j}{N}$.

vii. Conditional Inference Trees

Here we use multidimensional formulation of algorithm as it was given in (Molnar, 2013). The algorithm is based on exhaustive search of partition scheme that would maximize some test statistics defined as following, in generalized form as stated by (Hothorn and Hornik, 1999):

$$\mathbf{T}_j(L_n, \boldsymbol{\omega}) = vec\Big(\sum_{i=1}^{n}\omega_i g_j(X_{ij}) h(Y_i, (Y_1), ..., Y_n))^T\Big)$$

Here we are testing independence of response Y and covariate $\mathbf{X_i}$, j is a index of variable in m-dimensional X. L_n- length of a learning sample, $\boldsymbol{\omega}$ - weight, influence vector function h and transformation vector function g. The resulting vector statistics is mapped to scalar form using the following formula:

$$c(\mathbf{t}, \mu, \Sigma) = \max_{k=1,...,pq}\Big|\frac{(\mathbf{t} - \mu)_k}{\sqrt{(\Sigma)_{kk}}}\Big|$$

where t is an actual value of the test statistics T, μ is expected value of T under independence and Σ_{kk} is the k-th diagonal entry of the covariance matrix. The value of this scalar used to asses independence of X and Y. The algorithm consists of the following steps:

1. Stop criterion. Test global null hypothesis H_0 of independence between Y and all covariates X_j with $H_0 = \cap_{j=1}^{m} H_0^j$ and $D(Y|X_j) = D(Y)$ (permutation test for each covariate X_j). If H_0 not rejected (no signicance for all X_j) => stop.
2. Variable selection. Select covariate X_{j*} with strongest association (smallest p-value).
3. Best split point search. Search best split for X_{j*} (max. test statistics) and partition data
4. Repeat/Recurse. Repeat steps 1), 2), 3) for both of the new partitions.

III. Criteria of a Good binning

A good binning algorithm as viewed though "prism" of credit risk modeling should follow the following guideline (Thomas, 2002):

1. Missing values are binned separetely
2. Each bin should contain at least 5% percent observations
3. No bins should have 0 accounts for good or bad

First condition is a very technical implementation detail and can be address in any method rather easily. However other two conditions need to embedded into binning algorithm. None of the algorithm we reviewed above explicitly addresses combination conditions 2 and 3 while doing a decent good/bad

separation. Next we define a measure of separation. WoE is widely used in credit scoring (Mays, 2001) to separate good accounts from bad accounts of risk on a specific group:

$$WoE_i = log\left(\frac{G_i}{\sum_{i=1}^{n} G_i} / \frac{B_i}{\sum_{i=1}^{n} B_i}\right)$$

Here G_i is number of goods in bin i, B_i is number of bads in bin i. Then Information Value IV of a predictor is a weighted to sum of all WoE over all bins. It expresses the amount of information of the predictor in separating Goods from Bads in the target variable:

$$IV = \sum_{i=1}^{n}\left[WoE_i(\frac{G_i}{\sum_{i=1}^{n} G_i} - \frac{B_i}{\sum_{i=1}^{n} B_i})\right]$$

According to (Siddiqi, 2006), by the rule of thumb, the values of the IV statistic can be interpreted as follows. If the IV statistic is:

- Less than 0.02, then the predictor is not useful for modeling (separating the Goods from the Bads)
- 0.02 to 0.1, then the predictor has only a weak relationship to the Goods/Bads odds ratio
- 0.1 to 0.3, then the predictor has a medium strength relationship to the Goods/Bads odds ratio
- 0.3 or higher, then the predictor has a strong relationship to the Goods/Bads odds ratio.

IV reaches maximum if binning is nearly optimal. Author use IV as measure to decide whether one binning scheme is better then other's.

Another important metric we use to asses quality of binning is Herfindahl-Hirschman Index:

$$HHI = n\sum_{j=1}^{n}(\frac{N_i}{N_{total}})^2$$

Here N_i is number of sample in bin i. The lower HHI is, the dataset is more uniformly distributed in resulting bins. A large HHI is clear sign of deficiency of binning scheme.

IV. Monotone Optimal Binning

In credit risk we have some field specific requirements to a binning algorithm. Naturally we are able derive the following three requirements:

1. Monotonicity. It is expected from binning algorithm to divide an input dataset on bins in such way that if you walk from one bin to another in the same direction there is a monotonic change of credit risk indicator. This arises from basic assumption that a predictor variable is capable to separate higher risks from lower risks in case of the global non-monotonous relationship.

2. Representativeness. At the same time it is expected from algorithm to set up the bins in a way that they should reflect maximized correlation between the predicting and risk indicator. Then results of binning may be used to construct and train regression model, or they can be used as it is, to assign scores.

3. Constraints. Finally it is expected from algorithm to produce number of bins within particular constraints, see guideline for good binning in the previous section.

MLMCC algorithm is built around the the first requirement of monotonicity. It will be shown in the next section, that it will construct bins with strictly monotonic WoE . However it also creates bins that are not statistically important with too few samples, which can be rather viewed as outliers or statistical noise. This would always happen on real life data that always contains clusters of samples, sparse areas and abnormal samples.

Algorithms like Chi-Merge and Optimal binning are designed around the second requirement - the maximum correlation between predictor and indicator, they aim at constructing

bins that are coherent and separated on variance or point estimate (p-value). Algorithms like Maximum-interval discretization and conditional inference tree fall into the same category, they use information entropy and condition inference to construct more coherent bins.

Simple algorithms like equal-size or equal-width have advantage of producing constrained number of bins, the third requirement for perfect binning algorithms established in Section III. Maximum-interval discretization and conditional inference trees algorithms are very difficult on constraining number of bins they produce due to highly non-linear relation between parameters and number of bins.

The method we propose, was designed to address all the three requirements from section section 3 - monotonicity, coherent bins maximizing predictor-indicator relation, upper bound on number of bins. We did it by combining benefits of optimal binning on p-value and monotone ML coarse classifier algorithms in a single algorithm. It is also utilizes monotonic relationship between driver and indicator variable (good rate) and consists of the following steps, assuming that indicator variable (for example bad rate) is monotonically increasing with the value of driver:

1. Sort data by driver value, divide input range in smallest possible bins, i.e. each sample gets own sub-interval.
2. (Monotonicity) starting from top bin, walk down till last bin, stop and merge if rate of bin is bigger than the rate of consecutive bin, then restart from top. If no merge occur go to the next step.
3. (Optimality by p-value) for every neighboring pair of bins compute p-value.
4. (Filtering by bin size) for each pair of bins if any of two bins has size lower than some predefined threshold, modify the corresponding p-value increasing it by one..
5. (Filtering by lowest rate) for each pair of bins if any of two bins has indicator variable rate lower than some predefined threshold, modify the corresponding p-value increasing it by one.
6. Find the largest modified p-value, if all p-values are smaller than the specified threshold, then exit from algorithm.
7. Otherwise merge two bins corresponding to the largest modified p-value
8. Go to step 3.

Note that there are user specified input parameters to algorithm - p-threshold, size-threshold and rate-threshold. Changing the level of p-value threshold allows to tune the maximum size of bins, as higher the value of the threshold as more bins are merged together resulting in bins with higher number of observations. See appendix for reference implementation of algorithm in MATLAB language.

V. Comparison

Our aim is to validate Monotone Optimal binning algorithm and compare with other commonly used algorithm on a real-life reference dataset. We use publicly available HELOC(home equity line of credit) training data of FICO company (FICO, 2016). We use two columns - LTV (Loan to value ratio) as a risk driver and the default indicator. We limit dataset to cases of repaid and defaulted loans. The following is performed:

1. The general binning test of Equal-Size 5, Equal-Width 6, Optima 7l, MLMCC 8, Multi-interval discretization 9, ChiMerge10, Conditional Inference 11 and our Monotone Optimal 12 algorithms on HELOC LTV-DefaultIndicator dataset. For each binning scheme IV and HHI indicators were computed.
2. Noise stability test which resembles out-of-time validation test. Here we take original dataset, randomize 10% of default indicator value in one scenario, then in another attempt 30% of default indicator is randomized. All algorithms are applied to

these two datasets. For each test IV, HHI values were recorded. See 1 and 2.

3. Out-of-sample test. Original HELOC set is divided on in-sample and out-of-sample subsets in proportion approximately 80%/20%. All algorithms are applied to create binning scheme with in-sample tests. Then out-of-sample IV and HHI are computed on corresponding binnings schemes. See 4.

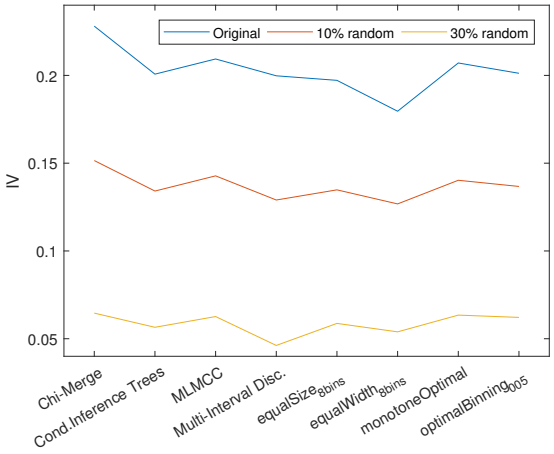

Figure 1: *IV on HELOC dataset*

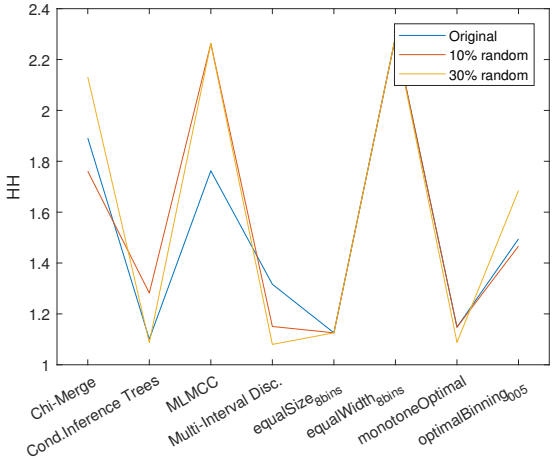

Figure 2: *HH on HELOC dataset*

The table 1] points at specific implementations of algorithms and parameters that were used in experiments.

VI. Conclusion

As it is shown the different method show very different performance.

Optimal binning algorithm is designed to satisfy need of credit scorecard risk modeling. As it is shown from comparison testing above it has satisfying performance, and it has not reveal overfiting tendencies. The proposed method is a hybrid method allowing for fine tuning statistically and constrained optimization.

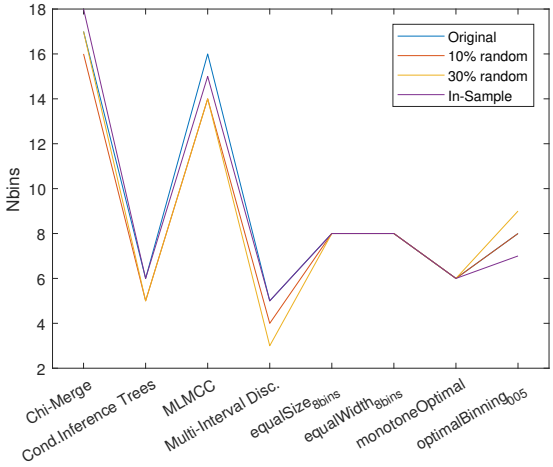

Figure 3: *nBins*

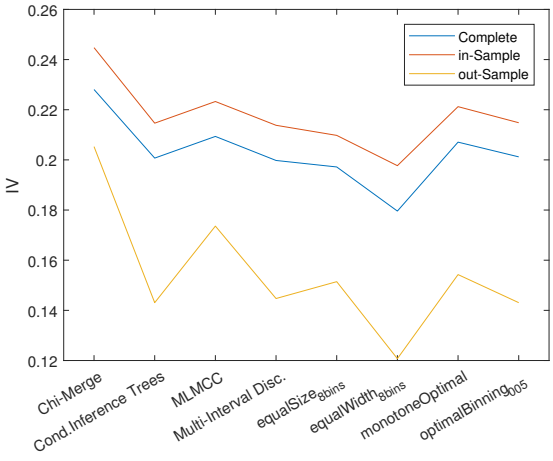

Figure 4: *IV on out of sample HELOC*

References

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. *Artificial intelligence*, 13:1022–1027.

FICO (2016). Heloc training datase.

Hothorn, T. and Hornik, K. (1999). party: A laboratory for recursive partytioning.

Kerber, R. (1992). Chimerge : Discretization of numeric attributes. *In Proceedings of the Tenth National Conference on Artificial Intelligence*, 10:123–128.

Mays, E. (2001). *Handbook of Credit Scoring*. Glanelake Publishing Company.

Molnar, C. (2013). Recursive partitioning by conditional inference. *Department of Statistics, University of Munich*.

Siddiqi, N. (2006). *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. John Wiley and Sons.

Thomas, L. (2002). *Credit Scoring and Its Application*. SIAM.

Zeng, G. (2014). Necessary condition for a good binning algorithm in credit scoring. *Applied Mathematical Sciences*, 8(65):3229–3242.

Table 1: *Reference Implementations*

Algorithm	Implementation	Parameters
Equal-Size	MATLAB histogram	nbins=8
Equal-Width	MATLAB histogram	edges=[0 12.5 25 .. 100] percetiles of LTV
Optimal	custom	p-value threshold 0.05
MLMCC	MATLAB autobinning	default parameters
Multi-Interval Discretization	R discretize::mldp	default parameters
Chi-Merge	R discretize::chim	default parameters
Conditional Inference Trees	R smbinning package	default parameters
Monotone Optimal	reference implementation	p=0.01 size-threshold=1%

A. HELOC

The binning results of HELOC dataset by various algorithms is presented below:

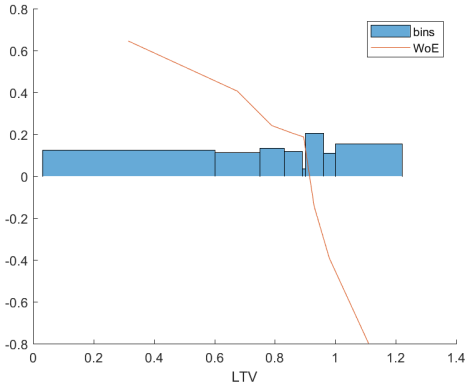

Figure 5: *Equal Size 8 bins*

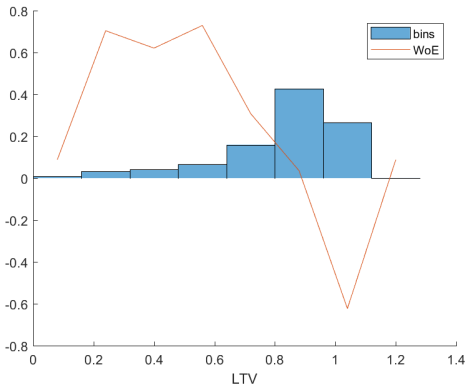

Figure 6: *Equal-Width 8 bins*

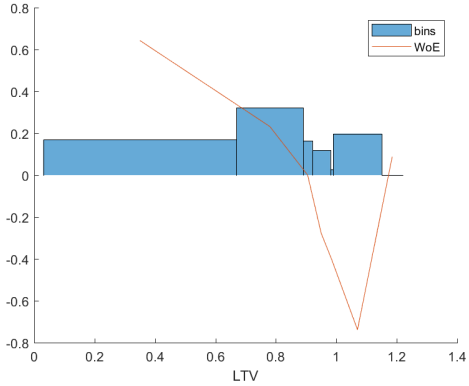

Figure 7: *Optimal Binning with p=0.05*

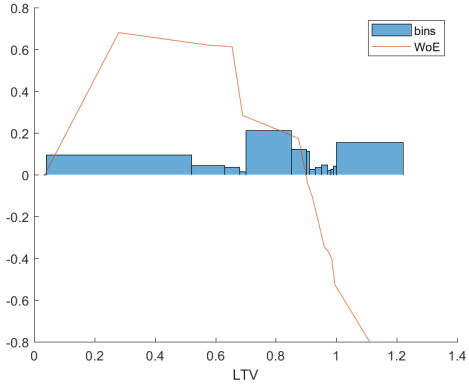

Figure 8: *MLMCC*

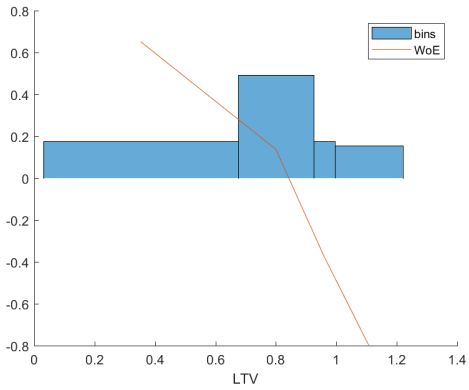

Figure 9: *MLDP*

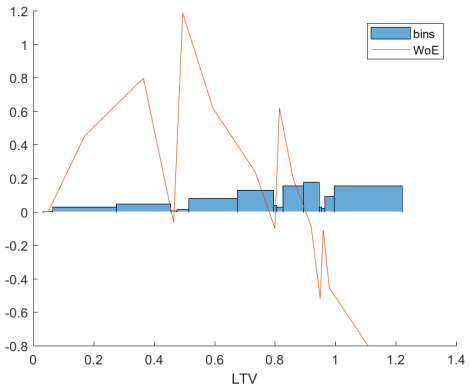

Figure 10: *ChiMerge*

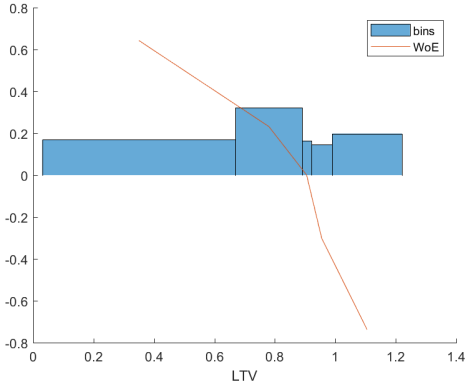

Figure 11: *Conditional Inference Trees*

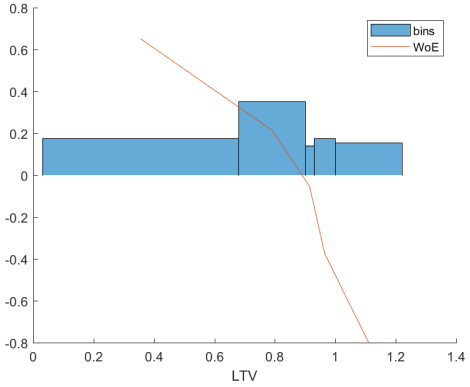

Figure 12: *Monotone Optimal Binning*

Listing 1: *Source code of monotone optimal binning algorithm*

```
function [ binstable, exttable ] = monotoneOptimal( inputtable, varName, ...
                        indicatorName, sign, ...
                        n_threshold, defaults_threshold, ...
                        p_threshold, nan_mask)

if ~ strcmp(class(inputtable), 'table')
    error('inputtable should be a table');
end

variable = table2array(inputtable(:,{varName}));
indicator = table2array(inputtable(:,{indicatorName}));
ordered_variables = unique(sort(variable(~isnan(variable))));

if sign < 0
    ordered_variables = ordered_variables(end:-1:1);
end

% finalTable columns
% 1 - variable
% 2 - mean
% 3 - nsamples
% 4 - std dev
finalTable = zeros(size(ordered_variables,1), 4);
for i = 1:size(variable,1)
    val = variable(i);
    if ~ isnan(val)
        ind = indicator(i);
        k = find(ordered_variables' == val);
        finalTable(k,1) = val;
        c = finalTable(k, 2 + ind);
        finalTable(k, 2 + ind) = c + 1;
    end
end

% means and stds
for i = 1:size(finalTable,1)
    i_0 = finalTable(i,2);
    i_1 = finalTable(i,3);

    n = (i_1 + i_0);
    m = (i_1/(i_1 + i_0));
    s = std([ zeros(1, i_0) ones(1, i_1) ]);
    finalTable(i, :) = [ finalTable(i,1) m n s  ];
end

% segment NaNs
use_nan_mask = false;
```

```
nan_row = [];
nan_positions = find(isnan(variable));
if (size(nan_positions,1) > 0)
    if exist('nan_mask')
       for submask_id = 1:size(nan_mask,2)
           nan_positions = find(isnan(variable) & nan_mask(:,submask_id));
           nan_row = [ nan_row; nan, ...
               mean(indicator(nan_positions)) ...
               size(nan_positions,1) std(indicator(nan_positions)) ];
       end
       use_nan_mask = true;
    else
        nan_row = [ nan,  mean(indicator(nan_positions)) ...
            size(nan_positions,1) std(indicator(nan_positions)) ];
    end
end

% order by mean
deleteMask = zeros(size(finalTable,1),1);
while true
     merge = false;

     % apply mask
     validItems = find(deleteMask==0);
     if size(validItems,1) < 2
         break;
     end

     j = 2;
     prevj=validItems(j-1);
     curj=validItems(j);

     while true
         if finalTable(prevj,2) <= finalTable(curj,2)
             merge = true;
             n = finalTable(prevj,3) + finalTable(curj,3);
             m = (finalTable(prevj,2)*finalTable(prevj,3) ...
                 + finalTable(curj,2)*finalTable(curj,3))/n;
             if n == 2
                 s = std([finalTable(prevj,2)  finalTable(curj,2)]);
             else
                s = sqrt((finalTable(prevj,3)*finalTable(prevj,4)^2 ...
                    + finalTable(curj,3)*finalTable(curj,4)^2)/(n));
             end
             finalTable(prevj,:) = [finalTable(prevj,1) m n s];
             deleteMask(curj) = 1;

             j = j + 1;
```

```
            if j > size(validItems,1)
                break
            end
            curj = validItems(j);
            continue;
        end
        prevj = curj;

        j = j + 1;
        if j <= size(validItems,1)
           curj = validItems(j);
        else
            break;
        end
    end
    if ~merge
        % no merges - good to exit
        break;
    end
end

% delete unused
finalTable = finalTable(~deleteMask,:);

% merging by p-value
% - Compute z-test p-value for all adjacent bins, if the number
%     of defaults is less than md or the number of observations is
%     less than mo add 1 to p-value, if a bin contains just one
%     observation then set p-value=2
% - Merge two bins with the highest p-value and repeat (10) till
%     all p-values are below the critical p-value
while true
    % take maximum to merge
    pvalues_table = [];
    for j = 2:size(finalTable,1)
        n = finalTable(j-1,3) + finalTable(j,3);
        m = (finalTable(j-1,2)*finalTable(j-1,3) ...
            + finalTable(j,2)*finalTable(j,3))/n;

        s2 = ((finalTable(j-1,3)*finalTable(j-1,4)^2 ...
            + finalTable(j,3)*finalTable(j,4)^2))/(n-2);
        if s2 > 0
            zvalue = (finalTable(j-1,2) - finalTable(j,2)) ...
                / sqrt(s2 * (1/finalTable(j-1,3) + 1/finalTable(j,3)));
            pvalue = 1 - normcdf(zvalue);
        else
            pvalue = 2;
        end
```

```
        if (finalTable(j-1,3) < n_threshold ...
         || finalTable(j,3) < n_threshold) ...
         || ((finalTable(j-1,2)*finalTable(j-1,3) < defaults_threshold) ...
         || (finalTable(j,2)*finalTable(j,3) < defaults_threshold))
            pvalue = pvalue + 1;
        end
        pvalues_table = [ pvalues_table ; pvalue j-1 m n sqrt(s2) ];
     end

    if size(pvalues_table, 1) > 0
        m = find(pvalues_table(:,1) == max(pvalues_table(:,1)));
        if size(m,1) > 0
            j = m(1); %randi([1,size(m,1)]));
        else
            j = m;
        end

        if pvalues_table(j,1) > p_threshold
           finalTable(pvalues_table(j,2),:) = ...
               [finalTable(pvalues_table(j,2),1) pvalues_table(j,3:end)];
           finalTable(pvalues_table(j,2)+1,:) = [];
        else
            break;
        end
    else
        break;
    end
end

if (size(nan_positions,1) > 0)
    finalTable = [ nan_row ; finalTable ];
end

% computing weight of evidence
bads = (finalTable(:,2) .* finalTable(:,3));
goods = finalTable(:,3) - (finalTable(:,2) .* finalTable(:,3));
totalBads = sum(bads);
totalGoods = sum(finalTable(:,3)) - totalBads;

WoE =  log((goods / totalGoods) ./ (bads / totalBads))*100;
WoEgood = find(~ isinf(WoE));
WoEmin = min(WoE(WoEgood));
WoEmax = max(WoE(WoEgood));
WoE = (WoE - WoEmin * ones(size(WoE))) / (WoEmax - WoEmin);

output = [ finalTable WoE ];
exttable = inputtable;
```

```
WoEinput = zeros(size(variable,1),1);

% assigning data to original table
for j = 1:size(variable,1)
    v = variable(j);
    if isnan(v)
        if use_nan_mask
            WoEinput(j) = output(find(nan_mask(j,:) == 1), 5);
        else
            WoEinput(j) = output(1, 5);
        end
    else
        detected = false;
        for k = 3:size(output,1)
            if (v < output(k,1) && sign >0) ...
                    || (v > output(k,1) && sign < 0)
                WoEinput(j) = output(k-1,5);
                detected = true;
                break;
            end
        end

        if ~ detected
            WoEinput(j) = output(end,5);
        end
    end
end

binstable = table;
binstable.Bin = [output(:,1)];
binstable.Mean = [output(:,2)];
binstable.N = [output(:,3)];
binstable.Std = [output(:,4)];
binstable.WoE = [output(:,5)];
exttable.WoE = WoEinput;
end
```