Learning to Aggregate Using Uninorms

Vitalik Melnikov and Eyke Hüllermeier(✉)

Department of Computer Science, Paderborn University, Paderborn, Germany
melnikov@mail.upb.de, eyke@upb.de

Abstract. In this paper, we propose a framework for a class of learning problems that we refer to as "learning to aggregate". Roughly, learning-to-aggregate problems are supervised machine 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. Our learning-to-aggregate framework establishes a close connection between machine learning and a branch of mathematics devoted to the systematic study of aggregation functions. We specifically focus on a class of functions called uninorms, which combine conjunctive and disjunctive modes of aggregation. Experimental results for a corresponding model are presented for a review data set, for which the aggregation problem consists of combining different reviewer opinions about a paper into an overall decision of acceptance or rejection.

1 Introduction

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 $f : \mathcal{X} \longrightarrow \mathcal{Y}$. Alternatively, instead of modeling dependencies with a deterministic function, a model may correspond to a probability distribution on $\mathcal{X} \times \mathcal{Y}$.

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

P. Frasconi et al. (Eds.): ECML PKDD 2016, Part II, LNAI 9852, pp. 756–771, 2016.
DOI: 10.1007/978-3-319-46227-1_47

less appropriate. This paper is devoted to one such class of problems that we refer to as *aggregation problems*. 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. Thus, the *learning-to-aggregate* framework we envision establishes a close connection between machine learning and a branch of mathematics devoted to the systematic study of aggregation functions [10].

Needless to say, the idea of aggregation is not new to machine learning. On the contrary, aggregation problems seem to abound in this field and appear in various guises; for example, combining the information of the neighbors in nearest neighbor estimation, the predictions of base learners in stacking, etc., can all be seen as specific types of aggregation problems. Yet, to the best of our knowledge, a common framework of learning-to-aggregate has not been proposed so far. We believe that such a framework, and the specific view on learning problems it comes along with, is useful for different reasons. In particular, it allows for looking at different learning problems as specific instances of the same problem class, thereby connecting and cross-fertilizing subfields that would otherwise remain separated. Moreover, it may of course motivate new learning problems and trigger the development of novel methods.

The remainder of the paper is organized as follows. In the next section, we outline our learning-to-aggregate framework. The description of this framework is completed in Sect. 3, which is devoted to a discussion of aggregation functions. In Sect. 4, we propose a specific instance of the framework, namely a model for learning to aggregate based on so-called uninorms. Related work is briefly reviewed in Sect. 5. Finally, to illustrate our approach, some experiments on a data set consisting of reviews on papers submitted the the ECML/PKDD 2014 conference are presented in Sect. 6, prior to concluding the paper in Sect. 7.

2 Learning to Aggregate

In this section, we introduce a formal framework of learning-to-aggregate and elaborate on some of its properties. Prior to doing so, we give a simple example that already highlights important aspects of aggregation problems as well as limitations of standard vectorial (feature-based) representations in this context.

2.1 A Simple Example

Suppose compositions $\boldsymbol{c}$ are multisets (*bags*) of real numbers from the unit interval, such as $\{0.8, 0.7\}$ or $\{0.2, 0.6, 0.3\}$. Moreover, suppose the output y associated with a composition $\boldsymbol{c} \subset [0, 1]$ is an aggregation of the constituents; to be concrete, consider the product as an example. The goal of the learner is to induce the dependency between inputs $\boldsymbol{c}$ and outputs y based on corresponding training examples, such as $(\boldsymbol{c}, y) = (\{0.8, 0.7\}, 0.56)$.

Although this toy example is actually very simple, tackling it with standard machine learning methods is non-trivial. As one important reason, note that, in contrast to a feature vector of fixed dimension, compositions are of variable length. In fact, the sought dependency is a mapping of the form $\mathcal{X} \longrightarrow \mathcal{Y}$, with the instance space

$$\mathcal{X} = \bigcup_{n \in \mathbb{N}} \mathcal{Y}^n \tag{1}$$

and $\mathcal{Y} = [0, 1]$ in our case. This instance space is a union of spaces of finite dimension but does not have a finite dimension itself; indeed, in our example, we allow compositions $\boldsymbol{c}$ of any size. It is thus neither clear how to define a suitable hypothesis space on $\mathcal{X}$, i.e., a set of functions with domain $\mathcal{X}$, nor how to learn in this space.

To make the problem amenable to standard methods, it is of course possible to map compositions $\boldsymbol{c}$ to feature vectors $\boldsymbol{x} = (x_1, \ldots, x_d) = (f_1(\boldsymbol{c}), \ldots, f_d(\boldsymbol{c}))$ of finite length, on which a model of the form $y = f(\boldsymbol{x})$ could then be learned; in fact, this is a common approach to dealing with structured data objects, which are given as bags in our case but could also be sequences or graphs, for example. Like in learning on structured objects in general, the success of this approach strongly hinges on the definition of the right features. In our example, features would be needed that allow for reconstructing, for any bag of numbers, the product of these numbers. Making sure that such features are available arguably presumes that the dependency between $\boldsymbol{c}$ and y is already known.

2.2 Formal Setting and Notation

We proceed from a set of training data

$$\mathcal{D} = \big\{(\boldsymbol{c}_1, y_1), \ldots, (\boldsymbol{c}_N, y_N)\big\} \subset \mathcal{C} \times \mathcal{Y}, \tag{2}$$

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:

- 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).

- 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.
- A *quantity* $q_{i,j} \in \mathbb{R}_+$ representing the amount of the constituent in the composition (instead of simply informing about the presence or absence of the constituent).
- A *local evaluation* in the form of a score $y_{i,j} \in \mathbb{R}_+$.

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 $\mathcal{D}$ (but without knowledge of the data-generating process, i.e., the joint probability distribution $\mathbf{P}$ generating composition/score tuples $(\boldsymbol{c}, y)$).

2.3 Learning Aggregation Functions

Our simple example in Sect. 2.1 already illustrates one of the key problems in learning-to-aggregate, namely the combination of a variable number of scores $y_{i,j}$, pertaining to evaluations of the constituents $c_{i,j}$ in a composition $\boldsymbol{c}$, into a single score y_i. In Fig. 1, which provides an overview of our setting, this step corresponds to the part marked by the dashed rectangle.

Now, suppose that we know, or can at least reasonably assume, that y_i is obtained from $y_{i,1}, \ldots, y_{i,n_i}$ through an aggregation process defined by a binary aggregation function $A : \mathcal{Y}^2 \longrightarrow \mathcal{Y}$:

$$y_i = A\Big(\ldots A\big(A(y_{i,1}, y_{i,2}), y_{i,3}\big), \ldots, y_{i,n_i}\Big)$$

In the simplest case, where the constituents do not have labels and hence cannot be distinguished, the aggregation should be invariant against permutation of the constituents in the bag. Thus, it is reasonable to assume A to be associative and symmetric. Besides, one may of course restrict an underlying class of candidate functions $\mathcal{A}$ by additional assumptions. In our example, for instance, we may know that the aggregation is monotone decreasing.

Starting from a class $\mathcal{A}$ of aggregation functions, instead of a hypothesis space $\mathcal{H}$ on the instance space (1) directly, has at least two important advantages.

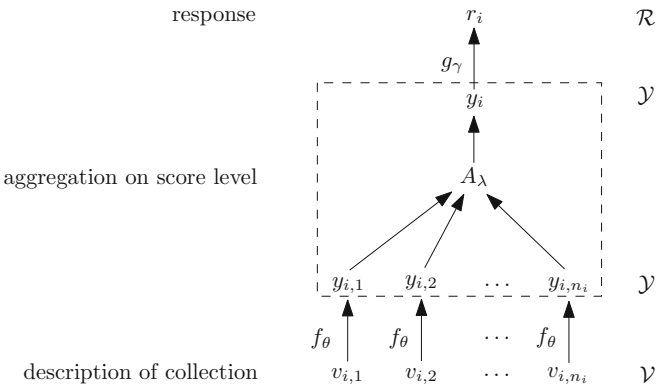

Fig. 1. Illustration of a basic version of the learning-to-aggregate model.

First, as just said, it allows for incorporating prior knowledge about the aggregation, which may serve as a suitable inductive bias of the learning process. Second, it naturally solves the problem that hypotheses $h \in \mathcal{H}$ must accept inputs of any size. Indeed, under the assumption of associativity and symmetry, a binary aggregation function A is naturally extended to any arity, and can hence be used as a "generator" of a hypothesis $h = h_A$:

$$h(y_1, \ldots, y_n) = A^{(n)}(y_1, \ldots, y_n) = A\big(A^{(n-1)}(y_1, \ldots, y_{n-1}), y_n\big)$$

for all $n \geq 1$, where $h(y_1) = A^{(1)}(y_1) = y_1$ by definition.

For these reasons, we consider the learning of (binary) aggregation functions, and related to this the specification of a suitable class $\mathcal{A}$ of candidates, as an integral part of learning-to-aggregate. In Sect. 3, such classes and different types of aggregation functions will be discussed in more detail. Before doing so, we elaborate on some extensions of our learning-to-aggregate setting.

2.4 Disaggregation

The aggregation we have been speaking about so far is an aggregation on the level of scores. Thus, we actually assume that local scores $y_{i,j}$ of the constituents $c_{i,j}$ are already given, and that we are interested in aggregating them into an overall score y_i of the composition $\boldsymbol{c}_i$. This is indeed the genuine purpose of aggregation functions, which typically assume that all scores are elements of the same scale $\mathcal{Y}$. For example, we 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.

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 $\boldsymbol{v}_{i,j} \in \mathcal{V}$ (and perhaps quantities $q_{i,j}$, which we subsequently

ignore for simplicity). 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 parameterized by a parameter vector θ, and the aggregation function A by a parameter λ. The model is then of the form

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

and the problem consists of learning both the aggregation function A, i.e., the parameter λ, and the mapping from features to local scores, i.e., the parameter θ, simultaneously. Here, supervision only takes place on 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.

The decomposition of global scores into several local scores is sometimes referred to as *disaggregation* (because it inverts the direction of aggregation, which is from local scores to global ones). For example, suppose we observe a user's ratings of different playlists, each one considered as a collections 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 parameterizations imply different models (or, more formally, whether $(\lambda, \theta) \neq (\lambda', \theta')$ implies that the corresponding models assign different scores $y_i \neq y_i'$ for at least one composition).

2.5 Further Extensions

Sometimes, not even the (aggregate) scores y_i can be observed directly, but only certain response values $r_i \in \mathcal{R}$ related to these scores, i.e., training data is of the form

$$\mathcal{D} = \big\{(\boldsymbol{c}_1, r_1), \ldots, (\boldsymbol{c}_N, r_N)\big\} \subset \mathcal{C} \times \mathcal{R}, \tag{3}$$

For example, in the case of the playlist, direct feedback of the user might not be available. Instead, it might only be possible to observe a user's behavior, e.g., how long she listens to the playlist, or whether or not she decides to buy it. The response must then be modeled by another link function g (parameterized by γ), for example a discrete choice model like logit, which assumes $\mathcal{R} = \{0, 1\}$ and models the probability of a positive response according to $\mathbf{P}(r_i = 1) = (1+\exp(-\gamma_1(y_i - \gamma_2)))^{-1}$. The model discussed so far, including indirect feedback in the form of a response, is summarized and illustrated graphically in Fig. 1.

Instead of absolute feedback in the form of a (binary) response, one may also assume relative feedback in the form of pairwise comparisons $\boldsymbol{c}_i \succ \boldsymbol{c}_j$ between compositions, suggesting that $\boldsymbol{c}_i$ is preferred to $\boldsymbol{c}_j$ (and hence that y_i is larger than y_j). This type of feedback and corresponding training data

$$\mathcal{D} = \left\{ \boldsymbol{c}_{i(1)} \succ \boldsymbol{c}_{j(1)}, \ldots, \boldsymbol{c}_{i(N)} \succ \boldsymbol{c}_{j(N)} \right\} \subset \mathcal{C} \times \mathcal{C}, \tag{4}$$

is especially interesting from the point of view of preference learning [9]. Model induction could then be based, for example, on discrete choice models like Bradley-Terrey [1].

Further extensions of the model are possible thanks to additional information provided about the constituents or structural information about the composition (cf. Sect. 2.2). In particular, the aggregation step can be generalized in the case where a label is assigned to the constituents. For example, we may assume that a user first rates the appetizer, main dish, and dessert (each of which may consist of several dishes) separately, and then aggregates the corresponding scores into an overall rating. Note that, since the intermediate scores are now associated with roles, the last aggregation step does not necessarily need to be invariant against permutation (for example, the user may give a higher weight to the main dish and a lower one to the starter), so that a larger class of aggregation functions could be used.

2.6 Learning Problems

Even in its basic form shown in Fig. 1, our learning-to-aggregate 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. In the most general case, compositions are of different size, and training data consists of properties of constituents together with a corresponding response. Then, the learning problem essentially comes down to estimating the full set of parameters $(\gamma, \lambda, \theta)$.

Learning becomes simpler for various special cases. For example, if scores are observed directly (i.e., $r_i = y_i$), the link from scores to responses, specified by g_γ, does not need to be learned (or, stated differently, g can be taken as the identity). The case where individual scores $y_{i,j}$ are observed, too, is often considered in decision analysis and related fields [12,26], typically even with the assumption that each individual score corresponds to a *criterion* (which, in our terminology, means that it has a unique label, 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.

3 Aggregation Functions

Aggregation functions have been studied intensively as a branch of applied mathematics; we refer to the monograph [10] for a comprehensive treatment

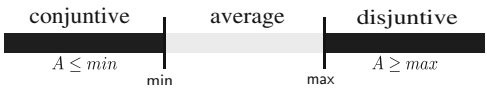

Fig. 2. Aggregation functions: conjunctive, disjunctive, and generalized averages.

of the topic. Roughly speaking, the purpose of an aggregation function operating on a scale $\mathcal{Y}$ is to combine values $y_1, \ldots, y_n \in \mathcal{Y}$ into another value y on the same scale. Typically, $\mathcal{Y}$ is taken as the unit interval $[0, 1]$; this is not a strong restriction, since aggregation functions on other domains can be studied via suitable transformations in the form of monotone bijections [11].

The study of aggregation functions is of axiomatic nature and proceeds from specific properties such functions should obey. Natural requirements, for example, include properties like symmetry (the result of the aggregation should not depend on the order of the values) and monotonicity. Especially interesting are binary aggregation functions A in the form of associative and commutative $[0, 1]^2 \longrightarrow [0, 1]$ mappings, because, as already said, these can be extended to n-ary aggregation functions in a canonical way:

$$A^{(n)}(y_1, \ldots, y_n) = A\big(A^{(n-1)}(y_1, \ldots, y_{n-1}), y_n\big),$$

where $A^{(1)}(y_1) = y_1$. One can then simply identify A with the family of functions thus defined, and write $A(y_1, \ldots, y_n)$ for any number n of arguments.

A natural order on (binary) aggregation functions is defined as follows: $A \leq B$ if $A(y_1, y_2) \leq B(y_1, y_2)$ for all $y_1, y_2 \in [0, 1]$. Based on this order relation, three important classes of aggregation functions are often distinguished: conjunctive, disjunctive, and generalized averaging operators. An aggregation A is called conjunctive if $A \leq \min$ and disjunctive if $A \geq \max$; all aggregations in-between min and max are called (generalized) averaging operators (see Fig. 2).

3.1 Conjunctive and Disjunctive Aggregation

In this paper, we are specifically interested in conjunctive and disjunctive aggregation, that is, aggregation functions that can be seen, respectively, as generalizations of the classical logical conjunction and disjunction. Important classes of such functions are given by the so-called t-norms and t-conorms [16].

Triangular norms (t-norms), which emerged in the context of probabilistic metric spaces [21], play a central role is many-valued and fuzzy logic, where they are used to generalize the logical conjunction [13]. A t-norm T is a monotone increasing, associative and commutative $[0, 1]^2 \longrightarrow [0, 1]$ mapping with neutral element 1 and absorbing element 0. Important examples include the minimum, which is the largest among all t-norms, the product $T(a, b) = ab$, and the Lukasiewicz t-norm $T(a, b) = \min(a + b, 1)$.

A t-conorm S is a monotone increasing, associative and commutative mapping $[0, 1]^2 \longrightarrow [0, 1]$ with neutral element 0 and absorbing element 1. These operators are dual to t-norms in the sense that, if T is a t-norm, then S defined

by $S(a,b) = 1 - T(1-a, 1-b)$ is a t-conorm. Important examples include the maximum, which is the smallest among all t-conorms, the algebraic sum $S(a,b) = a + b - ab$, and the Lukasiewicz t-conorm $S(a,b) = \max(a + b - 1, 0)$.

3.2 Uninorms

Generalized conjunctions and disjunctions share the properties of being monotone, associative and commutative, and actually only differ in their neutral element, which is 1 for the former and 0 for the latter. The location of the neutral element in the unit interval is also reflected by the characteristics of these two types of operators: For t-norms, the overall aggregation remains unchanged only when adding the highest value 1, i.e., $T(y_1, \ldots, y_n) = T(y_1, \ldots, y_n, y_{n+1})$ only if $y_{n+1} = 1$; otherwise, the overall aggregation can only decrease. Thus, t-norms aggregate very strictly and are fully non-compensatory: it is not possible to compensate for low evaluations by adding high ones. The dual class of t-conorms behaves in exactly the opposite way: aggregation via t-conorms is fully compensatory.

One may wonder whether a neutral behavior is only possible with respect to 0 and 1, or perhaps also some other value $e \in (0,1)$. Is there is a class of aggregation functions that shares the properties of t-norms and t-conorms, except for having an arbitrary value e as neutral element? This question is answered affirmatively by the class of so-called *uninorms* [27]. A uninorm U is a monotone increasing, associative and commutative $[0,1]^2 \longrightarrow [0,1]$ mapping with neutral element $e \in (0,1)$, i.e., such that $U(a,e) = U(e,a) = a$ for all $a \in [0,1]$.

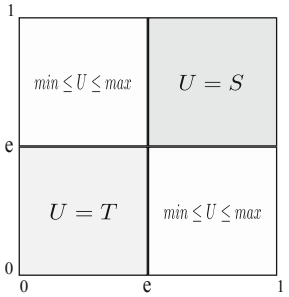

Fig. 3. Structure of a uninorm.

Uninorms U can be shown to have a specific structure: For arguments exceeding e, they behave like a t-conorm, i.e., there is a t-conorm S such that $U(a,b) = S(a,b)$ for all $a, b \in [e,1]^2$. Likewise, for arguments below e, they behave like a t-norm: $U(a,b) = T(a,b)$ for all $(a,b) \in [0,e]$. On the remaining part of the unit square $[0,1]^2$, U can be completed in different ways, though always remaining between the minimum and the maximum; see Fig. 3 for

an illustration. A concrete family of uninorms called min-uninorms is constructed from a t-norm T and a t-conorm S as follows:

$$U_e(a,b) = \begin{cases} e\,T\left(\frac{a}{e}, \frac{b}{e}\right) & \text{if } a,b \in [0,e] \\ e + (1-e)S\left(\frac{a-e}{1-e}, \frac{b-e}{1-e}\right) & \text{if } a,b \in [e,1] \\ \min(a,b) & \text{otherwise} \end{cases} \tag{5}$$

3.3 Complex Aggregation

Basic aggregation functions like those discussed above can be combined into more complex ones, for example in a hierarchical way [22,23]. An example is shown graphically in Fig. 4: The output produced by one aggregation serves as an input of another one on a higher level. In the particular example shown, the aggregation function is of the form

$$A_{\lambda_t,\lambda_s} : [0,1]^4 \longrightarrow [0,1],\ (y_1,y_2,y_3,y_4) \mapsto T_{\lambda_t}\big(S_{\lambda_s}(y_1,y_2), S_{\lambda_s}(y_3,y_4)\big), \tag{6}$$

and thanks to the logical interpretation of t-norms and t-conorms, A itself can be interpreted as a degree of truth of a generalized logical expression. For example, if y_1, y_2, y_3, y_4 correspond, respectively, to the evaluation of a job candidate on skills in math (M), programming (P), French (F), and Spanish (S), then A evaluates the expression $(M \wedge P) \vee (F \wedge S)$. In other words, a good candidate needs to be strong in math or programming, and also have good language skills, either in French or Spanish. Thus, there is no compensation between language and analytical skills, but full compensation within each of the two categories.

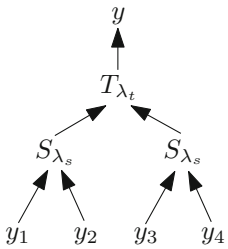

Fig. 4. Example of a complex (hierarchical) aggregation functions.

As shown by (6), complex aggregation functions typically assign different roles to different inputs. In our framework, this means that constituents must be identified by a label (such as M or P above). In principle, of course, structures more general than hierarchies (trees) could be used to design complex aggregation functions, for example directed acyclic graphs. Such structures appear to be especially useful in the case where the constituents $c_{i,j}$ in a composition $\boldsymbol{c}_i$ are equipped with a structure (i.e., $\boldsymbol{c}_i$ is not simply a bag). However, as extensions of this kind are beyond the scope of this paper, we refrain from a deeper discussion.

4 A Model Based on Uninorms

Suppose a component $\boldsymbol{c}_i$ is a multiset of constituents $c_{i,j}$ described in terms of feature vectors $\boldsymbol{v}_{i,j}$. We assume scores $y_{i,j} \in [0,1]$ to be of the form

$$y_{i,j} = f_\theta(\boldsymbol{v}_{i,j}) = \big(1 + \exp\big(-\theta^\top \boldsymbol{v}_{i,j}\big)\big)^{-1},$$

i.e., θ is a vector that assigns weights for the different entries in $\boldsymbol{v}_{i,j}$. The local scores $y_{i,j}$ are then aggregated using a uninorm U_λ parameterized by λ:

$$y_i = U_\lambda\big(\{y_{i,j}\}_{j=1}^{n_i}\big) = U_\lambda\big(y_{i,1}, \ldots, y_{i,n_i}\big)$$

Finally, the response r_i is a binary decision, for which

$$\mathbf{P}(r_i = 1) = \Big(1 + \exp\big(-\gamma_1(y_i - \gamma_2)\big)\Big)^{-1}.$$

Thus, the higher the score, the higher is the probability of a positive decision. More specifically, the probability of a positive decision is controlled by two parameters $\gamma = (\gamma_1, \gamma_2)$. The second parameter, $\gamma_2 \in [0,1]$, is a kind of aspiration level or ambition threshold, since $\mathbf{P}(r_i = 1) > 1/2$ for $y_i > \gamma_2$ and $\mathbf{P}(r_i = 1) < 1/2$ for $y_i < \gamma_2$. Moreover, $\gamma_1 \geq 0$ is a scaling parameter that models the precision with which decisions are made: For $\gamma_1 \to \infty$, decisions become deterministic, whereas for $\gamma_1 = 0$, decisions are made completely at random (i.e., without actually taking the score y_i into account).

Overall, we thus end up with a probabilistic model of the following form:

$$\mathbf{P}(r_i = 1) = \left(1 + \exp\left(-\gamma_1\left(U_\lambda\left(\left\{\big(1 + \exp\big(-\theta^\top \boldsymbol{v}_{i,j}\big)\big)^{-1}\right\}_{j=1}^{n_i}\right) - \gamma_2\right)\right)\right)^{-1} \tag{7}$$

Learning this model can be done using maximum likelihood estimation. Thus, given training data (4), the problem is to maximize the (regularized) log-likelihood function

$$L(\gamma, \lambda, \theta) = \sum_{i=1}^{N} \log\big(\mathbf{P}(r_i \,|\, \gamma, \lambda, \theta, \boldsymbol{c}_i)\big) - \alpha R(\gamma, \lambda, \theta), \tag{8}$$

where $\mathbf{P}(r_i \,|\, \gamma, \lambda, \theta, \boldsymbol{c}_i)$ is given by the expression on the right-hand side of (7) if $r_i = 1$ and by 1 minus this expression if $r_i = 0$, and $R(\gamma, \lambda, \theta)$ is a regularization term that is used to penalize large feature weights.

The above model simplifies in the case where the local scores $y_{i,j}$ are already given, i.e., training data is of the form (4):

$$\mathbf{P}(r_i = 1) = \left(1 + \exp\left(\gamma_1\left(U_\lambda\left(\{y_{i,j}\}_{j=1}^{n_i}\right) - \gamma_2\right)\right)\right)^{-1} \tag{9}$$

5 Related Work

As our framework is quite general, it has connections to various other branches of machine learning. These are either established by the non-standard representation of instances, or by the idea of using aggregation functions in one way or the other. This section is meant to point to some of the related fields, although space restrictions obviously prevent from a comprehensive discussion.

Compositions $\boldsymbol{c}_i$ can of course be seen as a specific types of structured objects, on which kernel functions can be defined; for example, kernel functions for "bags of feature vectors" have been studied in image processing and other fields [5]. Then, given such a kernel function, the large arsenal of kernel-based machine learning methods can be applied. Yet, an approach of that kind is not fully in line with our idea of learning to aggregate. First, kernel methods eventually produce a vectorial representation (in some feature space), which, for the reasons already mentioned, might not be fully appropriate. More importantly, they do not easily allow for incorporating knowledge about the process of aggregation, which is a key idea of our approach, nor do they lead to well interpretable models.

In the special case where compositions $\boldsymbol{c}_i$ are bags (i.e., multisets without additional structure) of feature vectors $\boldsymbol{v}_{i,j}$, our framework is similar to *multi-instance learning* (MIL) [2], especially with regard to the representation of data objects. Yet, there are also some notable differences. In MIL, for example, a bag is normally not viewed as a composition of constituents that belong together and form a whole; in the simplest case, one proceeds from a binary setting with positive and negative instances, and assumes a bag to be labeled positive as soon as it contains at least one positive instance. Correspondingly, aggregation over predictions for individual instances is done, either explicitly or implicitly, via the maximum (or generalizations like the noisy OR [14]), whereas less attention has been payed to a systematic study of the aggregation process.

Specific types of aggregation functions have attracted attention in machine learning in recent years. For example, copulas can be seen as a specific type of conjunctive aggregation that allows for combining marginal into joint probability distributions [6]. In preference learning, the so-called Choquet integral has been used as a generalization of the weighted average that is able to capture interactions between different variables [24]. Yet, these approaches still proceed from a feature representation of data objects.

There are other generalizations of supervised learning in which aggregation plays an important role. For example, in learning from aggregate outputs [18], the assumption is that output values cannot be observed for each training instance individually; instead, only an aggregation of these values is observed for sets of instances. Here, however, the aggregation function is supposed to be known.

As already mentioned, the scores assigned to a composition can often be interpreted as a kind of evaluation. Thus, there is also an obvious connection to the field of preference learning [9]. From the point of view of preference learning, a composition can be seen as a *bundle of goods*, to which a user assigns a degree of utility [25]. In comparison to learning preferences on items represented in terms of feature vectors, work on preference learning on bundles is still very scarce.

6 Illustration

As an illustration of our framework, we consider the problem of aggregating reviewer recommendations into an overall decision about the acceptance or rejection of a conference submission. Or, stated differently, we adopt a data-driven approach to modeling the way in which the program chairs of a conference aggregate different reviews of a paper into an overall decision.

To this end, we collected data about the reviewing process of ECML/PKDD 2014. More concretely, our data set consists of 481 submitted papers with corresponding reviews. While most papers have three reviews, there are also papers with two or four reviews. Each review consists of a rating of the originality and quality of the paper, an overall recommendation, and a level of confidence of the reviewer. The underlying scale comprises five categories (strong reject, weak reject, weak accept, accept, strong accept), which we embedded in the unit interval by mapping them to $\{0, 0.25, 0.5, 0.75, 1\}$. Finally, the decision of acceptance or rejection is known for each paper (with an acceptance rate of 23,9 %).

As already said, the problem we consider consists of learning to aggregate reviewer recommendations into a final decision. Here, a paper is modeled as a composition $\boldsymbol{c}_i$, the constituents of which consist of feature vectors $\boldsymbol{v}_{i,j}$ with values for originality, quality, and overall recommendation given by a reviewer, as well as his or her confidence (and an intercept). Moreover, the final decision is treated as a response (0 for rejection and 1 for acceptance).

We applied the model (7) introduced in Sect. 4 with two uninorms: The so-called 3-Π uninorm [3], and the uninorm (5) with the product t-norm $T(a, b) = ab$ and the dual t-conorm $S(a, b) = a + b - ab$; in the latter case, the parameter λ of U_λ is thus given by the neutral element e in (5). Note that a uninorm is a quite plausible aggregation function for this application: The neutral element e can be seen as kind of "borderline" recommendation. A recommendation better than e expresses a positive reviewer option and can only increase the probability of acceptance, whereas a recommendation worse than e has the opposite effect. For comparison, we also present results for purely conjunctive ($U_\lambda = \min$) and purely disjunctive aggregation ($U_\lambda = \max$).

To learn the parameters, we maximize the likelihood function (8) with L_2 regularization ($\alpha = 0.01$) using the L-BFGS-B algorithm [4]. To avoid local optima, we did 10 random restarts, choosing initial parameters according to Latin hypercube sampling [17] with 10 samples.

The problem considered in this study can in principle also be formalized in the setting of multi-instance (MI) learning: papers are considered as bags and the reviews as instances, represented in the form of feature vectors. Therefore, we also compare our method with several state-of-the-art MI algorithms [2].

A standard approach based on a feature representation of submissions does not appear meaningful. In fact, even if the number of reviewers would be the same for each paper, the order of reviewers should not play any role, i.e., the aggregation should be invariant against permutation (renumbering of the reviewers). For example, it does not make sense to give a higher weight to the first reviewer and a lower weight to the second one. Since all features are discrete, it is still

Table 1. Mean ± standard deviation for classification rate, AUC and F-measure.

Approach	Algorithm/Aggregation	Accuracy	AUC	F_1
Aggregation	3-Π uninorm	.921 ± .035	.974 ± .025	.823 ± .091
	min-uninorm	.890 ± .029	.949 ± .021	.767 ± .059
	minimum	.885 ± .045	.923 ± .034	.756 ± .100
	maximum	.831 ± .064	.903 ± .064	.568 ± .179
MIL	MILR [20]	.916 ± .038	.973 ± .017	.811 ± .097
	MIBoost [8]	.911 ± .037	.960 ± .027	.807 ± .087
	MISMO-PolyKernel [19]	.906 ± .041	.858 ± .070	.791 ± .099
	MISMO-RBFKernel [19]	.909 ± .041	.870 ± .067	.804 ± .095
	MIWrapper [7]	.880 ± .040	.955 ± .031	.664 ± .146
Feature	AdaBoostM1-Dec.Table	.858 ± .045	.892 ± .052	.683 ± .112
Vector	AdaBoostM1-Dec.Stumps	.873 ± .043	.906 ± .048	.742 ± .091
	Decision Table	.855 ± .045	.900 ± .049	.687 ± .123
	C4.5 (J48)	.856 ± .038	.860 ± .073	.671 ± .101
	KNN	.862 ± .038	.904 ± .045	.667 ± .109
	LBR	.857 ± .041	.924 ± .038	.731 ± .079
	RandomForest	.838 ± .043	.891 ± .048	.633 ± .109
	Logistic Regression	.872 ± .042	.911 ± .046	.738 ± .097
	SVM (SMO)	.868 ± .042	.773 ± .073	.675 ± .122

possible to create a vector representation, simply by counting, for each feature value, its total number of occurrences in all reviews. Obviously, this transformation comes with a loss of information, since the reviews are merged and cannot be distinguished anymore. Nevertheless, we used it as another baseline (with several standard learning methods implemented in WEKA [15]).

All performance measures were estimated using 10-fold cross validation repeated 10 times. The mean values and standard deviations of classification rate, AUC, and F-measure are reported in Table 1. As can be seen, our approach compares quite favorably with the baselines. Moreover, the estimated model appears to be quite plausible. For example, the parameter γ_2, which plays the role of an acceptance threshold, equals (on average) 0.687; moreover, $\gamma_1 \approx 8$, which means that the reviewer recommendations determine decisions quite precisely. The vector θ has a plausible interpretation too: the overall recommendation has the highest influence, with a relative importance of about 0.78, followed by originality and quality with around 0.11 and 0.09, respectively.

7 Summary and Conclusion

The learning-to-aggregate framework introduced in this paper is meant to provide a basis for learning (predictive) models in which aggregation plays an

integral role. We believe that, first, there are many applications of this kind of modeling, and second, that machine learning can strongly benefit from the large repertoire of existing work on aggregation functions in the mathematical literature. More specifically, we argue that this field offers interesting mathematical tools for constructing model classes, thereby helping to learn models that are not only accurate but also interpretable, as well as important theoretical insights about aggregation functions and their properties, thereby supporting the design of efficient learning algorithms.

We illustrated our framework by looking at one of its particular instances and applying that instance on a review data set, where the aggregation problem consists of combining a (variable) number of reviews of a paper submission into a final decision of acceptance or rejection. While this is only a specific example, we look forward to developing the learning-to-aggregate framework both more broadly and more deeply in future work. As explained in Sect. 2, various learning problems can be defined based on the representation of compositions, assumptions about the aggregation process, and the type of training data to learn from. Developing and analyzing learning-to-rank methods for concrete, practically relevant settings is a major goal of our future work.

Acknowledgments. We thank Pritha Gupta and Karlson Pfannschmidt for their helpful suggestions. This work is part of the Collaborative Research Center "On-the-Fly Computing", which is supported by the German Research Foundation (DFG).

References

1. Alvo, M., Yu, P.: Statistical Methods for Ranking Data. Springer, New York (2014)
2. Amores, J.: Multiple instance classification: review, taxonomy and comparative study. Artif. Intell. **201**, 81–105 (2013)
3. Beliakov, G., Calvo, T., James, S.: Aggregation of preferences in recommender systems. In: Recommender Systems Handbook, pp. 705–734. Springer, US (2011)
4. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**(5), 1190–1208 (1995)
5. Csurka, G., Dance, C.R., Fan, L., Willamowski, J., Bray, C.: Visual categorization with bags of keypoints. In: Workshop on Statistical Learning in Computer Vision, ECCV (2004)
6. Elidan, G.: Copula bayesian networks. In: Proceedings of the NIPS, Advances in Neural Information Processing Systems 23, pp. 559–567 (2010)
7. Frank, E.T., Xu, X.: Applying propositional learning algorithms to multi-instance data. Technical report, University of Waikato, Department of Computer Science, University of Waikato, Hamilton, NZ, June 2003
8. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, pp. 148–156. Morgan Kaufmann, San Francisco (1996)
9. Fürnkranz, J., Hüllermeier, E. (eds.): Preference Learning. Springer, Heidelberg (2011)
10. Grabisch, M., Marichal, J., Mesiar, R., Pap, E.: Aggregation Functions. Cambridge University Press, Cambridge (2009)

11. Grabisch, M., Marichal, J., Mesiar, R., Pap, E.: Aggregation functions: construction methods, conjunctive, disjunctive and mixed classes. Inf. Sci. **181**, 23–43 (2011)
12. Greco, S., Mousseau, V., Slowinski, R.: Robust ordinal regression for value functions handling interacting criteria. Eur. J. Oper. Res. **239**(3), 711–730 (2014)
13. Hajek, P.: Metamathematics of Fuzzy Logic. Springer, Dordrecht (1998)
14. Hajimirsadeghi, H., Mori, G.: Multiple instance real boosting with aggregation functions. In: Proceedings of the ICPR, 21st International Conference on Pattern Recognition, pp. 2706–2710 (2012)
15. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: an update. SIGKDD Explor. **11**(1), 10–18 (2009)
16. Klement, E., Mesiar, R., Pap, E.: Triangular Norms. Kluwer Academic Publishers, Dordrecht (2002)
17. McKay, M.D., Beckman, R.J., Conover, W.J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics **21**(2), 239 (1979)
18. Musicant, D.R., Christensen, J.M., Olson, J.F.: Supervised learning by training on aggregate outputs. In: Proceedings of the ICDM, 7th IEEE International Conference on Data Mining, Omaha, Nebraska, USA, pp. 252–261 (2007)
19. Platt, J.: Machines using sequential minimal optimization. In: Schoelkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods - Support Vector Learning. MIT Press (1998)
20. Ray, S., Page, D.: Multiple instance regression. In: ICML, vol. 1, pp. 425–432 (2001)
21. Schweizer, B., Sklar, A.: Probabilistic Metric Spaces. North-Holland, New York (1983)
22. Senge, R., Hüllermeier, E.: Top-down induction of fuzzy pattern trees. IEEE Trans. Fuzzy Syst. **19**(2), 241–252 (2011)
23. Senge, R., Hüllermeier, E.: Fast fuzzy pattern tree learning for classification. IEEE Trans. Fuzzy Syst. **23**(6), 2024–2033 (2015)
24. Tehrani, A.F., Cheng, W., Dembczynski, K., Hüllermeier, E.: Learning monotone nonlinear models using the Choquet integral. Mach. Learn. **89**(1), 183–211 (2012)
25. Tschiatschek, S., Djolonga, J., Krause, A.: Learning probabilistic submodular diversity models via noise contrastive estimation. In: Proceedings of the AISTATS, 19th International Conference on Artificial Intelligence and Statistics (2016)
26. Narukawa, Y., Torra, T.: Modeling Decisions: Information Fusion and Aggregation Operators. Springer, Berlin (2007)
27. Yager, R., Rybalov, A.: Uninorm aggregation operators. Fuzzy Sets Syst. **80**, 111–120 (1996)