LEARNING STABLE WEIGHTS FOR DATA OF VARYING DIMENSION

G. Beliakov, S. James
Deakin University
School of Information Technology
Faculty of Science,
Engineering and Built Environment
Burwood, VIC, Australia
{gleb,sjames}@deakin.edu.au

D. Gómez, J. T. Rodríguez, J. Montero
Complutense University
Department of Statistics and Operational Research
Faculty of Mathematics and Faculty of Statistics
Madrid, Spain
dagomez@estad.ucm.es,
{jtrodrig,monty}@mat.ucm.es

Summary

In this paper we develop a data-driven weight learning method for weighted quasi-arithmetic means where the observed data may vary in dimension.
Keywords: Aggregation functions, R-stability, linear programming, weights learning

1 INTRODUCTION

A fundamental problem in data analysis is to determine the influence of variable inputs on an observed output, however with real world data we acknowledge that sometimes the dimension of data to be processed cannot be fully fixed in advance (for example, some expected data might be lost or corrupted, but also some unexpected data might arrive).

We assume the existence of a modelling function $f : \Re^n \to \Re$, where f depends on a vector of weighting parameters $\mathbf{w}$ and $f_{\mathbf{w}}(\mathbf{x}) = y$. In the case of the classical weighted arithmetic mean, y is modeled as a linear combination of the inputs, i.e. $f(\mathbf{x}) = w_1x_1+w_2x_2+\ldots+w_nx_n$, $\sum_{i=1}^n w_i = 1$ and each weight w_i reflects the importance of the i-th contributing variable. Learning the weighting vector also allows us to predict the output for unobserved input vectors. In some situations however, rather than being dependent on n values, the dimension of $\mathbf{x}$ may vary. In practice, y may be a function of multiple input sensors, some of which are turned off or unaccessible at given times, or in multi-criteria decision making, some observable inputs may be missing.

The idea of stability (or L-stability, R-stability, LR-stability) has been proposed for aggregating data of varying dimension in a consistent way [18, 4], e.g. if we have a weighted mean given by $y = 0.6x_1 + 0.4x_2$ when $n = 2$, then it would not usually make sense if we extend this function to 3 variables with y expressed as $y = 0.1x_1 + 0.3x_2 + 0.6x_3$, as this now implies that the first variable is less important than the second. It was established in [8, 18] that for weighting vectors of n and $(n-1)$ dimensions, R-stability, i.e. stability with respect to adding a new input in the n-th position, requires:

$$w_i^n = (1 - w_n^n) \cdot w_i^{n-1}, i = 1, \ldots, n-1, \qquad (1)$$

where w_i^n denotes the i-th weight of the n-dimensional weighting vector.

In this paper, we consider the problem of learning stable weighting vectors satisfying this property from observed data.

We will set out the article as follows. We first give an overview of aggregation functions and stability, as well as some existing approaches to learning weighting vectors using linear programming techniques. In Section 3, we look at how weights can be learnt for data of different dimension, provided we have access to observed values y. In Section 4, we run some experiments to demonstrate the usefulness of the method, while in Section 5 we summarize with some discussion and directions for further research.

2 PRELIMINARIES

This contribution applies weight learning techniques in order to learn aggregation function tuples whose weights are stable. As well as providing an overview of aggregation functions, in this section we will also recall some results concerning the concept of aggregation stability and the least absolution deviation fitting method.

ISBN: 978–83–8012–519–3

2.1 AGGREGATION FUNCTIONS

Aggregation functions are core to many decision processes, providing a representative output from an n-dimensional input vector. Overviews of their properties and some fundamental results can be found in [5, 15, 21] (also see [2, 7, 9, 13]).

Definition 1 *An aggregation function $A : [0,1]^n \to [0,1]$ is a function increasing in each argument and satisfying $A(0,\ldots,0) = 0$ and $A(1,\ldots,1) = 1$.*

Here we are interested particularly in aggregation functions that are averaging, i.e. that their inputs are bounded such that for all $\mathbf{x} \in [0,1]^n$,

$$\min(\mathbf{x}) \leq A(\mathbf{x}) \leq \max(\mathbf{x}).$$

An aggregation functions that generalizes a number of important families is the quasi-arithmetic mean. We provide the following definition.

Definition 2 *For a strictly monotone continuous generating function[1] $g : [0,1] \to [-\infty, \infty]$ and weighting vector $\mathbf{w}$ such that $w_i \in [0,1]$ and $\sum_{i=1}^{n} w_i = 1$, the weighted quasi-arithmetic mean is given by,*

$$QAM_{\mathbf{w}}(\mathbf{x}) = g^{-1}\left(\sum_{i=1}^{n} w_i g(x_i)\right). \tag{2}$$

Special cases include weighted arithmetic means (WAM) $\sum_{i=1}^{n} w_i x_i$, where $g(t) = t$, weighted power means $(\sum_{i=1}^{n} w_i x_i^p)^{\frac{1}{p}}$, where $g(t) = t^p$ and weighted geometric means $G(\mathbf{x}) = \prod_{i=1}^{n} x_i^{w_i}$ if $g(t) = -\ln t$. The weights w_i are usually non-negative and sum to one.

2.2 R-STABLE WEIGHTING VECTORS

The idea of strict stability represents a kind of consistency in the aggregation process when aggregation families are defined for varying dimension. Rojas et al. proposed the following conditions in [18], using Yager's self-identity property [22] as their basis.

Definition 3 *Let $\{A_n : [0,1]^n \to [0,1], n \in N\}$ be a family of aggregation functions. Then it is said that:*

1. *$\{A_n\}_n$ is R-strictly stable if $A_n(x_1, x_2, \ldots, x_{n-1}, A_{n-1}(x_1, x_2, \ldots, x_{n-1}))$ coincides with $A_{n-1}(x_1, x_2, \ldots, x_{n-1})$.*
2. *$\{A_n\}_n$ is L-strictly stable if $A_n(A_{n-1}(x_1, x_2, \ldots, x_{n-1}), x_1, x_2, \ldots, x_{n-1})$ coincides with $A_{n-1}(x_1, x_2, \ldots, x_{n-1})$*
3. *$\{A_n\}_n$ is LR-strictly stable if both properties hold simultaneously.*

The idea that a subset of values can be replaced by their mean with no effect on the overall output for quasi-arithmetic means has been well established for almost a century, e.g. see [1] and the references contained therein. More recently, results have been established in [4, 18]. In particular, the geometric means and arithmetic means with respect to a weighting vector with equal weights are considered LR-strictly stable, as too are the maximum, minimum, and median.

For weighted versions of these operators, strict stability is dependent on the choice of weights. The basis result regarding the weights lies in the following proposition.

Proposition 1 *Let $\mathbf{w}^n \in [0,1]^n, n \in \mathbb{N}$ be a sequence of weighting vectors such that $\sum_{i=1}^{n} w_i^n = 1$ holds $\forall n \geq 2$. Then, the family of weighted means defined by these weights is R-strictly stable if and only if the weighting property described by Eq. (1) holds.*

Analogous results hold for the additional input being included in the 1st position or j-th position [4], since in most cases it is not important where the new input is placed as long as the relationship holds between corresponding inputs as the arity is increased. It is more complicated for functions such as the ordered weighted averaging operator because we cannot generally predict where the output of A_{n-1} would be placed when the inputs are reordered. In this paper we will contain ourselves to functions which do not involve a reordering step.

2.3 LEARNING WEIGHTS USING LINEAR PROGRAMMING

In order to analyze the data set and learn weights, we can use linear programming techniques based on the minimization of the least absolute deviation (LAD) of residuals [3, 6]. In the standard case for weight learning, we have a function $f_{\mathbf{w}}$ which is dependent on $\mathbf{w}$, and a set of observed values y_k which we want the function to predict once we know its parameters. So we have

$$\text{Minimize} \quad \sum_{k=1}^{K} \left| f_{\mathbf{w}}(\mathbf{x}_k) - y_k \right|, \tag{3}$$

subject to any desired constraints.

[1] See [5] for more information regarding the choice of generators and their construction. Where $g(0)$ or $g(1)$ approach $\pm\infty$, special care needs to be taken in calculation with the convention $0 \cdot \infty = 0$ usually adopted. Methods also exist for using non-continuous and non-strict generators.

The advantage of minimizing the least absolute deviation rather than a least-squares approach is that we convert it into a linear program, and further, the output set of weights should be less sensitive to outliers or extreme data. We represent the absolute differences between the predicted and observed output values in terms their positive and negative parts, i.e. $r_k = |f_{\mathbf{w}}(\mathbf{x})_k - y_k| = r_k^+ + r_k^-$. For each observed input/output pair $(x_{k1}, x_{k2}, ..., x_{kn}, y_k)$, one of the r_k^+, r_k^- will be zero.

The weight learning can then be performed with the objective of minimizing the residuals with the following program.

$$\begin{aligned} \underset{\mathbf{w}}{\text{Minimize}} \quad & \sum_{k=1}^{K} r_k^+ + r_k^-, \qquad (4) \\ \text{s.t.} \quad & f_{\mathbf{w}}(\mathbf{x}_k) - r_k^+ + r_k^- = y_k, k = 1, \ldots, K, \\ & w_1 + w_2 + \ldots + w_n = 1, \\ & r_k^+, r_k^- \geq 0, \\ & w_i \geq 0, i = 1, \ldots, n. \end{aligned}$$

The weights for quasi-arithmetic means, including all their special cases, can be fit in the same manner with generator transformations to the observed data.

3 LEARNING STABLE WEIGHTS WITH MISSING INPUT DATA

The problem of varying dimension or missing data is common in machine learning and classification with real datasets, e.g. see [11]. New information pertaining to a particular dataset may become available, introducing new variables that are not observable for the previously collected data. In some cases, it may be found after the collection phase that some measurements were extremely unreliable, making them useless, or it may simply be that some variables are not relevant for particular instances. The two standard ways to approach such situations are:

- to make the data uniform by removing all data relating to variables that have not been measured across the dataset;
- to assign a neutral or default value for that particular variable when it is missing.

The latter case is often preferable, since otherwise we may lose a lot of information that could be useful for the task at hand, however it may not always be possible to assign a 'neutral' value. We consider the following example scenario.

Example 1 (Stable student evaluation) *Students competing for a scholarship are evaluated against 4 criteria: exam marks (40 %), interview (30%), application letter (10%) and 1 written reference (20%). Due to unforseen circumstances, however, the decision needs to be made earlier than anticipated and the reference for many students is yet to arrive. In order to be as fair as possible, students with all data available have their scores aggregated with respect to the full weighting vector* $\mathbf{w}^4 = (0.4, 0.3, 0.1, 0.2)$*, while for those students with a missing reference, a stable weighting vector is defined, consistent with Eq. (1) such that* $w_i^3 = w_i^4/(1 - 0.2)$*. This gives* $\mathbf{w}^3 = (0.5, 0.375, 0.125)$*.*

While the results of stability are useful for defining weighting vectors that are stable or 'consistent' across varying dimensions, we now turn to the problem of learning such families of weighting vectors when we only have the observed input/output data and don't know the relative importance of each input. Example 2 follows from Example 1.

Example 2 (Learning stable weights) *The scholarship assessment panel is unconvinced that the proportional importance allocated to the criteria properly reflects the students' potential. After the first year, they have performance data available for all the candidates, along with the data used to award the scholarships (since the late references were not used, their score data is still unavailable). An example of such data is given in Table 1.*

Table 1: Scholarship and performance scores with data missing for some students

Student	s_1	s_2	s_3	s_4	s_5
Exam (x_1)	0.5	0.6	0.2	0.5	0.7
Interview (x_2)	0.8	0.3	0.7	0.9	0.8
App. Letter (x_3)	0.4	0.6	0.1	0.3	0.7
Reference (x_4)	-	0.6	0.4	-	-
Performance (y)	0.51	0.63	0.62	0.71	0.78

The problem is now to learn both a 3- and 4-dimensional weighting vector (stable with respect to one another) from the dataset in order to approximate the importance of each criteria in assessing the academic potential of each student.

In the following section we develop our approach for addressing this problem. We note that learning paramaters for aggregating data of varying dimension has previously been considered in [16], while aggregation with missing data has been more recently considered in [19, 10, 12].

3.1 FRAMING STABLE WEIGHT LEARNING AS A BILEVEL OPTIMIZATION PROBLEM

Suppose we wish to learn a weighted quasi-arithmetic mean that best fits the data of the form in Table 1. We note that we are essentially required to learn 7 parameters, i.e. the aggregation weights that best model the output y as a function of the respective 3- and 4-dimensional input vectors. If we learn $\mathbf{w}^3$ and $\mathbf{w}^4$ separately, the weights may not satisfy Eq. (1) and therefore could not be considered stable - in this case, they would give conflicting approximations of the importance for each criterion. It would also result in fewer data with respect to the number of variables we need to learn. Lastly, the relationship in Eq. (1) is not linear with respect to the weights and therefore could not be incorporated into a simple optimization algorithm.

We therefore consider the problem as the following bilevel optimization problem.

$$\underset{\alpha,\mathbf{w}^{n-1}}{\text{Minimize}} \quad \sum_{j=1}^{J} \left| f_{\mathbf{w}}(\mathbf{x}_j) - y_j \right| + \sum_{k=1}^{K} \left| f_{\mathbf{w}}(\mathbf{x}_k) - y_k \right|$$

where $\alpha = (1 - w_n^n)$, J and K represent the number of observed data of $n-1$ and n dimensions respectively, and $\mathbf{x}_j \in [0,1]^{n-1}$ and $\mathbf{x}_k \in [0,1]^n$.

In the case of fitting quasi-arithmetic means, for fixed α, we can still minimize with respect to the same objective as in (4) and sum the residuals, however we impose the following constraints.

For $\mathbf{x}_j \in [0,1]^{n-1}$, we have

$$\left(\sum_{i=1}^{n-1} w_i g(x_{ji}) \right) - r_j^+ + r_j^- = g(y_j), \tag{5}$$

then for $\mathbf{x}_k \in [0,1]^n$, we can use Eq. (1) and our α, giving us,

$$\alpha \left(\sum_{i=1}^{n-1} w_i g(x_{ki}) \right) + w_n g(x_{kn}) - r_k^+ + r_k^- = g(y_k). \tag{6}$$

Since α is a scalar, these constraints remain linear with respect to $\mathbf{w}^{n-1}$. We remind that w_n is obtained directly from α and hence is also a fixed constant in this step of the minimization process. We then only require the constraints such that the weights in $\mathbf{w}^{n-1}$ sum to 1 and the residuals are nonnegative.

3.2 IMPLEMENTATION

We implemented the approach in R [17], adapting existing libraries we had created for least absolute deviation fitting[2].

From a dataset with the structure given in Table 1, we first build the left hand side of the constraints array, comprising the entries $g(x_{ki})$ for each instance k and $i = 1, \ldots, n-1$ and coefficients of -1 or 1 for the residuals. There is also a row for constraining the sum of weights. We note that these are not dependent on α. For each given α, we then construct the right hand side of the constraints matrix, with entries $g(y_k) - (1-\alpha) \cdot g(x_{kn})$, while the right-hand side for the weights sum is set to α. We note that while this is equivalent to implementing Eqs. (5)-(6), it means that we do not need to keep on rebuilding the left hand side for each α and this can simply be stored in memory. We used the one-dimensional Brent method for optimization which is available as part of the standard *optim* function in R and also allows the setting of lower and upper bounds (in our case $0 \leq \alpha \leq 1$).

4 EXPERIMENTS

Here we provide some experimental results to demonstrate the usefulness of our stable-weight learning approach.

4.1 GENERATED DATASETS

For each run of the experiment, we first set the fifth weight to a predefined value w_5 and randomly generated the remaining 4 weights along with 50 random n-dimensional input vectors. We then calculated the y values based on the weighting vector and added Guassian noise with a mean of 0 and standard deviation of 0.05. We split the data into 25 training and 25 test data and removed a number of the n-th entries from 0% to 90% in increments of 10% from both datasets.

We compared 3 potential approaches to dealing with such a dataset:

1 To remove the variable that has missing information for some entries;

2 To remove all instances with incomplete information;

3 To use the stable weight learning techniques described in the previous section so that all instances can be used in both training and testing.

We note that with the first method, the training dataset will be smaller, discounting any of the 25 training data with missing information from the learning

[2] As with the implementation of the optimization procedure described here, the experimental method is also available from http://aggregationfunctions.wordpress.com.

Table 2: Average overall error in predicting test data with increasing percentage of missing data.

% missing	method 1	method 2	method 3
0	0.141	0.045	0.044
10	0.142	0.053	0.052
20	0.145	0.065	0.064
30	0.142	0.077	0.075
40	0.143	0.086	0.088
50	0.142	0.094	0.094
60	0.143	0.111	0.107
70	0.141	0.123	0.116
80	0.146	0.144	0.134
90	0.144	0.165	0.137

Table 3: Average overall error in predicting test data with increasing weight to variable with missing entries

fixed w_5	method 1	method 2	method 3
0	0.045	0.055	0.046
0.1	0.052	0.056	0.049
0.2	0.074	0.065	0.059
0.3	0.098	0.076	0.070
0.4	0.126	0.088	0.086
0.5	0.145	0.097	0.094
0.6	0.179	0.112	0.110
0.7	0.208	0.122	0.120
0.8	0.237	0.138	0.133
0.9	0.265	0.153	0.143

algorithm. For the test data, however, we used the fitted n-dimensional vector and then determined a stable $(n-1)$-dimensional weighting vector for the vectors with missing entries.

4.2 RESULTS

The artificial set simulates a well-behaving dataset where the data vary in dimension because information about a contributing variable is missing. In this case, the variable is independent of any of the other contributing inputs and so there is no perfect solution to dealing with the data we have. The purpose of the experiments is hence to demonstrate the usefulness of the approach as a reasonable solution in this scenario.

In some contexts, however, a missing input may not necessarily represent ignorance about a contributing variable, and rather the output may depend on input vectors which differ in dimension by construction. For example, we might consider a nearest-neighbor approximation method where we include only neighbours within a given distance of the point we are approximating. In such cases, the 'missing' input is less likely to be affecting the output. Alternatively, we can think of aggregating the citations data of a journal or researcher over a given timeframe [20]. With our artificial dataset however, we chose not to generate data in this way as it would clearly bias the method of obtaining stable weighting vectors.

As would be expected, the method of leaving out the variable with missing entries has worse error as the importance of the missing variable increases. It is unaffected by changes in the number of missing data (since it treats the dataset as if all of the entries are missing). This relationship can be seen clearly in Tables 2-3.

The average error associated with removing the missing entries as opposed to stable fitting is quite similar. We could assume that the increase in error with method 2 mainly revolves around the calculation of the output associated with 4-dimensional outputs - since on average, the learned weighting vector should be close to the true vector. As the number of missing data increases, however, the training data set gets smaller, which in turn is likely to affect the accuracy. We note that in one case, with 90% of the data with a missing value and $w_5 = 0.9$, this method failed to generate a model[3]. Error for method 3, however, can arise in the actual fitting process, since the 4-dimensional training data have an output that was actually calculated using the missing variable. The advantage then, is that it is still able to use the entire training set to generate the weights. This is perhaps why it performs slightly better as the number of missing data gets larger. Comparisons of the average error can be seen in Figs. 1-2 and Tables 2-3.

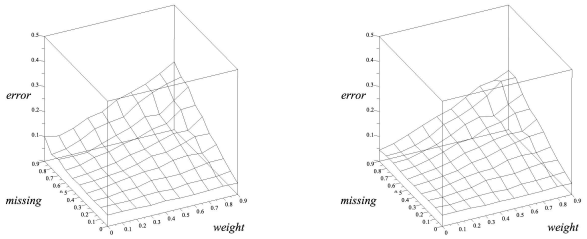

Figure 1: Average absolute error where data is fit by removing any training data with missing entries (left) and using proposed method (right) over 10 randomly generated datasets with w_5 and the percentage of x_5 entries removed.

5 CONCLUSION

We have developed and demonstrated a method for learning stable weighting vectors for datasets which have some data missing with respect to one of the variables. We conducted some experiments with artificial datasets to show that the method performs at least as reliably as other potential approaches to dealing with missing data pertaining to an independent variable. The method can obviously be extended to the problem of missing data pertaining to 2 or more variables, however with too many incomplete variables, the runtime may become impractical.

Future research for learning stable weights with miss-

[3]The error reported for this is hence the average of the 9 other tests.

ing data should take into account cases where the position of missing data cannot be imposed (see, e.g., [14], where a first attempt was proposed in terms of different variations of stability). Moreover, the search for efficient ways of dealing with missing data should be associated to the implementation with real datasets.

Acknowledgements

This research has been partially supported by the Government of Spain (grant TIN2012-32482), the Government of Madrid (CASI-CAM-CM project, grant S2013/ICE-2845)and Complutense University (research group 910149).

References

[1] J. Aczél. On mean values. *Bulletin of the American Math. Society*, 54, 1948.

[2] A. Amo, J. Montero, and E. Molina. Representation of consistent recursive rules. *European Journal of Operational Research*, 130:29–53, 2001.

[3] G. Beliakov. Construction of aggregation functions from data using linear programming. *Fuzzy Sets and Systems*, 160:65–75, 2009.

[4] G. Beliakov and S. James. Stability of weighted penalty-based aggregation functions. *Fuzzy Sets and Systems*, 226:1–18, 2013.

[5] G. Beliakov, A. Pradera, and T. Calvo. *Aggregation Functions: A Guide for Practitioners*. Springer, Heidelberg, 2007.

[6] P. Bloomfield and W. Steiger. *Least Absolute Deviations. Theory, Applications and Algorithms*. Birkhauser, Boston, Basel, Stuttgart, 1983.

[7] H. Bustince, B. de Baets, J. Fernández, R. Mesiar, and J. Montero. A generalization of the migrativity property of aggregation functions. *Information Sciences*, 191:76–85, 2012.

[8] T. Calvo, G. Mayor, J. Torrens, J. Suñer, M. Mas, and M. Carbonell. Generation of weighting triangles associated with aggregation functions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 8(4):417–451, 2000.

[9] V. Cutello and J. Montero. Recursive connective rules. *Int. J. Intelligent Systems*, 14:3–20, 1999.

[10] J. Dujmović. Aggregation operators and observable properties of human reasoning.

[11] J. Dujmović, G. D. Tré, N. Singh, D. Tomasevich, and R. Yokoohji. Soft computing models in online real estate. In M. Jamshidi, V. Kreinovich, and J. Kacprzyk, editors, *Advance Trends in Soft Computing, WCSC 2013, Studies in Fuzziness and Soft Computing 312*, pages 77–91. Springer, 2013.

[12] J. J. Dujmović. The problem of missing data in LSP aggregation. In S. G. et al., editor, *Advances in Computational Intelligence, IPMU 2012, Part III, CCIS 299*, pages 336 – 346. Springer, Catania, Italy, 2012.

[13] D. Gómez and J. Montero. A discussion on aggregation operators. *Kybernetika*, 40:107–120, 2004.

[14] D. Gómez, K. Rojas, J. Montero, J. Rodríguez, and G. Beliakov. Consistency and stability in aggregation operators, an application to missing data problems. *Int. J. Computational Intelligence Systems*, 7:595–604, 2014.

[15] M. Grabisch, J.-L. Marichal, R. Mesiar, and E. Pap. *Aggregation Functions*. Cambridge University Press, Cambridge, 2009.

[16] D. Nettleton and V. Torra. A comparison of active set method and genetic algorithm approaches for learning weighting vectors in some aggregation operators. *International Journal of Intelligent Systems*, 16(9):1069–1083, 2001.

[17] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. http://www.R-project.org/.

[18] K. Rojas, D. Gómez, J. Montero, and J. T. Rodríguez. Strictly stable families of aggregation operators. *Fuzzy Sets and Systems*, 228:44 – 63, 2013.

[19] K. Rojas, D. Gomez, J. Rodriguez, and J. Montero. Some properties of consistency in the families of aggregation functions. *Advances in Intelligent and Soft Computing*, 107:169–176, 2011.

[20] V. Torra and Y. Narukawa. The h-index and the number of citations: Two fuzzy integrals. *IEEE Transactions on Fuzzy Systems*, 16(3):795–797, 2008.

[21] Y. Torra and V. Narukawa. *Modeling Decisions. Information Fusion and Aggregation Operators*. Springer, 2007.

[22] R. R. Yager and A. Rybalov. Noncommutative self-identity aggregation. *Fuzzy Sets and Systems*, 85:73–82, 1997.