Learning to Aggregate: Tackling the Aggregation/Disaggregation Problem for OWA

Vitalik Melnikov MELNIKOV@MAIL.UPB.COM
Eyke Hüllermeier EYKE@UPB.COM
Paderborn University
Heinz Nixdorf Institute and Department of Computer Science
Intelligent Systems and Machine Learning Group

Editors: Wee Sun Lee and Taiji Suzuki

Abstract

The problem of "learning to aggregate" (LTA) has recently been introduced as a novel machine learning setting, in which instances are represented in the form of a composition of a (variable) number on constituents. Such compositions are associated with an evaluation, which is the target of the prediction task, and which can presumably be modeled in the form of a suitable aggregation of the properties of its constituents. An especially interesting class of LTA problems arises when the evaluations of the constituents are not available at training time, and instead ought to be learned simultaneously with the aggregation function. This scenario is referred to as the "aggregation/disaggregation problem". In this paper, we tackle this problem for an interesting type of aggregation function, namely the Ordered Weighted Averaging (OWA) operator. In particular, we provide an algorithm for learning the OWA parameters together with local utility scores of the constituents, and evaluate this algorithm in a case study on predicting the performance of classifier ensembles.

Keywords: supervised learning, aggregation operator, learning to aggregate, OWA

1. Introduction

The idea of combining models and aggregation functions from the field of (multi-criteria) decision making with data-driven approaches for model identification from the field of machine learning has attracted increasing attention in recent years. Examples of such combinations include methods for learning the majority rule model (Sobrie et al., 2013), the non-compensatory sorting model (Sobrie et al., 2015), or the Choquet integral (Tehrani et al., 2012b,a). In contrast to many other machine learning approaches, corresponding models are interpretable and meaningful from of decision making point of view, a property that has gained increasing attention in the recent past (Guo et al., 2019). Besides, aggregation operators offer other appealing properties that might be desirable from a machine learning point of view.

In this paper, we address the problem of "learning to aggregate", which has recently been introduced as a novel machine learning task by Melnikov and Hüllermeier (2016). Roughly speaking, learning-to-aggregate (LTA) problems are supervised learning problems, in which instances are represented in the form of a composition of a (variable) number on constituents; such compositions are associated with an evaluation, score, or label, which is

the target of the prediction task, and which can presumably be modeled in the form of a suitable aggregation of the properties of its constituents. Thus, LTA establishes another connection between machine learning and the study of aggregation functions. Indeed, from a machine learning point of view, a key concern is to take advantage of useful properties of aggregation functions, such as monotonicity, associativity, etc.

As an illustration and motivating example, consider the problem of learning to rate playlists of songs: A playlist can be considered as a composition (either a list, or, when ignoring the sequential structure, even a set) of individual songs, and it is reasonable to assume that the overall evaluation of a playlist (for example, on a scale ranging from one to five stars) can be expressed as a suitable aggregation of the evaluation of the constituent songs. The learning task may consist of inducing a predictive model, which can (hypothetically) rate any new playlist, on the basis of a set of training data in the form of playlists together with their ratings. From a machine learning point of view, this problem comes with a number of challenges. For example, note that playlists, which form the input of the sought predictor, might be of different length. Moreover, we may want the predictor to be invariant toward permutations of the songs in a list, at least to some extent. Standard feature representations as commonly used in supervised learning might therefore not be appropriate.

An especially interesting class of learning problems arises when the evaluations of the constituents are not available at training time, and instead ought to be learned simultaneously with the aggregation function. In our playlist example, for instance, it is reasonable to assume that ratings are only available at the level of entire lists, but not at the level of individual songs. This scenario is referred to as the "aggregation/disaggregation problem" — the notion of *disaggregation* (Jacquet-Lagreze and Siskosb, 2001) refers to the decomposition of global scores into several local scores, which inverts the direction of aggregation (from local scores to global ones).

In this paper, we tackle the aggregation/disaggregation problem for a specific type of aggregation function, namely the Ordered Weighted Averaging (OWA) operator (Yager, 1988). OWA is a family of parametrized averaging functions with several appealing properties. Although the learning of OWA operators from data has already been addressed in the literature (Filev and Yager, 1994; Beliakov, 2002, 2005), the problem of simultaneously learning OWA weights as well as its arguments has not been considered so far. To the best of our knowledge, this problem was not considered for other aggregation operators either. Instead, existing methods for learning aggregation functions, like for example the Choquet integral (Tehrani et al., 2012a; Islam et al., 2018; Dias et al., 2019; Aggarwal and Tehrani, 2019), assume that local scores $y_{i,j}$ of the constituents $c_{i,j}$ are already given, and consider the task of learning their aggregation into an overall score y_i.

The paper is organized as follows. In the next section, we recall the LTA framework as introduced by Melnikov and Hüllermeier (2016). In Section 3, the OWA operator is introduced, along with a corresponding learning algorithm. We evaluate this algorithm in Section 4 on synthetic data and in a practical use case in the context of ensemble learning. A summary and a discussion of directions for future work conclude the paper in Section 5.

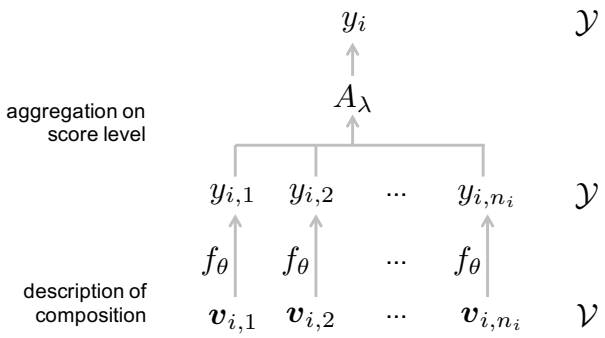

Figure 1: Illustration of the basic structure of a learning-to-aggregate model.

2. Learning to Aggregate

In spite of certain generalizations that have been proposed in the recent past, the bulk of methods for supervised machine learning still proceeds from a formal setting in which data objects (instances) are represented in the form of feature vectors. Thus, an instance $\boldsymbol{x}$ is described in terms of a vector $(x_1, \ldots, x_d) \in \mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$, where $\mathcal{X}_i$ is the domain of the ith attribute or feature. The corresponding view of instances as *points* in a *space* of fixed dimension d has largely influenced the way in which learning problems are studied and methods developed: Supervised learning is considered as *embedding* objects as data points in the space $\mathcal{X}$, and then *separating* these points (in the case of classification) or *fitting* them (in the case of regression) using models that have a natural geometric interpretation, such as hyperplanes or any other type of decision boundary or manifold in the space $\mathcal{X}$. A prediction $\hat{y}$ of the output $y \in \mathcal{Y}$ associated with an instance $\boldsymbol{x}$ is then obtained by means of a corresponding function $f : \mathcal{X} \longrightarrow \mathcal{Y}$.

While this approach to formalizing and tackling learning problems proved to be highly successful, there are problems for which the production of predictions $\hat{y}$ by means of a (single) function f defined on the space $\mathcal{X}$ is arguably less appropriate. One such class of problems is "aggregation". The view we promote is to consider data objects as *compositions* of individual *constituents*; moreover, we assume that the output associated with such a composition is obtained as an aggregation of the properties of the individual constituents, using a suitable type of aggregation function (Grabisch et al., 2009).

In the remainder of this section, we recall the formal framework of LTA as introduced by Melnikov and Hüllermeier (2016) and elaborate on some of its properties. An overview of the framework is given in Fig. 1.

2.1. Formal Setting and Notation

We proceed from a set of training data

$$\mathcal{D} = \big\{(\boldsymbol{c}_1, y_1), \ldots, (\boldsymbol{c}_N, y_N)\big\} \subset \mathcal{C} \times \mathcal{Y} \ , \tag{1}$$

where $\mathcal{C}$ is the space of *compositions* and $\mathcal{Y}$ a set of possible (output) values associated with a composition; since aggregation is often used for the purpose of evaluating a composition, we

also refer to the values y_i as *scores*. A composition $\boldsymbol{c}_i \in \mathcal{C}$ is a multiset (*bag*) of constituents

$$\boldsymbol{c}_i = \{c_{i,1}, \ldots, c_{i,n_i}\} \ ,$$

where $n_i = |\boldsymbol{c}_i|$ is the size of the composition; scores y_i are typically scalar values (real numbers or values from an ordinal scale, such as 1 to 5 star ratings in recommender systems). Constituents $c_{i,j}$ can be of different type. In particular, the description of a constituent may or may not contain the following information:

- **Constituent roles:** A label specifying the role of the constituent in the composition. For example, suppose a composition is a menu consisting of constituents in the form of dishes; each dish could then be labeled with appetizer, main dish, or dessert, thereby providing information about the part of the menu it belongs to (and hence adding additional structure to the composition).
- **Constituent properties:** A description of *properties* of the constituent. For example, each dish could be described in terms of certain nutritional values. Formally, we assume properties to be given in the form of a feature vector $\boldsymbol{v}_{i,j} \in \mathcal{V}$, where $\mathcal{V}$ is a corresponding feature space. We note, however, that more complex descriptions are conceivable; for example, the description could itself be a composition.
- **Constituent quantities:** A *quantity* representing the amount of the constituent in the composition (instead of simply informing about the presence or absence of the constituent).
- **Constituent scores:** A *local evaluation* in the form of a score $y_{i,j}$, which is typically real-valued (i.e., $y_{i,j} \in \mathbb{R}_+$) but may also be binary or ordinal.

Finally, a composition can also be equipped with an additional structure in the form of a (binary) relation on its constituents. In this case, a composition is not simply an unordered set (or bag) of constituents but a more structured object, such as a sequence or a graph.

Like in standard supervised learning, the goal in learning to aggregate is to induce a model $h : \mathcal{C} \longrightarrow \mathcal{Y}$ that predicts scores for compositions. More specifically, given a hypothesis space $\mathcal{H}$ and a loss function $L : \mathcal{Y}^2 \longrightarrow \mathbb{R}_+$, the goal is to find a risk-minimizing hypothesis

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int_{\mathcal{C} \times \mathcal{Y}} L\big(y, h(\boldsymbol{c})\big) \, d\,\mathbf{P}(\boldsymbol{c}, y)$$

on the basis of the training data (1).

2.2. Disaggregation

As already said, existing methods for learning aggregation functions assume the local scores $y_{i,j}$ of the constituents $c_{i,j}$ to be given, and consider the task of learning their aggregation into an overall score y_i. This is indeed the genuine purpose of aggregation functions, which typically assume that all scores are *commensurable* and elements of the same scale $\mathcal{Y}$. For example, one might be interested in how the scores on a conference paper (strong reject, reject, ..., strong accept) coming from a (variable) number of reviewers are aggregated into an overall rating by the program chairs. Sometimes, the additional assumption is made

that individual scores are *criteria* (which, in our terminology, means that they have a unique role, and that n_i is given by the number of criteria and hence the same for each composition). The main question, then, is how the rating of an alternative on different criteria is aggregated into an overall rating. For example, one might be interested in how reviewers combine their ratings on criteria such as readability, novelty, etc. into an overall rating of a paper.

Now, suppose that local scores $y_{i,j}$ are *not* part of the training data. Instead, the constituents $c_{i,j}$ are only described in terms of properties in the form of feature vectors $\boldsymbol{v}_{i,j} \in \mathcal{V}$, or, even simpler, in the form of labels (i.e., their names). In our playlist example, for instance, individual songs may just be identified by their title, or perhaps described in terms of features such as genre, artist, length, etc. A natural way to tackle the learning problem, then, is to consider the local scores as latent variables, and to induce them as functions $f : \mathcal{V} \longrightarrow \mathcal{Y}$ of the properties.

In the following, we assume these functions to be parametrized by a vector $\boldsymbol{\theta}$, and the aggregation function A by a parameter $\boldsymbol{\lambda}$. The model is then of the form

$$y_i = A_{\boldsymbol{\lambda}}(y_{i,1}, \ldots, y_{i,n_i}) = A_{\boldsymbol{\lambda}}\Big(f_{\boldsymbol{\theta}}(\boldsymbol{v}_{i,1}), \ldots, f_{\boldsymbol{\theta}}(\boldsymbol{v}_{i,n_i})\Big) \ , \tag{2}$$

and the problem consists of learning both the aggregation function A, i.e., the parameter $\boldsymbol{\lambda}$, and the mapping from features to local scores, i.e., the parameter $\boldsymbol{\theta}$, simultaneously. Here, supervision only takes place at the level of the entire composition, namely in the form of scores y_i, whereas the "explanation" of these scores via induction of local scores is part of the learning problem.

As already mentioned, the decomposition of global scores into several local scores is referred to as disaggregation. For example, suppose we observe a user's ratings of different playlists, each one considered as a collection of songs, but not of the individual songs themselves. In order to predict the user's rating of new playlists, we could then try to learn how she rates individual songs and, simultaneously, how she aggregates several (local) ratings into a global rating.

Obviously, there is a strong interaction between the local ratings and their aggregation into a global score. For example, if we consistently observe low scores for different playlists, this could be either because the user dislikes (almost) all songs, or because she dislikes only a few but aggregates very strictly (i.e., a playlist gets a low score as soon as is contains a single or a few poor songs). An important question, therefore, concerns the *identifiability* of the model, i.e., the question whether different parametrizations imply different models (or, more formally, whether $(\boldsymbol{\lambda}, \boldsymbol{\theta}) \neq (\boldsymbol{\lambda}', \boldsymbol{\theta}')$ implies that the corresponding models assign different scores $y_i \neq y_i'$ for at least one composition).

3. Learning the OWA Operator

Even in its basic form shown in Figure 1, the LTA framework can be instantiated in various ways and gives rise to a number of different learning problems, in particular depending on the type of data that is observed and can be used for training: Compositions are of the same size or vary in size; a composition is a simple (multi-)set or has a more structured description, for example as a sequence or graph; constituent roles, constituent properties, constituent quantities, and constituent scores are available or not.

In this paper, we are interested in a setting of the following kind: Compositions $\boldsymbol{c}_i$ are bags of varying size n_i but have no additional structure. Moreover, constituents are taken from a finite set

$$\mathcal{O} = \{o_1, \ldots, o_M\} \tag{3}$$

of objects, which are only identified by their name but not described in terms of any other properties. Perhaps most importantly, no information about local evaluations, i.e., scores of the constituents, is supposed to be given.

An aggregation function that appears to be suitable in this setting is the Ordered Weighted Averaging (OWA) operator (Yager, 1988). In the following, we first recall the definition of the OWA operator and some of its properties, prior to addressing the problem of learning this operator by means of an aggregation/disaggregation approach.

3.1. The OWA Operator

Recall that, in our model (2), the aggregation step consists of combining the local scores $y_{i,1}, \ldots, y_{i,n_i}$ into an overall score y_i, using a suitable aggregation function $A_{\boldsymbol{\lambda}}$. As already said, we instantiate the latter by the OWA operator. Simplifying notation by omitting the index i and writing $(z_1, \ldots, z_n)$ for the vector of local scores $(y_{i,1}, \ldots, y_{i,n_i})$, the OWA operator is defined as follows:

$$A_{\boldsymbol{\lambda}}(\boldsymbol{z}) = A_{\boldsymbol{\lambda}}(z_1, z_2, \ldots, z_n) = \sum_{i=1}^{n} \lambda_i \, z_{(i)} \,, \tag{4}$$

where $(\lambda_1, \ldots, \lambda_n) \in [0,1]^n$ is a weight vector such that $\lambda_1 + \ldots + \lambda_n = 1$, and $(\cdot)$ a permutation $\{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ such that $z_{(1)} \leq z_{(2)} \leq \cdots \leq z_{(n)}$. Thus, as an important difference to the standard weighted average (and, more generally, linear models typically used in machine learning), the weight λ_i pertains to the i^{th} largest element of the input vector $\boldsymbol{z} = (z_1, \ldots, z_m)$, not to the i^{th} element. This makes perfect sense if, like in the context of LTA, this vector is actually supposed to represent a set with no specific order of the elements. Reconsider, for example, the problem to aggregate reviewer scores z_1, z_2, z_3 on a conference article. Obviously, a (generalized) linear model $\lambda_1 z_1 + \lambda_2 z_2 + \lambda_3 z_3$, in which λ_i quantifies the influence of the i^{th} reviewer, would be meaningless, simply because the reviewers have no natural order and may vary from article to article. What does make sense, on the other side, is to quantify the influence of the best, the second best, and the worst evaluation on the overall score.

According to (4), $\min(\boldsymbol{z}) \leq A_{\boldsymbol{\lambda}}(\boldsymbol{z}) \leq \max(\boldsymbol{z})$ holds for every input vector $\boldsymbol{z}$, and both operators min and max are special cases of OWA. Indeed, the minimum can be obtained by the weight vector $\boldsymbol{\lambda}_{min} = (1, 0, \ldots, 0)$ and the maximum by $\boldsymbol{\lambda}_{max} = (0, \ldots, 0, 1)$. Likewise, the arithmetic mean is recovered as a special case for $\boldsymbol{\lambda}_{am} = (1/n, 1/n, \ldots, 1/n)$.

Additional mathematical properties of the OWA operator include piecewise linearity, Lipschitz continuity, symmetry (i.e., invariance toward permutation of the inputs $z_1, \ldots, z_n$), and shift-invariance (Beliakov et al., 2011):

$$A_{\boldsymbol{\lambda}}(\boldsymbol{z}') = \sum_{i=1}^{n} \lambda_i \, z'_{(i)} = \sum_{i=1}^{n} \lambda_i \, (z_{(i)} + c) = \sum_{i=1}^{n} \lambda_i z_{(i)} + \sum_{i=1}^{n} \lambda_i \, c = \sum_{i=1}^{n} \lambda_i z_{(i)} + c = A_{\boldsymbol{\lambda}}(\boldsymbol{z}) + c$$

for $\boldsymbol{z}' = \boldsymbol{z} + c$, $c \in \mathbb{R}$. Obviously, the OWA operator is also identifiable: Consider two OWA functions $A_{\boldsymbol{\lambda}}$ and $A_{\boldsymbol{\lambda}'}$ parametrized by $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$, respectively, and such that $A_{\boldsymbol{\lambda}}(\boldsymbol{z}) = A_{\boldsymbol{\lambda}'}(\boldsymbol{z})$ for all $\boldsymbol{z} \in \mathbb{R}^n$. Then, $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$ immediately follows from

$$\sum_{j=1}^{k} \lambda_j = A_{\boldsymbol{\lambda}}(\underbrace{1, \ldots, 1}_{k \text{ entries}}, 0, \ldots, 0) = A_{\boldsymbol{\lambda}'}(\underbrace{1, \ldots, 1}_{k \text{ entries}}, 0, \ldots, 0) = \sum_{j=1}^{k} \lambda'_j$$

for all $k \in \{1, \ldots, n\}$.

3.2. BUM Functions

In spite of its many appealing mathematical properties, the OWA operator is not *associative*. Indeed, as already shown by Dubois and Prade (1984), the only symmetric associative means are the α-medians (Grabisch et al., 2009). For the OWA operator, this implies that only the special cases min and max are associative. Since we assume that the size of compositions n_i can vary even inside the same dataset, we have to find another way to construct OWA operators of varying arity n.

One solution was proposed by Yager (1996, 2004). Intuitively, the idea is to parametrize a whole family of "similar" OWA operators by a so-called basic unit interval monotone (BUM) function $q : [0,1] \longrightarrow [0,1]$, for which $q(0) = 0$ and $q(1) = 1$. The i^{th} weight λ_i of the OWA operator with the arity n is defined as

$$\lambda_i = q\left(\frac{i}{n}\right) - q\left(\frac{i-1}{n}\right) .$$

An interesting choice for BUM functions are monotone splines. As argued by Beliakov (2005), linear splines are especially appealing from a computational point of view. Although other types of monotone splines exist, we make use of monotone univariate B-splines in this paper. A BUM function q is then given by

$$q(x) = \sum_{l=0}^{m+k-1} a_l \, B_{l,k}(x) \,, \tag{5}$$

where k is the spline order, m is the number of equidistant spline knots t_l in $[0,1]$, and a_l is the coefficient of the basis spline $B_{l,k}(x)$. B-splines are recursively defined (De Boor, 2001) as

$$B_{l,0}(x) = \begin{cases} 1, & \text{if } x \in [t_l, t_{l+1}) \\ 0, & \text{otherwise} \end{cases}$$

$$B_{l,k}(x) = \frac{x - t_l}{t_{l+k} - t_l} B_{l,k-1}(x) + \frac{t_{l+k+1} - x}{t_{l+k+1} - t_{l+1}} B_{l+1,k-1}(x) \,,$$

where the set of spline knots $\{t_l\}_{l=0}^{m+2k}$ is chosen such that $t_0 \leq \cdots \leq t_k = 0 < \cdots < t_{m+k-1} = 1 \leq \cdots \leq t_{m+2k}$. The position of the knots outside the interval $[0,1]$ is arbitrary. As shown by De Boor (2001), a sufficient condition for (5) to be monotone is $a_{l+1} \geq a_l$ for $l \in \{0, 1, \ldots, m+k-1\}$, i.e., the set of spline coefficients should be non-decreasing. The boundary constraints of the BUM function are satisfied if $a_0 = 0$ and $a_{m+k-1} = 1$ (Beliakov, 2000).

3.3. Learning Model and Algorithm

Recall the learning problem as outlined in Section 2.1: From training data (1) consisting of compositions $\boldsymbol{c}_i = \{c_{i,1}, \ldots, c_{i,n_i}\}$ together with scores y_i, we seek to induce a model (2). Here, we assume that the constituents $c_{i,j}$ are taken from the set of objects (3). The local scores of these constituents, $\boldsymbol{s} = (s_1, \ldots, s_M) \in \mathbb{R}^M$, are assumed to be unknown and therefore considered as parameters of the model. More specifically, our (noisy) OWA model assumes the following dependency:

$$y_i = F_{\boldsymbol{a},\boldsymbol{s},k,m}\big(\{o_{i,1}, \ldots, o_{i,n_i}\}\big) + \epsilon_i \tag{6}$$

where ϵ_i is an additive noise term, and

$$\begin{aligned}
F_{\boldsymbol{a},\boldsymbol{s},k,m}\big(\{o_{i,1}, \ldots, o_{i,n_i}\}\big) &= \sum_{j=1}^{n_i} \lambda_j \, s_{i,(j)} = \sum_{j=1}^{n_i} \left(q\left(\frac{j}{n_i}\right) - q\left(\frac{j-1}{n_i}\right)\right) s_{i,(j)} \\
&= \sum_{j=1}^{n_i} \left(\sum_{l=0}^{m+k-1} a_l \, B_{l,k}\left(\frac{j}{n_i}\right) - \sum_{l=0}^{m+k-1} a_l \, B_{l,k}\left(\frac{j-1}{n_i}\right)\right) s_{i,(j)} \\
&= \sum_{j=1}^{n_i} \left(\sum_{l=0}^{m+k-1} a_l \left(B_{l,k}\left(\frac{j}{n_i}\right) - B_{l,k}\left(\frac{j-1}{n_i}\right)\right)\right) s_{i,(j)} \\
&= \sum_{j=1}^{n_i} \left(\sum_{l=0}^{m+k-1} a_l G_l(j)\right) s_{i,(j)} \\
&= \sum_{l=0}^{m+k-1} \left(a_l \sum_{j=1}^{n_i} G_l(j) s_{i,(j)}\right)
\end{aligned}$$

with $s_{i,(j)}$ the j^{th} largest score among the constituents $\{o_{i,1}, \ldots, o_{i,n_i}\}$, and $\boldsymbol{a} \in \mathbb{R}_+^{m+k}$ the (unknown) spline coefficients (i.e., the parameters of the OWA operator). Moreover, m and k are hyper-parameters of the BUM function as described in the previous section.

Notice that the computation of basis spline differences

$$G_l(j) = B_{l,k}\left(\frac{j}{n}\right) - B_{l,k}\left(\frac{j-1}{n}\right) \tag{7}$$

is independent of the model parameters $\boldsymbol{s}$ and $\boldsymbol{a}$, and can therefore be precomputed for every meaningful n, k, m combination for a given dataset. In addition, it rarely makes sense to go beyond a piecewise linear spline ($k = 2$) and set the number of knots in the unit interval $m \gg n$, since only the difference of basis splines at n points and not the approximation between those points is considered.

As for the learning process, our goal is to find the model parameters $\boldsymbol{s} \in [0,1]^M$ and $\boldsymbol{a} \in \mathbb{R}_+^{m+k}$ minimizing the squared error loss

$$L(\boldsymbol{s}, \boldsymbol{a}) = \sum_{i=1}^{N} \Big(y_i - F_{\boldsymbol{a},\boldsymbol{s},k,m}\big(\{o_{i,1}, \ldots, o_{i,n_i}\}\big)\Big)^2. \tag{8}$$

To this end, we make use of (standard) gradient-based optimization techniques, which essentially presumes an efficient way to compute the empirical risk (8). We tackle this problem by processing the training data in chunks (or blocks) consisting of instances with the same composition size n. The overall empirical risk is then obtained by summing the results over these chunks (cf. Algorithm 1). In the following, we explain the computation for one data chunk containing $d \leq N$ compositions of size n.

Notice that the model (6) can be stated in matrix form (and therefore be efficiently computed on modern architectures). Since the data is assumed to be fixed in the learning process, we rewrite the model as a function of its parameters $\boldsymbol{s} \in [0,1]^M$ and $\boldsymbol{a} \in \mathbb{R}_+^{m+k}$ (and hyper-parameters $m,\ k$) as follows:

$$H(\boldsymbol{s}, \boldsymbol{a})_{m,k} = \underbrace{r\left(\text{sort}\left(O \circ \left(\mathbb{1}_d^T \boldsymbol{s}\right)\right)\right)}_{\tilde{S}} \underbrace{\left(G_{n,k,m} L \boldsymbol{a}^T\right)}_{\boldsymbol{\lambda}^T}, \tag{9}$$

where $\tilde{S}$ corresponds to the row-wise sorted matrix of constituent scores for the given data chunk and $\boldsymbol{\lambda}^T$ are the estimated OWA weights. The single components have the following interpretation:

- $r : A_{[d,M]} \to A_{[d,m+k]}$ is a row-wise projection, which removes the first $M - (m+k)$ columns from the matrix A.
- sort : $A \to A$ is a row-wise sorting operation on a given matrix A.
- O is the $d \times M$ 0/1 data matrix. A row corresponds to the "one hot" encoding of a single composition.
- $G_{n,k,m}$ is the $n \times (m+k)$ matrix of spline differences as in (7).
- L is the lower triangle matrix of size $(m+k) \times (m+k)$ consisting of ones. The product of this matrix and the parameter vector $\boldsymbol{a}$ allows us to produce a non-decreasing set of spline coefficients.
- $\circ$ is the Hadamard (element-wise) product.

The model parameters are estimated by minimizing a regularized version of the empirical risk (8), i.e., of the squared loss function $L_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ (matrix form):

$$R(H) = d^{-1} L_2\left(H(\boldsymbol{s}, \boldsymbol{a})_{m,k}^T, \boldsymbol{y}\right) \mathbb{1}_d^T + \gamma ||\boldsymbol{a}||_2^2, \tag{10}$$

where $\boldsymbol{y} = (y_1, \ldots, y_d)$ is the vector of composition scores. Recall that the spline coefficients $\boldsymbol{a}$ have to satisfy $a_0 = 0$ and $a_{m+k-1} = 1$. The hyper-parameters γ, k and m allow for controlling the smoothness of the BUM function as well as the model complexity. We discuss their influence in Section 4.

The major challenge in the minimization of (10) is the sorting operation sort. Indeed, for a sufficiently small vector $\boldsymbol{\epsilon} \in \mathbb{R}^m$, the sorted sequence sort($\boldsymbol{s}$) is almost surely the same as for sort($\boldsymbol{s} + \boldsymbol{\epsilon}$). The corresponding Jacobian matrix $\partial \text{sort}(\boldsymbol{s}) / \partial \boldsymbol{s}$ is zero-filled almost everywhere. As discussed by Grover et al. (2019), this restricts the use of any sorting operation in a backpropagation framework. In our current solution, we avoid this problem by using the optimization algorithm L-BFGS (Byrd et al., 1995), which approximates the gradient numerically.

Algorithm 1 OWA empirical risk

Data: $\boldsymbol{s},\ \boldsymbol{a}, \{O_1, \ldots, O_l\},\ \{G_1, \ldots, G_l\}, \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_l\},\ L,\ M,\ \gamma$
Result: $R(\mathbb{H})$, the regularized empirical risk over l data chunks
for $i \in \{1, 2, \ldots, l\}$ **do**
  $H_i = r\left(\text{sort}\left(O_i \circ \left(\mathbb{1}_{d_i}^T \boldsymbol{s}\right)\right)\right)\left(G_i L \boldsymbol{a}^T\right)$
  $R_i = L_2\left(H_i^T,\ \boldsymbol{y}_i\right)\mathbb{1}_d^T$
end
$R(\mathbb{H}) = \left(\sum R_i\right)/M + \gamma||\boldsymbol{a}||_2^2$

4. Experimental Evaluation

In this section, we present experimental results for the OWA model[1] proposed in Section 3.3. First, as a sanity check and to investigate the convergence behavior of the learning process, we conduct experiments with synthetic data, for which the data-generating process can be controlled. Then, to show that our approach can be used successfully for a practical problem, we apply it in a case study on predicting classifier ensemble performance.

As discussed before, we are not aware of existing learning methods for the problem of aggregation/disaggregation as stated in this paper. As a baseline, we therefore use the approach that is arguably most obvious in this setting, namely a linear (additive) model. This model defines the overall score y_i of a composition $\boldsymbol{c}_i$ by the sum of the local scores of the constituents involved. More specifically, to account for the variability in the size of compositions, the sum of local scores is normalized by n_i. Thus, we formally arrive at the following normalized linear model (NLM):

$$y_i = \frac{1}{n_i} \sum_{j=1}^{M} \beta_j \, \mathbb{I}\{o_j \in \boldsymbol{c}_i\} + \epsilon_i \ , \tag{11}$$

where $\mathbb{I}$ denotes the indicator function. Since the model is linear in the constituent scores β_j, these coefficients can be estimated easily by standard least squares regression. We deliberately omit an intercept, because the model is also shift-invariant. Interestingly, (11) can be obtained as a special case of the OWA model (6) for $\lambda_1 = \ldots = \lambda_n = 1/n$.

4.1. Synthetic Data

In the first experiment, we assume data to be produced according to our model (6) with normally distributed noise[2] (zero expectation and standard deviation $\sigma \in \{0, 0.1, 0.3, 0.5\}$). We generate a set of $M = 100$ constituent scores as random numbers in $[0, 1]$. These scores are fixed and do not change throughout the experiment. In the second step, 1000 compositions are generated uniformly at random by sampling without replacement. In the last step, we randomly generate 100 OWA weight vectors $\boldsymbol{\lambda}$ of size $n = 10$, such that every λ_i is chosen independently from the unit interval, and the vector $\boldsymbol{\lambda}$ is then rescaled. The hyper-parameters of the OWA model are set to $k = 2$, $m = 21$ and $\gamma = 10^{-5}$.

1. Python implementation is available at https://github.com/v-melnikov/LTA-OWA
2. Due to the noise, it may happen that y_i falls outside the range $[0, 1]$. In this case, it is clamped at the boundary points.

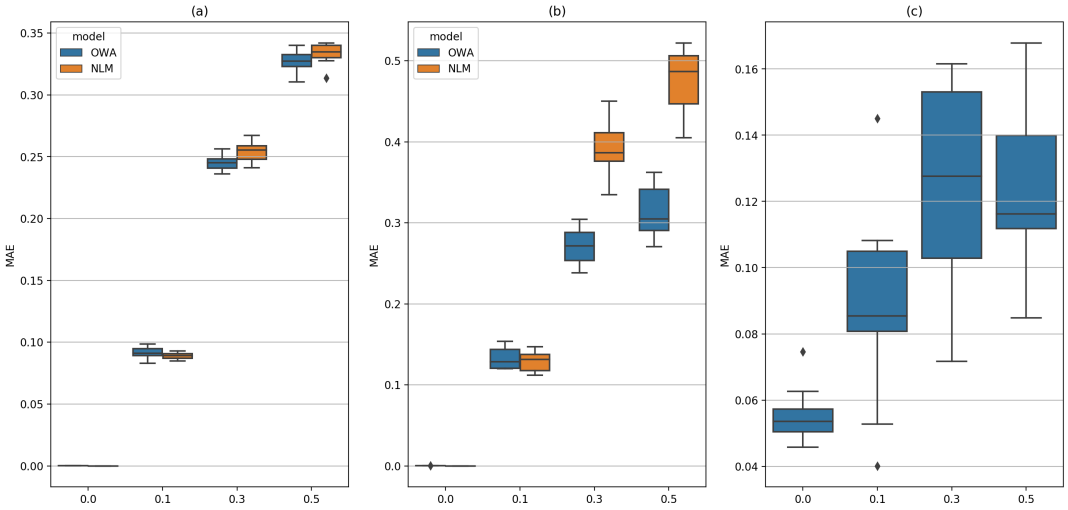

Figure 2: Mean absolute error in estimation of the ground truth parameters of the data-generating process for different noise levels σ: (a) composition scores, (b) constituent scores, (c) OWA weight vectors $\boldsymbol{\lambda}$.

Both models are trained on 500 compositions and tested on the remaining ones. Fig. 2 compares the model performance in terms of the parameter fit. The estimates of the composition scores are comparable across all noise levels, with a slight advantage in favor of the OWA model. More interesting is the comparison of the constituent score estimates. Here, the OWA model clearly outperforms the baseline, especially for higher noise levels — even then, the ground truth vectors $\boldsymbol{\lambda}$ are estimated quite well by OWA.

In the second experiment with synthetic data, we use the same setup as before, but additionally vary the OWA model hyper-parameters $k \in \{1, 2, 3\}$ (ranging from piecewise constant to cubic splines) and $m \in \{5, 10, 15, 20\}$. Again, we measure the mean absolute error of the estimated composition scores and the parameters of the data-generating process. The results in Fig. 3 suggest some trends, which appear to be stable across all considered scenarios. For all three quantities, the estimation error increases with increasing noise. However, like in the first experiment, we observe a saturation of the OWA weight error, which is independent of the model hyper-parameters m and k.

No performance gain can be observed on our dataset when increasing the order of splines beyond $k = 2$. As expected, a piecewise linear BUM function ($k = 2$) is flexible enough to approximate all OWA operators. Interestingly, even a piecewise constant BUM ($k = 1$) has a competitive performance in cases with high noise for the composition and constituent scores. The effect of the number of spline knots in the unit interval (m) is less obvious. Although smaller values of $m < n$ lead to a drop in performance and higher variance for all types of error, no significant performance increase can be observed if $m \gg n$. Based on this experiment, we recommend the following default hyper-parameters for the OWA model: $k = 2$ and $m =$ expected number of constituents (provided this information is available).

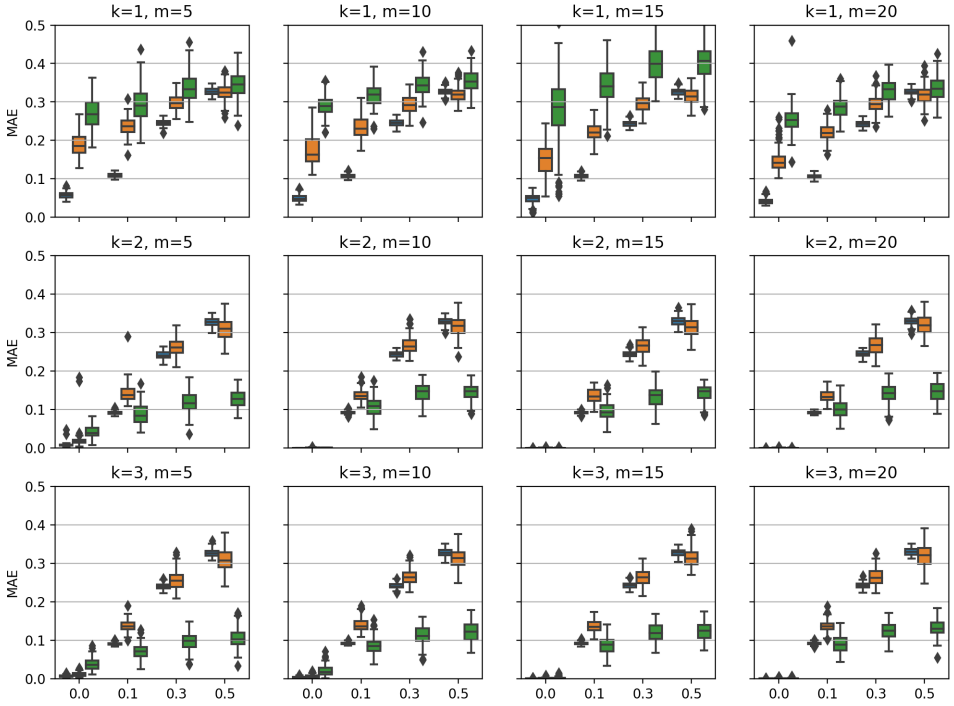

Figure 3: Mean absolute error in estimation of the ground truth parameters of the data-generating process for different noise levels σ and different combinations of m and k: composition scores (blue), constituent scores (yellow), and OWA weight vectors (green).

4.2. Case Study: Predicting Classifier Ensemble Performance

In so-called "ensemble pruning", the goal is to select an optimal subset of members of an ensemble of classifiers. Zhou (2012) indeed shows that, in ensemble learning, a properly chosen subset of a set of individual learners can indeed perform better than the entire set ("*many could be better than all*"). Motivated by this observation, we investigate the problem of predicting the performance of an ensemble of classifiers (which, if successful, may support the search for an optimal subset).

We generate the ensemble meta-dataset using five different types of learners: Gaussian Naïve Bayes, CART decision tree, k-Nearest Neighbors, stochastic gradient descent (SGD) with hinge loss (i.e., linear SVM), and SGD with log-loss (i.e., logistic regression). The last two learners are extended to the multi-class case using a one-vs-rest decomposition. In total, we generate 61 individual models by randomly instantiating the hyper-parameters (and the random seed) of all learners except for Naïve Bayes. The classifier ensembles are chosen randomly without replacement. The ensemble sizes vary between 3 and 21 (only odd numbers). In total, 1000 ensembles following the majority vote decision rule are generated. The ensembles are evaluated on well-known multi-class benchmark datasets. The detailed

Table 1: Mean absolute error (MAE) and Kendall's τ for both models (mean and standard deviation over 10 random shuffled folds).

name (id)	MAE_{OWA}	MAE_{NLM}	τ_{OWA}	τ_{NLM}
LED (40496)	0.023 ± 0.001	0.053 ± 0.002	0.57 ± 0.019	0.472 ± 0.026
ecoli (39)	0.021 ± 0.001	0.044 ± 0.001	0.826 ± 0.012	0.74 ± 0.017
glass (41)	0.035 ± 0.001	0.046 ± 0.001	0.636 ± 0.018	0.572 ± 0.032
kr-vs-k (1481)	0.013 ± 0.001	0.017 ± 0.001	0.755 ± 0.023	0.728 ± 0.016
letter (6)	0.052 ± 0.003	0.079 ± 0.002	0.69 ± 0.022	0.552 ± 0.035
mfeat-factors (12)	0.016 ± 0.003	0.034 ± 0.006	0.273 ± 0.049	0.139 ± 0.05
mfeat-fourier (14)	0.035 ± 0.002	0.066 ± 0.002	0.633 ± 0.025	0.472 ± 0.03
mfeat-karhunen (16)	0.018 ± 0.002	0.036 ± 0.004	0.382 ± 0.047	0.266 ± 0.055
mfeat-morphological (18)	0.045 ± 0.002	0.061 ± 0.002	0.585 ± 0.024	0.465 ± 0.039
mfeat-pixel (20)	0.018 ± 0.002	0.036 ± 0.004	0.368 ± 0.057	0.228 ± 0.05
mfeat-zernike (22)	0.018 ± 0.002	0.032 ± 0.002	0.291 ± 0.071	0.201 ± 0.044
optdigits (28)	0.018 ± 0.003	0.035 ± 0.003	0.375 ± 0.071	0.261 ± 0.044
page-blocks (30)	0.005 ± 0.001	0.004 ± 0.0	0.334 ± 0.045	0.484 ± 0.024
pendigits (32)	0.021 ± 0.003	0.034 ± 0.003	0.555 ± 0.077	0.409 ± 0.043
satimage (182)	0.012 ± 0.001	0.017 ± 0.001	0.821 ± 0.012	0.72 ± 0.021
segment (36)	0.023 ± 0.005	0.033 ± 0.001	0.571 ± 0.111	0.513 ± 0.028
zoo (62)	0.031 ± 0.003	0.084 ± 0.002	0.716 ± 0.018	0.621 ± 0.013
mean	0.024 ± 0.002	0.042 ± 0.002	0.552 ± 0.041	0.461 ± 0.033

description of every dataset can be found in the OpenML[3] database using the corresponding dataset id provided in Table 1. On every dataset, we perform a 5-fold cross-validation and store the mean test accuracy as the composition score.

The performance comparison between the OWA model (with hyper-parameters $k = 2$, $m = 21$, $\gamma = 10^{-5}$) and the baseline NLM is provided in Table 1. To assess the prediction variance, a Monte Carlo cross-validation (50% train, 50% test) was repeated ten times. In addition to the mean absolute error (MAE), we also compare the predicted performance scores using Kendall's τ ranking metric (ranging between -1 and 1). This is motivated by the observation that, when searching for an optimal subset of classifiers, the relative comparison of ensembles (i.e., their ranking) might actually be more important than their comparison in terms of absolute performance differences. For both metrics, the OWA model outperforms the baseline on all datasets except *page-blocks*, where the MAE performance is comparable (although the baseline achieves slightly better ranking performance). The prediction variance is quite low and comparable for both models, suggesting that our OWA learning algorithm is stable.

To verify the plausibility of the learned OWA weights λ_i, Fig. 4 shows the distribution of these weights for an ensemble of size $n = 20$. For most datasets, the distribution is left skewed, i.e., the aggregation tends to be "maximum-like", giving higher weight to higher local scores. In the context of classifier ensembles, this is indeed quite plausible: the performance of an ensemble is typically worse than the best-performing ensemble member,

3. www.openml.org

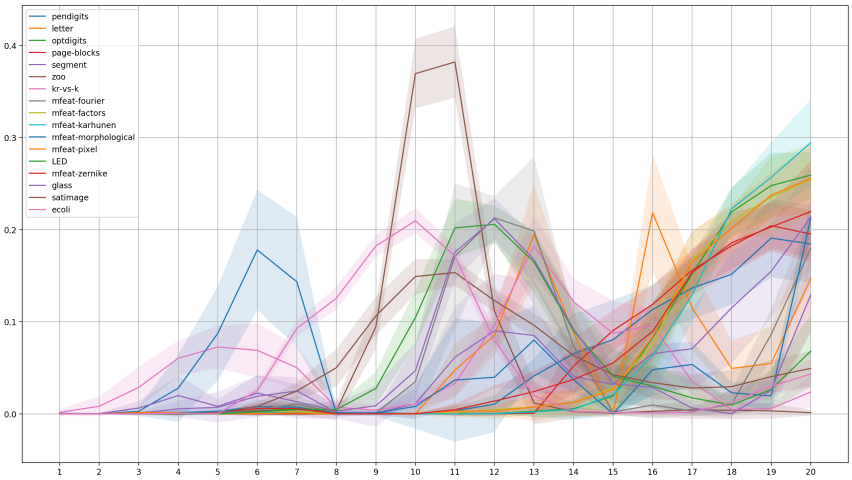

Figure 4: Mean and standard deviation (shaded) of the learned OWA weights instantiated for an ensemble of size $n = 20$.

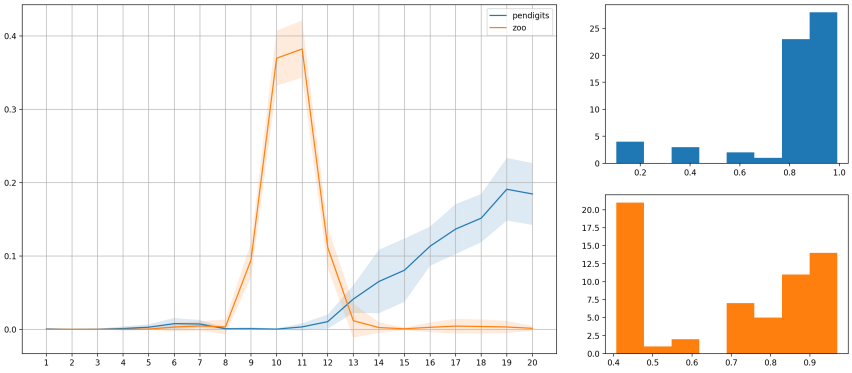

Figure 5: Learned OWA weights for the datasets *pendigits* and *zoo* compared to the ground truth performance of single classifiers on these datasets.

but better than the average. In Fig. 5, the OWA weights learned on the datasets *pendigits* and *zoo* are shown together with the ground truth performance of single ensemble members. For the first dataset, the majority of classifiers has a good accuracy. It is therefore enough to give the most weight to the ensemble members that are ranked higher (based on their estimated accuracy) and ignore the other ones. For the second dataset, however, there is a large group of classifiers with low accuracy. In this case, it is more reasonable to weight the middle-ranked ensemble members the most, since their performance will be decisive for the overall performance of the ensemble.

5. Summary and Outlook

We tackled the "aggregation/disaggregation problem", a novel type of machine learning problem, for the case of the OWA operator, and developed a learning algorithm specifically tailored for this task. First experimental results on synthetic data as well a practical case study on predicting the performance of classifier ensembles are very encouraging.

One promising direction for future work is to develop a customized learning algorithm for the OWA model. An efficient algorithm can take advantage of a differentiable approximation of the sorting function, like the one recently proposed by Cuturi et al. (2019).

Also interesting is a generalization of our model toward the description of constituents in terms of feature vectors $\boldsymbol{v}_{i,j}$. In this case, the local scores $y_{i,j}$ would no longer be constants, but instead be expressed as functions $f_{\boldsymbol{\theta}}(\boldsymbol{v}_{i,j})$, where $\boldsymbol{\theta}$ is another parameter to be learned. An extension of that kind will also enable the handling of a potentially infinite set of constituents (instead of a restricted set of finite size M).

Acknowledgments

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center On-The-Fly Computing (GZ: SFB 901/3) under the project number 160364472.

References

M. Aggarwal and A. Fallah Tehrani. Modelling human decision behaviour with preference learning. *INFORMS Journal on Computing*, 31(2), 2019.

G. Beliakov. Shape preserving approximation using least squares splines. *Approximation Theory and its Applications*, 16(4):80, 2000.

G. Beliakov. Monotone approximation of aggregation operators using least squares splines. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(6), 2002.

G. Beliakov. Learning weights in the generalized OWA operators. *Fuzzy Optimization and Decision Making*, 4(2), 2005.

G. Beliakov, T. Calvo, and S. James. Aggregation of preferences in recommender systems. In *Recommender Systems Handbook*. Springer US, Boston, MA, 2011.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

M. Cuturi, O. Teboul, and J. Vert. Differentiable sorting using optimal transport: The sinkhorn CDF and quantile operator. *CoRR*, abs/1905.11885, 2019.

C. De Boor. *A practical guide to splines*. Applied mathematical sciences. Springer, 2001.

C. Dias, J. Bueno, E. Borges, G. Lucca, H. Santos, G. Dimuro, H. Bustince, P. Drews, S. Botelho, and E. Palmeira. Simulating the behaviour of choquet-like (pre) aggregation

functions for image resizing in the pooling layer of deep learning networks. In *Fuzzy Techniques: Theory and Applications*. Springer International Publishing, 2019.

D. Dubois and H. Prade. Criteria aggregation and ranking of alternatives in the framework of fuzzy set theory. *Fuzzy Sets and Decision Analysis*, 20, 1984.

D. Filev and R. R. Yager. Learning OWA operator weights from data. In *Proceedings of 1994 IEEE 3rd Int. Fuzzy Systems Conference*, 1994.

M. Grabisch, J. Marichal, R. Mesiar, and E. Pap. *Aggregation Functions*. Cambridge University Press, 2009.

A. Grover, E. Wang, A. Zweig, and S. Ermon. Stochastic optimization of sorting networks via continuous relaxations. *CoRR*, abs/1903.08850, 2019.

M. Guo, Q. Zhang, X. Liao, and Y. Chen. An interpretable machine learning framework for modelling human decision behavior. *CoRR*, abs/1906.01233, 2019.

M.A. Islam, D. T. Anderson, A. J. Pinar, and Timothy C. Havens. Data-driven compression and efficient learning of the choquet integral. *IEEE Trans. on Fuzzy Systems*, 26(4), 2018.

E. Jacquet-Lagreze and Y. Siskosb. Preference disaggregation: 20 years of MCDA experience. *European Journal of Operational Research*, 130(2):233–245, 2001.

V. Melnikov and E. Hüllermeier. Learning to aggregate using uninorms. In *Machine Learning and Knowledge Discovery in Databases*, 2016.

O. Sobrie, V. Mousseau, and M. Pirlot. Learning a majority rule model from large sets of assignment examples. In *Algorithmic Decision Theory*, 2013.

O. Sobrie, V. Mousseau, and M. Pirlot. Learning the parameters of a non compensatory sorting model. In *Proc. ADT, 4th Int. Conf. on Algorithmic Decision Theory*, 2015.

A. Fallah Tehrani, W. Cheng, K. Dembczynski, and E. Hüllermeier. Learning monotone nonlinear models using the Choquet integral. *Machine Learning*, 89(1):183–211, 2012a.

A. Fallah Tehrani, W. Cheng, and E. Hüllermeier. Preference learning using the Choquet integral: The case of multipartite ranking. *IEEE Tran. on Fuzzy Syst.*, 20(6), 2012b.

R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. on Systems, Man, and Cybernetics*, 18(1), 1988.

R. R. Yager. Quantifier guided aggregation using OWA operators. *Int. Journal of Intelligent Systems*, 11(1):49–73, 1996.

R. R. Yager. Generalized OWA aggregation operators. *Fuzzy Optimization and Decision Making*, 3(1), 2004.

Z. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 2012.