



# Efficient set-valued prediction in multi-class classification

Thomas Mortier<sup>1</sup> · Marek Wydmuch<sup>2</sup> · Krzysztof Dembczyński<sup>2,3</sup> ·  
Eyke Hüllermeier<sup>4</sup> · Willem Waegeman<sup>1</sup>

Received: 15 May 2020 / Accepted: 12 March 2021

© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2021

## Abstract

In cases of uncertainty, a multi-class classifier preferably returns a set of candidate classes instead of predicting a single class label with little guarantee. More precisely, the classifier should strive for an optimal balance between the correctness (the true class is among the candidates) and the precision (the candidates are not too many) of its prediction. We formalize this problem within a general decision-theoretic framework that unifies most of the existing work in this area. In this framework, uncertainty is quantified in terms of conditional class probabilities, and the quality of a predicted set is measured in terms of a utility function. We then address the problem of finding the Bayes-optimal prediction, i.e., the subset of class labels with the highest expected utility. For this problem, which is computationally challenging as there are exponentially (in the number of classes) many predictions to choose from, we propose efficient algorithms that can be applied to a broad family of utility functions. Our theoretical

Responsible editors: Ira Assent, Carlotta Domeniconi, Aristides Gionis.

✉ Thomas Mortier  
thomasf.mortier@ugent.be

Marek Wydmuch  
mwydmuch@cs.put.poznan.pl

Krzysztof Dembczyński  
kdembczynski@cs.put.poznan.pl

Eyke Hüllermeier  
eyke@upb.de

Willem Waegeman  
willem.waegeman@ugent.be

<sup>1</sup> Department of Data Analysis and Mathematical Modelling, Ghent University, Coupure links 653, 9000 Ghent, Belgium

<sup>2</sup> Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland

<sup>3</sup> Yahoo! Research, 770 Broadway, New York, NY 10003, USA

<sup>4</sup> Institute of Informatics, LMU Munich, Akademiestr. 7, 80799 Munich, Germany

results are complemented by experimental studies, in which we analyze the proposed algorithms in terms of predictive accuracy and runtime efficiency.

**Keywords** Set-valued prediction · Multi-class classification · Expected utility maximization

## 1 Introduction

In probabilistic multi-class classification, one often encounters situations in which the classifier is uncertain about the class label for a given instance. In such cases, instead of predicting a single class, it might be beneficial to return a set of classes as a prediction, with the idea that the correct class should at least be contained in that set. For example, in medical diagnosis, when not being sure enough about the true disease of a patient, it is better to return a set of candidate diseases. Provided this set is sufficiently small compared to the total number of diagnoses, it can still be of great help for a medical doctor, because only the remaining candidate diseases need further investigation.

Let us introduce set-valued prediction in a formal way. We assume training examples  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$  from a distribution  $P(\mathbf{x}, y)$  on  $\mathcal{X} \times \mathcal{Y}$ , with  $\mathcal{X}$  some instance space (e.g., images, documents, etc.) and  $\mathcal{Y} = \{c_1, \dots, c_K\}$  a class space consisting of  $K$  classes. In a probabilistic multi-class classification setting, we estimate the conditional class probabilities  $P(\cdot | \mathbf{x})$  over  $\mathcal{Y}$ , with properties  $\forall c \in \mathcal{Y} : 0 \leq P(c | \mathbf{x}) \leq 1, \sum_{c \in \mathcal{Y}} P(c | \mathbf{x}) = 1$ . This distribution can be estimated using a wide range of well-known probabilistic methods (see further below). We will consider a set-valued prediction  $\hat{Y}$  from the power set of  $\mathcal{Y}$ , i.e., predictions are (non-empty) subsets of  $\mathcal{Y}$ , or more formally,  $\hat{Y} \in 2^{\mathcal{Y}} \setminus \{\emptyset\}$ .

In the literature, different methods for set-valued prediction have been proposed (cf. Sect. 4), essentially following two main directions. The first idea is to construct a set that covers the true outcome with a predefined (high) probability. A set-valued prediction of that kind can be seen as a generalization of the notion of confidence interval in frequentist statistics or credible interval in Bayesian statistics. A well-known representative of this statistical approach is conformal prediction (Shafer and Vovk 2008). The second direction is rooted in (Bayesian) decision theory and involves the notion of a *utility function*, which evaluates a set-valued prediction in terms of its usefulness (Del Coz et al. 2009; Corani and Zaffalon 2008, 2009; Zaffalon et al. 2012; Yang et al. 2017b). Typically, the utility specifies a compromise between two natural though conflicting criteria: like in the statistical approach, the prediction should be *correct* in the sense of covering the true class, but at the same time, it should be *precise* and not contain too many options. Given a utility function of that kind, combined with a probability estimate on the classes, the natural decision-theoretic approach consists of predicting the set with the highest expected utility. In this paper, we will focus on this approach, which we refer to as the *set-based utility maximization framework*.

## 1.1 Set-based utility maximization

In set-based utility maximization, the quality of the prediction  $\hat{Y}$  can be expressed by means of a set-based utility function  $u(c, \hat{Y})$ , where  $c$  corresponds to the ground-truth class and  $\hat{Y}$  is the predicted set. Typically, a decision-theoretic framework is considered, where one estimates a probabilistic model, followed by an inference procedure at prediction time. At prediction time, the goal is to find the Bayes-optimal solution  $\hat{Y}_u^*$  by expected utility maximization:

$$\begin{aligned}\hat{Y}_u^*(\mathbf{x}) &= \arg \max_{\hat{Y} \in 2^{\mathcal{Y}} \setminus \{\emptyset\}} \mathbb{E}_{P(c|\mathbf{x})}[u(c, \hat{Y})] = \arg \max_{\hat{Y} \in 2^{\mathcal{Y}} \setminus \{\emptyset\}} \sum_{c \in \mathcal{Y}} u(c, \hat{Y}) P(c|\mathbf{x}), \\ &= \arg \max_{\hat{Y} \in 2^{\mathcal{Y}} \setminus \{\emptyset\}} U(\hat{Y}, P, u),\end{aligned}\quad (1)$$

where we introduce the shorthand notation  $U(\hat{Y}, P, u)$  for the expected utility.

Several authors have solved this optimization problem for different utility functions that are members of a general family  $u : \mathcal{Y} \times 2^{\mathcal{Y}} \setminus \{\emptyset\} \rightarrow [0, 1]$ , defined as follows:

$$u(c, \hat{Y}) = \begin{cases} 0 & \text{if } c \notin \hat{Y}, \\ g(|\hat{Y}|) & \text{if } c \in \hat{Y}, \end{cases} \quad (2)$$

where  $|\hat{Y}|$  denotes the cardinality of the predicted set  $\hat{Y}$ . This family is characterized by a decreasing sequence  $(g(1), \dots, g(K)) \in [0, 1]^K$  that can have different forms. Del Coz et al. (2009), who use the term *nondeterministic classification* as a synonym for set-based utility maximization, concentrate on three scores from the information retrieval community: precision with  $g_P(s) = 1/s$ , recall with  $g_R(s) = 1$ , and the  $F_\beta$ -measure with  $g_{F_\beta}(s) = (1 + \beta^2)/(\beta^2 + s)$ . Other utility functions with specific choices for  $g$  are also studied in the literature (Corani and Zaffalon 2008, 2009; Zaffalon et al. 2012; Yang et al. 2017b; Nguyen et al. 2018). Those utility functions include:

$$g_{\delta, \gamma}(s) = \frac{\delta}{s} - \frac{\gamma}{s^2}, \quad g_\delta(s) = 1 - \exp\left(-\frac{\delta}{s}\right), \quad g_{\log}(s) = \log\left(1 + \frac{1}{s}\right).$$

Especially  $g_{\delta, \gamma}(s)$  is commonly used in the above papers, where  $\delta$  and  $\gamma$  can only take certain values to guarantee that the utility is in the interval  $[0, 1]$ . Precision (also called discounted accuracy) corresponds to the case  $(\delta, \gamma) = (1, 0)$ . However, typical choices for  $(\delta, \gamma)$  are  $(1.6, 0.6)$  and  $(2.2, 1.2)$  (Nguyen et al. 2018), which overcome some of the limitations of precision (see below for a discussion). The utility function  $g_\delta$  is an exponentiated version of precision, where the parameter  $\delta$  controls the reward when sets become larger.

## 1.2 Contributions and outline

In this paper, we will focus on aspects related to optimizing (1). This is a non-trivial optimization problem, as a brute-force search requires checking all subsets of  $\mathcal{Y}$ ,

resulting in an exponential time complexity. However, we will be able to find the Bayes-optimal prediction more efficiently. As our main contribution, we present several algorithms that solve (1) in an efficient manner. In the literature, the work of Del Coz et al. (2009) is the closest to our work. We extend their work in two directions: 1) the algorithms that we introduce are more efficient in multi-class classification settings where the number of classes is large, such as language modelling and reinforcement learning, and 2) our algorithms are applicable to a wide range of utility functions, unlike the algorithm of Del Coz et al. (2009), which concentrates on the  $F_\beta$ -measure.

In Sect. 2 we present several theoretical results. Those results are essential building blocks for solving (1), but we also discuss the impact of these results for different utility functions. The algorithms that we develop are further materialized in Sect. 3. We first discuss an exact Bayes-optimal algorithm that makes  $K$  queries to the conditional distribution  $P(c | \mathbf{x})$ , with  $K$  the number of classes. In addition, we also introduce two approximate algorithms that make less than  $K$  calls to  $P(c | \mathbf{x})$ . Those algorithms are based on two different paradigms: maximum inner product search and hierarchical factorization of the conditional distribution. To conclude the theoretical part of this work, we provide an overview of related work in Sect. 4. In Sect. 5 three different types of experimental results are discussed. In a first experiment, we use image and text classification datasets to highlight the usefulness of set-valued prediction in case of uncertainty. In a second type of experiments, we evaluate the exact algorithm against some simple baselines for set-valued prediction. In a last experiment, we focus on the runtime of the exact algorithm, highlighting the additional speedups that can be obtained by considering approximate algorithms for set-valued prediction.

## 2 Theoretical results

In this section, we present several theoretical results as building blocks of the algorithms that we consider later on. We start with some general results, followed by a discussion of considerations for specific utility functions.

### 2.1 General results

The formulation in (1) seems to suggest that all subsets of  $\mathcal{Y}$  need to be analyzed to find the Bayes-optimal solution, but a first result shows that this is not the case.

**Theorem 1** *The exact solution of (1) can be computed by analyzing only  $K$  subsets of  $\mathcal{Y}$ .*

**Proof** With  $P(\hat{Y} | \mathbf{x}) = \sum_{c \in \hat{Y}} P(c | \mathbf{x})$ , the expected utility can be written as

$$\begin{aligned} U(\hat{Y}, P, u) &= \sum_{c \in \mathcal{Y}} u(c, \hat{Y}) P(c | \mathbf{x}) = \sum_{c \in \hat{Y}} u(c, \hat{Y}) P(c | \mathbf{x}) + \sum_{c' \notin \hat{Y}} u(c', \hat{Y}) P(c' | \mathbf{x}), \\ &= \sum_{c \in \hat{Y}} g(|\hat{Y}|) P(c | \mathbf{x}) = g(|\hat{Y}|) P(\hat{Y} | \mathbf{x}), \end{aligned} \quad (3)$$

where the last summation in the second equality cancels out since  $u(c', \hat{Y}) = 0$ . Let us decompose (1) into an inner and an outer maximization step. The inner maximization step then becomes

$$\hat{Y}_u^{*s} = \arg \max_{|\hat{Y}|=s} g(s) P(\hat{Y} | \mathbf{x}) = \arg \max_{|\hat{Y}|=s} P(\hat{Y} | \mathbf{x}), \quad (4)$$

for  $s = \{1, \dots, K\}$ , where the last equality trivially holds due to  $g(s)$  being constant. This step can be done very efficiently, by sorting the conditional class probabilities, as for a given  $s$ , only the subset with the highest probability mass needs to be considered. The outer maximization simply consists of computing

$$\hat{Y}_u^*(\mathbf{x}) = \arg \max_{\hat{Y} \in \{\hat{Y}_u^{*1}, \dots, \hat{Y}_u^{*K}\}} g(|\hat{Y}|) P(\hat{Y} | \mathbf{x}), \quad (5)$$

which only requires the evaluation of  $K$  sets.  $\square$

So, one only needs to evaluate  $\hat{Y}_u^{*1}, \dots, \hat{Y}_u^{*K}$  to find the Bayes-optimal solution, which limits the search to  $K$  subsets. In fact, we can even do better as it turns out that by restricting  $g$ , we can assure that the sequence  $U(\hat{Y}_u^{*1}, P, u), \dots, U(\hat{Y}_u^{*K}, P, u)$  will have reached its global maximum when it starts to decrease. This will allow us to further limit the search, by means of an early stopping criterion, as soon as we reach that maximum. The restriction required for  $g$  is  $(1/x)$ -convexity, i.e., convexity after a  $(1/x)$  transformation. This is a somewhat surprising and rather technical result that is summarized in the following definition and theorem.

**Definition 1** A sequence  $g(1), g(2), \dots, g(K)$  is  $(1/x)$ -convex if

$$1/g(s+1) \leq \frac{1/g(s) + 1/g(s+2)}{2} \quad \text{for all } s \in \{1, \dots, K-2\}. \quad (6)$$

**Theorem 2** Let  $g(1), g(2), \dots, g(K)$  be a decreasing  $(1/x)$ -convex sequence. Then the following implication holds for any  $s \in \{1, \dots, K-2\}$ :

$$U(\hat{Y}_u^{*s}, P, u) > U(\hat{Y}_u^{*s+1}, P, u) \implies U(\hat{Y}_u^{*s+1}, P, u) > U(\hat{Y}_u^{*s+2}, P, u).$$

**Proof** Let us first observe the following equivalence:

$$\begin{aligned} 1/g(s+1) &\leq \frac{1/g(s) + 1/g(s+2)}{2} \\ \Leftrightarrow 2 &\leq \frac{g(s+1)}{g(s)} + \frac{g(s+1)}{g(s+2)} \\ \Leftrightarrow g(s)g(s+1) + g(s+2)g(s+1) &\geq 2g(s)g(s+2) \\ \Leftrightarrow g(s)[g(s+1) - g(s+2)] &\geq g(s+2)[g(s) - g(s+1)] \\ \Leftrightarrow \frac{g(s)}{g(s) - g(s+1)} &\geq \frac{g(s+2)}{g(s+1) - g(s+2)} \\ \Leftrightarrow \frac{g(s+1)}{g(s) - g(s+1)} + 1 &\geq \frac{g(s+2)}{g(s+1) - g(s+2)}. \end{aligned} \quad (7)$$

Assume that for a given  $s$  it holds that  $U(\hat{Y}_u^{*s}, P, u) > U(\hat{Y}_u^{*s+1}, P, u)$ . Let  $p_i = P(c_i | \mathbf{x})$  and observe that the following equivalences holds:

$$\begin{aligned}
 U(\hat{Y}_u^{*s}, P, u) > U(\hat{Y}_u^{*s+1}, P, u) &\Leftrightarrow g(s) \sum_{i=1}^s p_i > g(s+1) \sum_{i=1}^{s+1} p_i \\
 &\Leftrightarrow [g(s) - g(s+1)] \sum_{i=1}^s p_i > g(s+1) p_{s+1} \\
 &\Leftrightarrow \sum_{i=1}^s p_i > \frac{g(s+1)}{g(s) - g(s+1)} p_{s+1}. \quad (8)
 \end{aligned}$$

Observe that from (7) and (8) it follows that:

$$\begin{aligned}
 \sum_{i=1}^{s+1} p_i > \frac{g(s+1)}{g(s) - g(s+1)} p_{s+1} + p_{s+1} &\Leftrightarrow \sum_{i=1}^{s+1} p_i > \left( \frac{g(s+1)}{g(s) - g(s+1)} + 1 \right) p_{s+1} \\
 &\Rightarrow \sum_{i=1}^{s+1} p_i > \frac{g(s+2)}{g(s+1) - g(s+2)} p_{s+2} \\
 &\Leftrightarrow U(\hat{Y}_u^{*s+1}, P, u) > U(\hat{Y}_u^{*s+2}, P, u).
 \end{aligned}$$

This is what we needed to prove.  $\square$

Thus, what Theorem 2 tells us is that, for  $(1/x)$ -convex sequences, we have found a stopping criterion so that even less than  $K$  sets need to be analyzed when optimizing (1). More specifically, we can stop as soon as the sequence

$$U(\hat{Y}_u^{*1}, P, u), U(\hat{Y}_u^{*2}, P, u) \dots, U(\hat{Y}_u^{*K}, P, u)$$

starts to decrease. The stopping criterion is  $U(\hat{Y}_u^{*s}, P, u) > U(\hat{Y}_u^{*s+1}, P, u)$  for a given  $s \in \{1, \dots, K-1\}$ .

Theorems 1 and 2 provide guarantees to optimize (1) in a classical decision-theoretic context. In the appendix, we present a short theoretical analysis that relates the Bayes-optimal solution for the set-based utility functions to the solution obtained on the conditional class probabilities given by a trained model. The goal is to upper bound the regret of the set-based utility functions by the  $L_1$  error of the class probability estimates. Similarly, as in decision-theoretic utility maximization, the analysis is performed on the level of a single  $\mathbf{x}$ .

## 2.2 Considerations w.r.t. specific utility functions

Let us discuss the implications of the above theorems for the utility functions that were mentioned in the introduction (see also Table 1 and the left panel of Fig. 1 for an overview). Is  $(1/x)$ -convexity satisfied for those utility functions? For the ones

that are most commonly used in the literature the answer turns out to be positive: precision, recall, the  $F_\beta$ -measure, as well as the  $g_{\delta,\gamma}$  family for recommended values of  $\delta$  and  $\gamma$ , are all utilities with associated  $(1/x)$ -convex sequences. Let us remark that  $(1/x)$ -convexity cannot easily be assessed by plotting the graph of a specific sequence  $g(s)$ . In practice one needs to check this formally using (6) for all  $s$ .

Precision, with  $g_P(s) = 1/s$ , in fact defines “how convex” a sequence is allowed to be, because (6) is satisfied as an equality in that boundary case. As shown in the left panel of Fig. 1, most utility functions from the literature behave very similar to precision; they decrease very quickly, and their curvature is similar to precision, but they are somewhat less convex (remark that for functions the degree of convexity is determined by the slope of the first derivative). From that perspective, it is obvious that concave sequences will be  $(1/x)$ -convex. The following proposition states this formally.

**Definition 2** A sequence  $g(1), g(2), \dots, g(K)$  is concave if

$$g(s+1) \geq \frac{g(s) + g(s+2)}{2} \quad \text{for all } s \in \{1, \dots, K-2\}$$

**Proposition 1** Let  $g(1), g(2), \dots, g(K)$  be a decreasing, concave sequence. Then  $g(1), g(2), \dots, g(K)$  is also  $(1/x)$ -convex.

**Proof** As shown above, the following equivalence holds:

$$\begin{aligned} 1/g(s+1) &\leq \frac{1/g(s) + 1/g(s+2)}{2} \\ \Leftrightarrow g(s)[g(s+1) - g(s+2)] &\geq g(s+2)[g(s) - g(s+1)] \end{aligned} \quad (9)$$

Due to the fact that  $g(1), g(2), \dots, g(K)$  is a decreasing, concave sequence we know that  $g(s) \geq g(s+2)$  and  $g(s+1) - g(s+2) \geq g(s) - g(s+1)$ . Combining the two inequalities lets us conclude that the sequence must be  $(1/x)$ -convex.  $\square$

Apart from  $(1/x)$ -convexity, there is another property that an interesting utility function should obey:  $g(s)$  should be lower bounded by precision, i.e.  $g_P$ . Precision and recall are frequently used in binary classification, but one may argue that they are both not very useful utility functions for assessing set-valued predictions. For recall this is rather obvious, as this measure does not have any penalty for the size of the set. Moreover, for precision the problem is that its utility maximiser will always be a set of cardinality one. For example, consider a multi-class problem with hundred classes, and assume that for a given instance the conditional class probabilities are 0.1 for ten of these hundred classes. Clearly, in this case, the best prediction is to return a set consisting of the ten classes, resulting in an expected utility of 0.1. If  $g(10) = 1/10$ , a singleton set that contains only one of the ten classes also yields an expected utility of 0.1. Both are Bayes-optimal predictions in view of (1). Thus, the problem with precision is that it is not risk-averse. In the face of uncertainty, a risk-averse utility function will prefer a set with many classes over a singleton set that contains one of those classes (Zaffalon et al. 2012). That is why we highlight

$g(s) \geq 1/s$  as an absolute requirement. Utility functions violating this requirement are practically pointless, because the solution to (1) is always a set of cardinality one in that case. The next proposition formalizes this claim, which has been reported by Zaffalon et al. (2012) from a different perspective and using a different notation.

**Proposition 2** *Let  $g$  be a sequence such that  $g(1) = 1$ , then for any distribution  $P$  the following statements hold.*

- When  $g(s) < 1/s$  for some  $s > 1$ , then  $\hat{Y}_u^{*s}$  is not a solution of (1),
- When  $g(s) < 1/s$  for all  $s > 1$ , then the unique solution of (1) is  $\hat{Y}_u^{*1}$ ,
- When  $g(s) = 1/s$  for all  $s$ , then  $\hat{Y}_u^{*1}$  is a solution of (1).

**Proof** Let us start by proving the first statement. When  $g(s) < 1/s$  it follows that:

$$U(\hat{Y}_u^{*s}, P, u) < \frac{P(\hat{Y}_u^{*s} | \mathbf{x})}{s} \leq P(\hat{Y}_u^{*1} | \mathbf{x}) = U(\hat{Y}_u^{*1}, P, u),$$

where the second inequality holds because  $\hat{Y}_u^{*1}$  must correspond to the mode of  $P$ , and the last equality holds because we assume  $g(1) = 1$ . So, we have shown that, when  $g(s) < 1/s$  for some  $s$ , it holds that  $U(\hat{Y}_u^{*s}, P, u) < U(\hat{Y}_u^{*1}, P, u)$  for any distribution  $P$ . This reasoning can be applied to any  $s > 1$ , which also proves the second statement. Using  $g(s) = 1/s$  in the above reasoning, the inequalities become equalities. This proves the third statement.  $\square$

The proposition lets us conclude that it is pointless to use any utility for which  $g(s) < 1/s$  for all  $s$ , because  $\hat{Y}_u^{*1}$  is then the unique solution of (1). It is also pointless to use  $g_P$  as utility function:  $\hat{Y}_u^{*1}$  will then be one of the solutions of (1), but there might be other solutions as well. When  $g(s) < 1/s$  for some but not all  $s$ , only sets of specific cardinalities can be the solution of (1). This is probably also unwanted in practice.

So, what about the other utility functions from the literature? In Fig. 1, one can see that utility functions  $g_{F1}$  and  $g_{\delta=1.6, \gamma=0.6}$  behave similarly as  $g_P$ , but they are lower bounded by  $g_P$ , so with those utility functions it will be possible to predict non-singleton sets. In general, the faster those functions decrease, the smaller the predicted sets will be. For  $g_{F\beta}$  the parameter  $\beta$  controls the degree of convexity, as shown in the right panel of Fig. 1. As such, this parameter will in a way also control the size of the sets that are predicted. Note that Proposition 2 cannot be applied for  $g_{\log}$  and  $g_{\delta}$ , because  $g(1) \neq 1$  in those cases. One would need to rescale those utility functions first, before applying the proposition.

As a summary of the above discussion, we define four properties that a meaningful utility function  $g$  should have:

- $g$  should be strictly decreasing. Smaller sets should have a higher utility than bigger sets.
- $g(1) = 1$ . Normalization facilitates the comparison between different utility functions. A function violating this property should be rescaled, if possible.



- $g(s) > 1/s$  for all  $s$ . The spectrum of utility maximizers will be limited to sets of particular cardinalities when  $g(s) < 1/s$  for some  $s$ . If  $g(s) < 1/s$  for all  $s$ , the utility function becomes completely useless.
- $g$  should be  $(1/x)$ -convex. This guarantees that the utility maximizer can be found efficiently.

There is a close link between the third and fourth property: if  $g(s) < 1/s$  for some  $s$ , then  $g$  cannot be  $(1/x)$ -convex. However, the converse is not true. To give such an example, let us introduce a novel family of utility functions that generalizes a utility function that is often used in the literature on multi-class classification with reject option, see e.g. (Ramswamy et al. 2015). When a reject option is allowed, the prediction can only be a singleton or the full set  $\mathcal{Y}$  containing  $K$  classes. The first case typically gets a reward of one, while the second case should receive a lower reward, e.g.  $1 - \alpha$ . This second case corresponds to abstaining, i.e., not predicting any class label, and the (user-defined) parameter  $\alpha$  specifies a penalty for doing so, with the requirement  $0 < \alpha < 1 - 1/K$  to be risk-averse. To include sets of any cardinality  $s$ , the utility could be generalized as follows:

$$g_{\alpha,\beta}(s) = 1 - \alpha \left( \frac{s-1}{K-1} \right)^\beta, \quad (10)$$

which we call the generalized reject option utility. Here, we have the same interpretation for  $\alpha$ , whereas  $\beta \in (0, \infty)$  defines whether  $g(s)$  is convex or concave, as shown in the bottom panel of Fig. 1. While convexity (like in most of the above utility functions) appears natural in most applications, a concave utility might be useful when predicting a large set is tolerable. In the limit, when  $\beta \rightarrow 0$ , we obtain the simple utility function for classification with reject option.

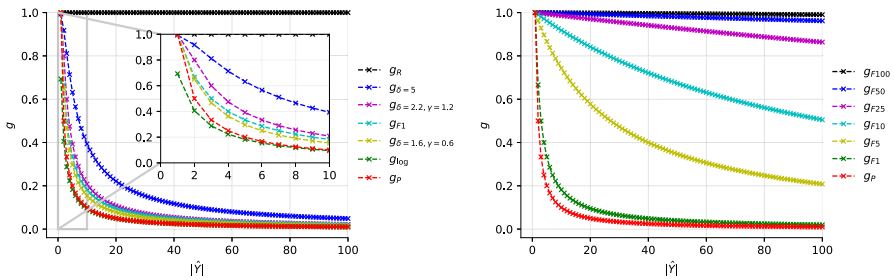
The  $g_{\alpha,\beta}$  family is quite intuitive from an application perspective, and it has a lot of flexibility. This makes this family interesting, but for certain parameterizations it is not lower bounded by precision. It is important to choose  $\alpha \leq \frac{K-1}{K}$  and  $\beta \geq \log_{\frac{1}{K-1}} \frac{K}{2} + 1$  to guarantee that  $g_{\alpha,\beta}(s) \geq 1/s$  for all  $s$  (see appendix for derivations). For example, as shown in the figure in the appendix, for  $\alpha = 1$ ,  $g_{\alpha,\beta}(s)$  is dominated by  $g_P(s)$  for large  $s$ . We also observed that  $g_{\alpha,\beta}(s)$  is for some  $\alpha$  and  $\beta$  not  $(1/x)$ -convex. In contrast, the  $g_{F\beta}$  family and  $g_{\delta,\gamma}$  for recommended values of  $\delta$  and  $\gamma$  deliver utility functions that satisfy all the four properties that are listed above. This makes them interesting from an application perspective, and they will be our focus in the experiments.

### 3 Algorithmic solutions

The above theoretical results (in particular Theorems 1 and 2) suggest that problem (1) can be efficiently solved for  $(1/x)$ -convex set-based utility functions. In this section, we present three algorithmic solutions for this problem, which are all based on the same generic framework. The first algorithm returns the Bayes-optimal solution to (1) in an exact manner. The other two algorithms compute approximate solutions, but

**Table 1** The utility functions  $g(s)$  discussed in this paper

Name	$g(s)$	Article
Precision	$g_P(s) = \frac{1}{s}$	(Del Coz et al. 2009)
Recall	$g_R(s) = 1$	(Del Coz et al. 2009)
$F_\beta$ -measure	$g_{F\beta}(s) = \frac{1+\beta^2}{s+\beta^2}$	(Del Coz et al. 2009)
Credal utility	$g_{\delta,\gamma}(s) = \frac{\delta}{s} - \frac{\gamma}{s^2}$	(Zaffalon et al. 2012)
Exp. utility	$g_\delta(s) = 1 - \exp\left(-\frac{\delta}{s}\right)$	(Zaffalon et al. 2012)
Log. utility	$g_{\log}(s) = \log\left(1 + \frac{1}{s}\right)$	(Zaffalon et al. 2012)
Reject option	$g_{\text{rej}}(s) = \begin{cases} 1 & \text{if } s = 1, \\ 1 - \alpha & \text{if } s = K. \end{cases}$	(Ramaswamy et al. 2015)
Gen. reject option	$g_{\alpha,\beta}(s) = 1 - \alpha \left(\frac{s-1}{K-1}\right)^\beta$	Extension of (Ramaswamy et al. 2015)



**Fig. 1** A visualization of  $g$  in function of different values of  $|\hat{Y}|$  and set-based utility functions.  $K = 100$

---

**Algorithm 1** SVBOP – **input:**  $u(\cdot)$ ,  $\mathbf{x}$ ,  $\mathcal{Y} = \{c_1, \dots, c_K\}$ ,  $PC$

---

1: $\hat{Y} \leftarrow \emptyset, p_{\hat{Y}} \leftarrow 0, U^* \leftarrow 0$ 2: $PC.initPrediction(\mathbf{x}, \mathcal{Y})$ 3: <b>while</b> $(c, p_c) \leftarrow PC.getNextClass()$ <b>do</b> 4: $\hat{Y} \leftarrow \hat{Y} \cup \{c\}, p_{\hat{Y}} \leftarrow p_{\hat{Y}} + p_c$ 5: $U_{\hat{Y}} \leftarrow p_{\hat{Y}} \times g(\hat{Y})$ 6: <b>if</b> $U^* \leq U_{\hat{Y}}$ <b>then</b> 7: $\hat{Y}_u^* \leftarrow \hat{Y}, U^* \leftarrow U_{\hat{Y}}$ 8: <b>else break</b> 9: <b>return</b> $\hat{Y}_u^*$	▷ Initialize the current best solution, its probability and utility ▷ Initialize the prediction procedure ▷ Repeat until all the classes are returned by $PC$ ▷ Update the current solution and its probability ▷ Compute $U(\hat{Y}, P, u)$ according to Eq. (3) ▷ If the current solution is better than the best solution so far ▷ Replace the current best solution ▷ If there is no improvement break the while loop according to Theorem 2 ▷ Return the set of classes with the highest utility
--	---

---

yield substantial runtime gains. Those algorithms can be used when the number of classes is large.

Algorithm 1 presents the generic framework. We use the acronym SVBOP, which stands for Set-Valued Bayes-Optimal Prediction. SVBOP uses a probabilistic classifier, denoted as  $PC$ , that supports two operations. The first operation, *initPrediction*, initializes the prediction procedure for a given test example  $\mathbf{x}$ . The second operation, *getNextClass*, placed in the while loop, returns the next class label with respect to decreasing conditional class probabilities. In each subsequent iteration of the while loop, the solution of inner maximization (4) is for a given  $s$  found by adding the class with the  $s$ -highest conditional class probability to the predicted set. In this way,  $U(\hat{Y}_u^{*1}, P, u), \dots, U(\hat{Y}_u^{*s}, P, u)$  are computed in a sequential way, till the stopping criterion of Theorem 2 is satisfied.

This framework can be seen as a generalization of an algorithm introduced by Del Coz et al. (2009) for optimizing the  $F_\beta$ -measure in multi-class classification. There is also a strong correspondence with certain  $F_\beta$ -maximization algorithms in multi-label classification (see e.g. Jansche 2007; Ye et al. 2012; Waegeman et al. 2014).

### 3.1 SVBOP-full

The first algorithm is further referred to as SVBOP-Full, because it computes all conditional class probabilities.  $PC$  is here a standard multi-class probabilistic classifier that returns the estimated conditional class distribution for a given test example  $\mathbf{x}$ . Examples of such classifiers are logistic regression, linear discriminant analysis, gra-

**Algorithm 2** SVBOP-Full.initPrediction – **input:**  $\mathbf{x}, \mathcal{Y} = \{c_1, \dots, c_K\}$ 


---

```

1:  $\mathcal{Q} \leftarrow \emptyset$  ▷ Initialize a list to store classes and  $P(c | \mathbf{x})$ 
2: for  $c \in \mathcal{Y}$  do  $p_c \leftarrow P(c | \mathbf{x})$ ,  $\mathcal{Q}.add((c, p_c))$  ▷ Query  $PC$  to get  $P(c | \mathbf{x})$  for all classes
3:  $\mathcal{Q}.sort()$  ▷ Sort the list decreasingly according to  $P(c | \mathbf{x})$ 

```

---

**Algorithm 3** SVBOP-Full.getNextClass

---

```

1: return  $\mathcal{Q}.pop()$  ▷ Pop the next highest element from the sorted list.

```

---

dient boosting trees or neural networks with a softmax output layer. The inference algorithm starts by querying the underlying classifier to get all  $K$  conditional class probabilities  $P(c | \mathbf{x})$ . Then, the conditional class probabilities are sorted in decreasing order (Algorithm 2). When in Algorithm 1 the next class label is needed, it can simply be taken from this sorted list (Algorithm 3).

This approach is simple and elegant but requires sorting of all  $K$  conditional class probabilities, which results in an  $O(K \log K)$  complexity. However, the most costly operation is usually querying the distribution  $P$  to obtain values of conditional probabilities for all  $K$  classes. In case of linear models, this cost scales linearly with the number of classes, multiplied by the number of non-zero feature values. For problems with a large number of classes, often referred to as extreme classification problems (Prabhu and Varma 2014), this is usually too costly.

### 3.2 Hierarchical search with similarity graphs (SVBOP-HSG)

Since only class labels with high probability mass are required, a procedure would be desirable that returns the top classes without the need to compute conditional class probabilities for all classes. To accomplish this, we leverage approaches for approximate nearest neighbor search (Yagnik et al. 2011; Shrivastava and Li 2014; Johnson et al. 2017) and adapt them to our setting. The use of such methods essentially becomes possible through two problem transformations: first, we reduce maximum probability search to maximum inner product search, which we then in turn reduce to nearest neighbor search. In the following, we briefly comment on both reduction steps and eventually on the approximate nearest neighbor search itself.

As for the first step, note that most learning algorithms produce class probability estimates in the last step of the learning procedure by mapping scores to probabilities. A typical example, which is also used in our approach, is the softmax transformation:

$$P(c | \mathbf{x}) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x})}{\sum_{c' \in \mathcal{Y}} \exp(\mathbf{w}_{c'} \cdot \mathbf{x})}. \quad (11)$$

Obviously, as long as the probability is a monotone function of an inner product  $\mathbf{w}_c \cdot \mathbf{x}$  between the query instance  $\mathbf{x}$  and a weight vector  $\mathbf{w}_c$ , like in (11), finding  $\arg \max_c P(c | \mathbf{x})$  is equivalent to finding  $\arg \max_c \mathbf{w}_c \cdot \mathbf{x}$ . More generally, finding the top- $s$  inner products corresponds to finding the top- $s$  probabilities. Furthermore, there is no need to compute the value of the partition function, i.e., the denominator of

(11), to find the optimal set-valued prediction. For a given  $\mathbf{x}$ , the value of the partition function is constant for all  $c$  and since  $\arg \max f(\mathbf{x}) = \arg \max a \times f(\mathbf{x})$ , for any constant  $a > 0$ , the lack of normalization does not affect the result of both inner (4) and outer (5) maximization.

As for the second step, let us assume that, as discussed before, we have a linear model for each class  $c$ , which is represented by a vector  $\mathbf{w}_c$  in a suitable (perhaps transformed) feature space  $\mathcal{X}$  (e.g., the last but one layer in a neural network, which is mapped to the output via softmax). Now, since the squared Euclidean distance between  $\mathbf{w}_c$  and  $\mathbf{x}$  is given by  $d(\mathbf{w}_c, \mathbf{x}) = \|\mathbf{w}_c\|_2^2 - 2\mathbf{w}_c \cdot \mathbf{x} + \|\mathbf{x}\|_2^2$ , maximizing  $\mathbf{w}_c \cdot \mathbf{x}$  is “almost” equivalent to minimizing  $d(\mathbf{w}_c, \mathbf{x})$ . More specifically, it is equivalent to minimizing the distance  $d(\mathbf{w}'_c, \mathbf{x}')$  between two expanded vectors  $\mathbf{w}'_c$  and  $\mathbf{x}'$ . The former is obtained by adding an entry  $-\sqrt{\|\mathbf{w}_c\|_2^2}$  to  $\mathbf{w}_c$ , and the latter by adding a 0 to  $\mathbf{x}$ . Consequently, maximizing inner products in  $\mathcal{X}$  is equivalent to minimizing distances in the augmented space  $\mathcal{X}'$ .<sup>1</sup>

A reduction, as outlined above, is interesting because many methods for efficient nearest neighbor search have been proposed in the literature. In this work, we use Hierarchical Navigable Small World (H-NSW) graphs, introduced by Malkov and Yashunin (2018). This method is based on the concept of a similarity graph (Navarro 2002), in which edges connect similar objects. In our case, these objects are weight vectors  $\mathbf{w}_c$ . H-NSW uses multiple such graphs, each on a different level. The lowest level contains all objects, while higher levels contain successively sparser subsets of these objects. Roughly speaking, the idea is to search the nearest neighbors of the query  $\mathbf{x}$  level-wise (by traversing the graph for the layer in a greedy way), starting with the highest level. The neighbors found on each level do not necessarily correspond to the true neighbors of  $\mathbf{x}$  on the lowest level, but should at least indicate the region in which these neighbors are located, and hence provide a good entry point for refining the search on the next level. For technical details and the complete pseudocode of the H-NSW method, we refer to Malkov and Yashunin (2018). What the algorithm eventually returns is a list of  $s$  weight vectors  $\mathbf{w}_c$  for the  $s$  classes that have (approximately) the highest inner products with the test example  $\mathbf{x}$ .

We conclude this section with two remarks. First, finding the top- $s$  classes may not be enough to satisfy the stopping criterion of Theorem 2. To solve this problem, we use a simple doubling strategy. When  $PC$  is requested to provide the  $(s + 1)$ -st class, we double the value of  $s$  and query the H-NSW index again. Since the nearest neighbor search is approximate, we append all new labels to the original list. In this way, we do not omit any new label with a probability higher than the minimum probability of labels found in the previous query. Second, this search method should lead to a faster inference than SVBOP-Full, as the number of inner products should be lower than a number required to compute all conditional class probabilities. While this method significantly speeds up inference, it adds additional cost to the training phase, due to the need to construct the H-NSW index. For training, one also relies on multi-class classifiers that scale linearly with the number of classes.

<sup>1</sup> Practically, this augmentation is often omitted for simplicity.

**Algorithm 4** SVBOP-HSG.initPrediction – **input:**  $\mathbf{x}, \mathcal{Y} = \{c_1, \dots, c_K\}$ 


---

```

1:  $\mathcal{Q} \leftarrow \emptyset$                                 ▷ Initialize a list of classes with their inner prod.  $\mathbf{w}_c \cdot \mathbf{x}$ 
2:  $i \leftarrow 0$                                 ▷ Initialize the class counter
3:  $\mathcal{Q}' \leftarrow \text{H-NSW Index.query}(\mathbf{x}, k)$     ▷ Query for initial top- $k$  elements
4: for  $c \in \mathcal{Q}'$  do                                ▷ For initial top- $k$  classes
5:    $\mathcal{Q}.\text{add}((c, \exp(\mathbf{w}_c \cdot \mathbf{x})))$           ▷ Transform inner prod. to unnorm.  $P(c|\mathbf{x})$  and add to list

```

---

**Algorithm 5** SVBOP-HSG.getNextClass

---

```

1:  $i \leftarrow i + 1$                                 ▷ Increment class counter
2: if  $|\mathcal{Q}| < i$  then                                ▷ If class counter is greater than size of list of classes
3:    $\mathcal{Q}' \leftarrow \text{H-NSW Index.query}(\mathbf{x}, 2 \times |\mathcal{Q}|)$     ▷ Perform doubling
4:   for  $c \in \mathcal{Q}' / \mathcal{Q}$  do                                ▷ For each new class found by index
5:      $\mathcal{Q}.\text{add}((c, \exp(\mathbf{w}_c \cdot \mathbf{x})))$           ▷ Transform inner prod. to unnorm.  $P(c|\mathbf{x})$  and add to list
6: return  $\mathcal{Q}.\text{at}(i)$                                 ▷ Return next class from the list

```

---

**3.3 Hierarchical factorization of the conditional distribution (SVBOP-HF)**

To have a compatible probabilistic classifier that allows for efficient prediction of top- $s$  classes, while having a much faster training at the same time, we investigate in this subsection a solution based on a hierarchical factorization of the distribution  $P(c | \mathbf{x})$ . This approach underlies many popular algorithms, such as nested dichotomies (Fox 1997; Frank and Kramer 2004; Melnikov and Hüllermeier 2018), conditional probability estimation trees (Beygelzimer et al. 2009), probabilistic classifier trees (Dembczyński et al. 2016), or hierarchical softmax (Morin and Bengio 2005), often used in neural networks as an output layer.

With a hierarchical tree structure over the classes, where the root represents the class space and leafs the singleton sets of classes, one can express the conditional class probability  $P(c | \mathbf{x})$  via the chain rule of probability:

$$P(c | \mathbf{x}) = \prod_{v \in \text{Path}(c)} P(v | \text{Parent}(v), \mathbf{x}), \quad (12)$$

where  $\text{Path}(c)$  is a set of nodes on the path connecting the leaf and the root of the tree structure.  $\text{Parent}(v)$  gives the parent of node  $v$ , and for the root node  $r$  we have  $P(r | \text{Parent}(r), \mathbf{x}) = 1$ . In each node of the tree, we train a multi-class probabilistic classifier of choice.

For inference, we adapt an  $A^*$ -style algorithm, closely related to search methods used with probabilistic tree classifiers (Dembczyński et al. 2012, 2016; Mena et al. 2017). It uses a priority queue for storing visited nodes in the order of their decreasing conditional class probabilities. The queue is initialized with the root node (Algorithm 6). In the main loop, for each iteration, the next label is returned in order of decreasing conditional class probabilities. This is achieved by visiting nodes one by one, taking them from the queue and adding for each visited node its children to the queue (Algorithm 7).

On average, this search should result in significantly faster inference than the standard SVBOP-Full algorithm, as only a part of the tree will be explored. Optimistically,

**Algorithm 6** SVBOP-HF.initPrediction – **input:**  $\mathbf{x}, \mathcal{Y} = \{c_1, \dots, c_K\}$ 


---

```

1:  $\mathcal{Q} = \emptyset$  ▷ Initialize a priority queue
2:  $\mathcal{Q}.\text{add}((v_{\text{root}}, P(v_{\text{root}}|\mathbf{x})))$  ▷ Add the tree root with the corresponding  $P(v|\mathbf{x})$ 

```

---

**Algorithm 7** SVBOP-HF.getNextClass

---

```

1: while  $\mathcal{Q} \neq \emptyset$  do ▷ While a priority queue is not empty
2:    $(v, p_v) \leftarrow \mathcal{Q}.\text{pop}()$  ▷ Pop the node with highest  $P(v|\mathbf{x})$  from the queue
3:   if  $v$  is a leaf then ▷ If the node is a leaf node
4:     return  $(\text{Class}(v), p_v)$  ▷ Return corresponding class and  $P(c|\mathbf{x})$ 
5:   for  $v' \in \text{Children}(v)$  do ▷ Else for all child nodes of  $v$ 
6:      $p_{v'} \leftarrow p_v \times P(v'|v, \mathbf{x})$  ▷ Computed probability estimate to the child node
7:      $\mathcal{Q}.\text{add}((v', p_{v'}))$  ▷ Add the child node and its  $P(v'|\mathbf{x})$  to the priority queue

```

---

the speedup can be even exponential (i.e., the query for a single  $\mathbf{x}$  can proceed in logarithmic time in the number of classes  $K$ ). Yet, in the worst case, the algorithm can visit all nodes in the tree, a number that is upper bounded by  $2K - 1$ . With specific optimization algorithms, such as the ones used for hierarchical softmax implementations in deep neural networks, the hierarchical factorization might also lead to much faster training. One only needs to update nodes on a path from the root to a leaf corresponding to the class label of the example. This results in logarithmic training times in terms of the number of classes, assuming that the tree is balanced.

Similarly, as in the previous approach, there is an additional step required for building the hierarchical structure before training. This structure can be obtained from data. For example, Huffman trees are commonly used for similar algorithms in natural language processing problems (Mikolov et al. 2013). More involved learning algorithms, such as the one used in (Prabhu et al. 2018) run hierarchical balanced 2-means on class profiles. In some applications, a natural hierarchy may exist and this one can be used as well, as we will show in the experiments.

## 4 Related work

The paper that is the closest to our work is (Del Coz et al. 2009), in which an efficient algorithm for the  $F_\beta$ -measure is proposed. Our work extends this paper in two directions: 1) we introduce a general optimization framework that generalizes the results of Del Coz et al. (2009) to other utility functions, and 2) we also develop efficient algorithms that further improve their algorithm, which can be interpreted as a specific case of the SVBOP-Full algorithm.

We discussed several papers that introduce various set-based utility functions (Corani and Zaffalon 2008, 2009; Zaffalon et al. 2012; Yang et al. 2017b; Nguyen et al. 2018; Ramaswamy et al. 2015). Those papers mainly highlight the usefulness and properties of these functions, while focussing less on algorithmic aspects. From that perspective, we rather see our work as complementary instead of competing.

In the literature, one can find several simple approaches to generate set-valued predictions. Arguably, the most simple approach is to look at the conditional class probabilities, and return a predefined number of  $s$  classes; the classes with the highest

conditional class probabilities (top- $s$  prediction). The main downside of this approach is that set-valued predictions for different instances will have the same cardinality. In practice, this is often unwanted; small sets should be returned when the uncertainty about the true class label is small, while bigger sets are needed when the uncertainty becomes larger. Another simple approach is thresholding on conditional class probabilities. One can define a fixed threshold for  $P(c^{(i)} | \mathbf{x})$  and return those classes that exceed this threshold, or one can define a threshold on the cumulative probability of the resulting set. In the latter case, one first sorts the classes in decreasing order of conditional class probabilities. For a user-defined  $\theta \in [0, 1]$ , one then returns the top- $s$  classes for which  $s$  is given by

$$\inf \left\{ s : \sum_{i=1}^s P(c^{(i)} | \mathbf{x}) \geq \theta \right\}, \quad (13)$$

with  $c^{(1)}, \dots, c^{(K)}$ . Both thresholding approaches have a clear disadvantage: they only look at a specific threshold, and do not account for the fact that the size of the predicted set might considerably change if the threshold is slightly changed. Thresholding will be suboptimal in view of optimizing (1). This can be best observed from (8), which proves that the Bayes-optimal set not only depends on the sum of the first  $s$  conditional probabilities, but also on the next probability in the sorted list, i.e.,  $p_{s+1}$ . Nonetheless, top- $s$  prediction and thresholding are two obvious baselines that will be analyzed in our experiments.

In the work of Stock et al. (2016), Fagin's algorithm and Fagin's thresholding algorithm are described for computing the top- $s$  efficiently in a wide range of multi-target prediction settings, including multi-label classification, dyadic prediction and collaborative filtering. Interestingly, both algorithms can be applied in combination with the proposed SVBOP-Full algorithm, as well as top- $s$  prediction. The two algorithms are guaranteed to find the top- $s$  by using data structures that avoid the iteration over all labels. They can realize a speed-up compared to a naive implementation that iterates over all labels, especially in situations where the dimensionality of learned hidden representations in deep learning models or input space for linear models is low.

Set-valued predictions are also considered in hierarchical multi-class classification, mostly in the form of internal nodes of the hierarchy (Freitas 2007; Rangwala and Naik 2017; Yang et al. 2017a). Compared to the "flat" multi-class case, the prediction space is thus restricted, because only sets of classes that correspond to nodes of the hierarchy can be returned as a prediction. In this paper, we do not consider such a setup, but the SVBOP-HF algorithm could be adjusted for that purpose.

Set-based utility functions have been analyzed in the context of hierarchical multi-class classification. For example, Yang et al. (2017a) evaluate various members of the  $u_{\delta, \gamma}$  family in a framework where hierarchies are considered for computational reasons, while Oh (2017) optimizes recall by fixing  $|\hat{Y}|$  as a user-defined parameter. Popular in hierarchical multi-label classification is the tree-distance loss, which could also be interpreted as a way of evaluating set-valued predictions (see e.g. (Bi and Kwok 2015)). This loss is not a member of the family (2), however. Besides, from the



perspective of abstention in the case of uncertainty, it appears to be a less useful loss function, because it has the tendency to return large sets.

Set-valued predictions are also produced in the framework of conformal prediction (CP) (Vovk et al. 2003; Shafer and Vovk 2008; Balasubramanian et al. 2014; Denis and Hebiri 2017). This framework is rooted in statistical hypothesis testing, and in a sense can be seen as a “frequentist statistics” counterpart to our approach, which is more in the spirit of Bayesian decision theory. More specifically, given a new query instance  $\mathbf{x}$ , CP constructs a prediction set or *prediction region*  $Y \subseteq \mathcal{Y}$  that is guaranteed to cover the true outcome  $y$  with a pre-defined probability  $1 - \epsilon$  (for example 95 %). To this end, the hypothesis that  $y = \hat{y}$  is tested for each candidate prediction  $\hat{y} \in \mathcal{Y}$ , and only those candidates for which the test can be rejected are excluded from  $Y$ . The test itself is non-parametric and leverages a “nonconformity” function  $s : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$  that assigns scores  $s(\mathbf{x}, y) \in \mathbb{R}$  to input/output tuples; the latter can be interpreted as a measure of “strangeness” of the pattern  $(\mathbf{x}, y)$ , i.e., the higher the score, the less the data point  $(\mathbf{x}, y)$  conforms to what one would expect to observe. Roughly speaking, the hypothesis  $y = \hat{y}$  is then rejected if the nonconformity score  $s(\mathbf{x}, \hat{y})$  is among the  $\epsilon$  % highest of all scores  $s(\mathbf{x}_1, y_1), \dots, s(\mathbf{x}_N, y_N)$  observed in the (training) data so far. Conformal prediction has originally been introduced as an online learning method, but inductive variants have been developed as well (Papadopoulos 2008).

As we are maximizing a set-based utility function, our work is also somewhat connected to submodular optimization in machine learning (Syed 2016; Vondrak 2019). From (3) it follows that  $U(\hat{Y}, P, u)$  is either a submodular or supermodular function, when  $g$  is concave or convex, respectively. In the first case, when  $g$  is decreasing and concave, then  $g(|\hat{Y}|)$  is submodular, and because  $P(\hat{Y} | \mathbf{x})$  is modular,  $U(\hat{Y}, P, u)$  must be submodular. One could therefore think of using submodular optimization algorithms to solve (1), but we believe that such algorithms would not be very useful. Theorem 1 made us conclude that the combinatorial problem (1) could be reduced to a line search on  $K + 1$  sets. In contrast, a line search cannot be applied in combinatorial machine learning problems that are solved with submodular optimization techniques. To give one example, submodular optimization of (1) by means of a continuous relaxation via the Lovasz extension (Vondrak 2019) would still require the computation of  $P(c | \mathbf{x})$  for all classes  $c$ , whereas the algorithms that we introduce avoid this.

Currently, the development of efficient algorithms is an active theme of research in the area of extreme classification. The overwhelming majority of algorithms developed in this area focus on multi-label classification, but also some algorithms for multi-class problems, with a large number of classes, have been proposed. Such algorithms are not immediately applicable to the setting of set-valued prediction. We believe that the proposed framework, albeit with some modifications, together with the approximate inference algorithms could be an interesting direction for extreme multi-label classification with set-valued predictions. The idea of sorting probabilities is commonly used in extreme multi-label classification, for optimizing specific loss functions, such as the  $F_\beta$ -measure, precision, and normalized discounted cumulative gain, see e.g. (Waegeman et al. 2014; Babbar and Dembczyński 2018). For several of those measures, one typically either selects the top- $s$  scoring labels (with  $s$  predefined), or those labels for which the marginal probability exceeds a threshold. Those two approaches are suboptimal in view of optimizing (1), but it is interesting to see how much we

**Table 2** Summary of image (top) and text (bottom) datasets used in all experiments. Notation:  $K$  – number of classes,  $D$  – number of features,  $N$  – number of samples




Dataset	$K$	$D$	$N_{train}$	$N_{test}$
MNIST (LeCun and Cortes 2010)	10	32	33600	8400
VOC 2006 (Everingham et al. 2006)	10	25088	1398	1477
VOC 2007 (Everingham et al. 2007)	20	25088	2808	2841
Caltech-101 (Li et al. 2003)	97	25088	4338	4339
Caltech-256 (Griffin et al. 2007)	256	25088	14890	14890
ALOI.BIN (Geusebroek et al. 2005)	1000	636911	90000	8000
DBpedia (Ofer 2019)	219	483214	240942	96797
Bacteria (RIKEN 2013)	2659	2472	10587	2294
Proteins (Li et al. 2018)	3485	26276	11830	10179
DMOZ (Partalas et al. 2015)	11939	833484	335068	38340
LSHTC1 (Partalas et al. 2015)	12166	381571	93805	34905

gain with our more complicated algorithms. That is something we will analyze in the experiments.

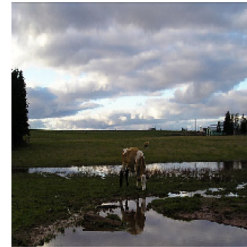
Finally, set-valued prediction is also closely connected to uncertainty modelling for multi-class classification. For safety-critical applications such as self-driving cars and medical decision making, it is important to have an indication of uncertainty when making decisions. Some recently developed methods make a distinction between epistemic and aleatoric uncertainty, see e.g. (Hüllermeier and Waegeman 2019) for an overview. The former type of uncertainty arises due to a lack of data for training, whereas the latter alludes to uncertainty that cannot be reduced by collecting more data, e.g. measurement noise, low-quality features, etc. In our framework, we only consider aleatoric uncertainty, because we produce a set based on the estimated conditional class probabilities  $P(c | \mathbf{x})$ . Approaches that analyze epistemic uncertainty take other properties of the data into account, such as generalizations of probability theory (Senge et al. 2014; Nguyen et al. 2018) or measures based on dropout resampling at test time (Kendall and Gal 2017; Depeweg et al. 2018). Very recently, Ziyin et al. (2019) combine the idea of set-based utility maximization with density estimation, producing an empty set in case of high epistemic uncertainty. This might also be an interesting path for future work.

## 5 Experiments

We perform three types of experiments to illustrate and benchmark the algorithms that we introduce. The datasets for all experiments are shown in Table 2. In the following section, we first illustrate the practical relevance of set-valued prediction on both image and text classification datasets. In Sect. 5.2, we evaluate the proposed exact algorithm, together with simple baseline methods that produce sets, for different set-based utility functions on a wide range of practical datasets. For the last group of experiments,

			
$\beta = 5$	{ <u>8</u> }	{ <u>3</u> , 5, 9}	{9, 3, <u>2</u> , 7, 8, 1}
$\beta = 2$	{ <u>8</u> }	{ <u>3</u> }	{9, 3, <u>2</u> }
$\beta = 1$	{ <u>8</u> }	{ <u>3</u> }	{9, 3}
	$H = 10^{-5}$	$H = 0.56$	$H = 2.54$

(a) Predictions for three MNIST test samples, for increasing  $\beta \in \{1, 2, 5\}$  and Shannon entropy  $H$ .



(b) VOC 2006 test sample with top-5 prediction {sheep, cow, horse, car, motorbike}. SVBOP-Full candidates, for  $\beta = 1$ , are indicated in bold.

Lance Ten Broeck (born March 21, 1956) is an American professional golfer who has played on the PGA Tour, Nationwide Tour, and Champions Tour. Ten Broeck was born in Chicago, Illinois, and grew-up in Beverly, a community on the city's southwest side...	MTS TV is a digital television service owned by Telekom Srbija. The service provides thematic channels, HD channels, video on demand, video recording, the use of an Electronic Program Guide (EPG) and other services...	Liao Bingxiong was a Chinese political cartoonist, painter and calligrapher. He remained active from 1934 until he gave up in 1995 (with a 20-year break between 1957 and 1978). Liao is widely regarded as one of China's foremost political cartoonists...
{ <u>GolfTournament</u> , <u>GolfPlayer</u> , RugbyPlayer, SoccerPlayer, CyclingRace}	{ <u>TelevisionStation</u> , <u>BroadcastNetwork</u> , BusCompany, Comedian, RailwayLine}	{ <u>Painter</u> , <u>ComicsCreator</u> , PoliticalParty, President, Philosopher}

(c) DBpedia test samples with corresponding top-5 predictions. SVBOP-Full candidates, for  $\beta = 1$ , are indicated in bold.

**Fig. 2** Set-valued predictions with SVBOP-Full and utility function  $g_{F\beta}$  illustrated on MNIST, VOC 2006 and DBpedia. Ground truths are underlined in each prediction

in Sect. 5.3, we compare the proposed exact and approximate algorithms by looking at runtime efficiency versus predictive performance. For a general discussion on the experimental setup, we refer the reader to the appendix.

## 5.1 Illustrations on image datasets

In the illustrative experiments, we provide some examples that emphasize the practical usefulness of set-valued prediction. In Fig. 2a we show predictions obtained by the SVBOP-Full algorithm with utility function  $g_{F\beta}$ , on three MNIST test samples. From left to right, we show three test instances for which the uncertainty (in the conditional class probabilities) is increasing. To this end, uncertainty is expressed by the Shannon entropy

$$H = \sum_{i=1}^K P(c_i | \mathbf{x}) \log P(c_i | \mathbf{x}).$$

From top to down, we show predictions for each image in function of decreasing  $\beta \in \{5, 2, 1\}$ . For increasing  $\beta$  and uncertainty, the SVBOP-Full algorithm typically predicts larger sets. For the last image, corresponding to number two, one can see that predicting the class with the highest conditional class probability, i.e., the first element in the predicted set,<sup>2</sup> would result in a false positive. However, for  $\beta \in \{2, 5\}$ , the ground truth is returned as candidate solution in the set-valued predictions.

We further illustrate the usefulness of set-valued prediction by looking at another image (VOC 2006) and a couple of text (DBpedia) examples in Fig. 2b and c. There we show top-5 predictions and predictions obtained by SVBOP-Full (denoted in bold), by using utility function  $g_{F1}$ . For the VOC 2006 test image, with a cow and sheep in the background, the uncertainty reflected in the conditional class probabilities is high, most likely due to taking the picture in low light conditions. Again, returning the label with the highest conditional class probability (i.e., sheep) results in a false positive. A set-valued prediction by means of the simple top-5 or SVBOP-Full method, however, includes the ground truth as candidate solution for this particular case. A second observation is the suboptimality of the top-5 method, compared to SVBOP-Full. As the size of the prediction does not depend on the conditional class probabilities, but is rather fixed, we are at risk of including irrelevant candidates in the predicted set. These irrelevant candidates are most often characterized by low conditional class probabilities, for which an inclusion would result in a drop of expected utility in the SVBOP-Full algorithm (Algorithm 1). For example, for the image of the cow in Fig. 2b, two irrelevant candidates  $\{car, motorbike\}$  are included in the top-5 prediction. The same can be observed for the examples in Fig. 2c.

## 5.2 Comparison of different utility functions and baselines

The goal of the second type of experiments is to evaluate the SVBOP-Full algorithm for several utility functions. Two different parameterizations of the  $u_{\delta, \gamma}$  and  $u_{F\beta}$  families are studied, leading to four utility functions in total. We benchmark the SVBOP-Full algorithm against several baselines (which are all described in the related work section):

1. **Thresholding:** predicting those classes for which the total cumulative probability mass exceeds a user-defined threshold, as explained by (13). For each of the four utility functions we tune the threshold on a validation set, by considering ten equally-spaced values. As a result, we obtain four different thresholding strategies, each of them tailored for a specific utility function.
2. **Top- $s$ :** predicting a set consisting of the  $s$  classes with highest conditional class probabilities. Here we consider three versions, with  $s \in \{1, 3, 5\}$ . As discussed in the related work section, top- $s$  returns a fixed number of classes for each instance,

<sup>2</sup> Note that the candidate solutions in the set are sorted in decreasing order of conditional class probability.

**Table 3** Comparison of SVBOP-Full with baselines (thresholding, top- $s$  and inductive conformal prediction) in terms of optimizing different utility functions (listed in decreasing order of convexity:  $u_{\delta=1.6, \gamma=0.6}$ ,  $u_{F1}$ ,  $u_{\delta=2.2, \gamma=1.2}$  and  $u_{F5}$ ) for all datasets.

Method	VOC 2006				VOC 2007			
	$u_{\delta=1.6, \gamma=0.6}$	$u_{F1}$	$u_{\delta=2.2, \gamma=1.2}$	$u_{F5}$	$u_{\delta=1.6, \gamma=0.6}$	$u_{F1}$	$u_{\delta=2.2, \gamma=1.2}$	$u_{F5}$
SVBOP-Full- $u_{\delta=1.6, \gamma=0.6}$	91.03	91.11	91.73	92.51	88.93	89.01	89.62	90.39
SVBOP-Full- $u_{F1}$	91.13	91.22	91.90	92.82	88.84	88.94	89.64	90.62
SVBOP-Full- $u_{\delta=2.2, \gamma=1.2}$	91.14	<u>91.31</u>	<u>92.61</u>	94.27	88.73	88.91	<u>90.27</u>	92.06
SVBOP-Full- $u_{F5}$	87.76	88.35	90.74	<u>97.12</u>	83.60	84.29	86.80	<u>94.74</u>
Threshold- $u_{\delta=1.6, \gamma=0.6}$	87.84	88.40	90.64	96.73	83.72	84.36	86.43	94.18
Threshold- $u_{F1}$	87.84	88.40	90.64	96.73	83.72	84.36	86.43	94.18
Threshold- $u_{\delta=2.2, \gamma=1.2}$	87.84	88.40	90.64	96.73	83.72	84.36	86.43	94.18
Threshold- $u_{F5}$	86.92	87.54	89.80	96.71	83.72	84.36	86.43	94.18
Top-1	90.72	90.72	90.72	90.72	<u>88.95</u>	88.95	88.95	88.95
Top-3	45.91	49.19	59.03	91.35	44.94	48.15	57.78	89.43
Top-5	29.46	33.18	39.01	86.26	29.03	32.69	38.44	84.99
ICP- $\epsilon=0.01$	87.06	87.64	90.09	96.23	70.44	71.67	74.65	92.45
ICP- $\epsilon=0.10$	89.72	89.72	89.72	89.72	88.94	<u>89.02</u>	89.64	90.40
Caltech-101								
SVBOP-Full- $u_{\delta=1.6, \gamma=0.6}$	95.29	<u>95.36</u>	95.83	96.49	88.47	88.50	88.76	89.08
SVBOP-Full- $u_{F1}$	95.24	95.33	<u>95.84</u>	96.67	88.49	88.52	88.82	89.18
SVBOP-Full- $u_{\delta=2.2, \gamma=1.2}$	94.73	94.86	95.77	97.08	88.65	88.72	89.29	89.99
SVBOP-Full- $u_{F5}$	87.89	88.41	90.10	96.93	88.57	88.76	90.07	91.91
Threshold- $u_{\delta=1.6, \gamma=0.6}$	81.90	82.31	83.31	91.64	88.48	88.70	90.22	92.37
Threshold- $u_{F1}$	81.90	82.31	83.31	91.64	88.48	88.70	90.22	92.37
Threshold- $u_{\delta=2.2, \gamma=1.2}$	81.90	82.31	83.31	91.64	88.09	<u>88.38</u>	<u>90.30</u>	93.16
DBpedia								

Table 3 continued

	Caltech-101			DBpedia		
Threshold- $u_{F5}$	81.90	82.31	83.31	91.64	87.46	90.18
Top-1	95.07	95.07	95.07	95.07	88.22	88.22
Top-3	46.02	49.31	59.17	91.57	45.48	58.47
Top-5	29.37	33.08	38.90	86.01	29.22	38.70
ICP $_{\epsilon=0.01}$	89.31	89.83	91.64	97.69	65.04	72.30
ICP $_{\epsilon=0.10}$	95.34	95.34	95.34	95.34	88.04	88.66
Caltech-256						
SVBOP-Full- $u_{\delta=1.6, \gamma=0.6}$	81.90	82.04	83.06	84.46	96.38	96.66
SVBOP-Full- $u_{F1}$	81.91	82.09	83.21	85.02	96.34	96.65
SVBOP-Full- $u_{\delta=2.2, \gamma=1.2}$	80.97	81.27	83.39	86.39	96.25	96.86
SVBOP-Full- $u_{F5}$	69.47	70.60	73.74	89.06	94.04	95.23
Threshold- $u_{\delta=1.6, \gamma=0.6}$	69.73	70.48	72.28	85.59	89.56	90.54
Threshold- $u_{F1}$	69.73	70.48	72.28	85.59	89.56	90.54
Threshold- $u_{\delta=2.2, \gamma=1.2}$	69.73	70.48	72.28	85.59	89.56	90.54
Threshold- $u_{F5}$	69.73	70.48	72.28	85.59	89.56	90.54
Top-1	81.46	81.46	81.46	81.46	96.43	96.43
Top-3	42.52	45.56	54.67	84.61	46.10	59.26
Top-5	27.77	31.27	36.78	81.31	29.38	38.91
ICP $_{\epsilon=0.01}$	45.43	46.77	49.49	75.23	93.33	94.71
ALOI.BIN						
SVBOP-Full- $u_{\delta=1.6, \gamma=0.6}$	81.90	82.04	83.06	84.46	96.38	96.66
SVBOP-Full- $u_{F1}$	81.91	82.09	83.21	85.02	96.34	96.65
SVBOP-Full- $u_{\delta=2.2, \gamma=1.2}$	80.97	81.27	83.39	86.39	96.25	96.86
SVBOP-Full- $u_{F5}$	69.47	70.60	73.74	89.06	94.04	95.23
Threshold- $u_{\delta=1.6, \gamma=0.6}$	69.73	70.48	72.28	85.59	89.56	90.54
Threshold- $u_{F1}$	69.73	70.48	72.28	85.59	89.56	90.54
Threshold- $u_{\delta=2.2, \gamma=1.2}$	69.73	70.48	72.28	85.59	89.56	90.54
Threshold- $u_{F5}$	69.73	70.48	72.28	85.59	89.56	90.54
Top-1	81.46	81.46	81.46	81.46	96.43	96.43
Top-3	42.52	45.56	54.67	84.61	46.10	59.26
Top-5	27.77	31.27	36.78	81.31	29.38	38.91
ICP $_{\epsilon=0.01}$	45.43	46.77	49.49	75.23	93.33	94.71

Table 3 continued

Caltech-256				ALOI.BIN			
ICP <sub><math>\epsilon=0.10</math></sub>	77.10	77.77	80.95	87.80	96.18	96.18	96.18
Bacteria				Proteins			
SVBOP-Full- $u_\delta=1.6, \gamma=0.6$	94.50	94.54	94.87	95.29	83.30	83.34	83.59
SVBOP-Full- $u_{F1}$	94.55	94.60	94.96	95.44	83.29	83.33	83.61
SVBOP-Full- $u_\delta=2.2, \gamma=1.2$	94.42	94.50	95.13	95.94	<u>83.34</u>	<u>83.42</u>	83.96
SVBOP-Full- $u_{F5}$	93.42	93.66	94.98	97.46	83.04	83.28	<u>84.45</u>
Threshold- $u_\delta=1.6, \gamma=0.6$	93.85	94.05	95.17	97.23	81.31	81.52	82.50
Threshold- $u_{F1}$	93.85	94.05	95.17	97.23	81.31	81.52	82.50
Threshold- $u_\delta=2.2, \gamma=1.2$	93.85	94.05	95.17	97.23	81.31	81.52	82.50
Threshold- $u_{F5}$	92.48	92.80	94.22	97.72	81.31	81.52	82.50
Top-1	94.29	94.29	94.29	94.29	83.07	83.07	83.07
Top-3	46.20	49.50	59.40	91.93	42.84	45.90	55.08
Top-5	29.41	33.12	38.94	86.10	27.70	31.20	36.69





which can be considered as a limitation. However, it is interesting to see how this suboptimal approach performs w.r.t. set-based utility functions.

3. Inductive conformal prediction (ICP): we experiment with the commonly-used nonconformity function  $s(\mathbf{x}, y) = 1 - P(y | \mathbf{x})$ , and consider two fixed significance levels  $\epsilon \in \{0.01, 0.10\}$ .

Table 3 shows for all methods the results obtained on test sets, where the highest obtained utilities are underlined. The utility functions are ordered in decreasing order of convexity:  $u_{\delta=1.6, \gamma=0.6}$ ,  $u_{F1}$ ,  $u_{\delta=2.2, \gamma=1.2}$  and  $u_{F5}$ . The first three utility functions all behave very similar to precision, which explains why the results are similar. Due to a higher convexity, these utility functions give a high reward to small sets, such that the top-1 in general yields very good results for those utility functions. At the other side, for  $u_{F5}$  the picture looks different; there top-3 or top-5 are often much better than top-1, because this utility function gives a higher reward to larger sets.

The performance of the SVBOP-Full algorithm is in accordance with our theoretical results. In general, it is one of the best-performing methods for all datasets and utility functions that were analyzed. However, the differences with the other methods are small. This is, of course, not very surprising because all tested inference algorithms depart from the same conditional class probabilities. Differences in performance can only be attributed to (relatively small) differences in the inference algorithms. As discussed in Sect. 4, thresholding is not Bayes-optimal w.r.t. (1), but on the analyzed datasets it performs quite well. We can conclude that the theoretical shortcomings of thresholding will only lead to small performance drops in practice.

Finally, inductive conformal prediction performs quite well on some datasets, but this method yields bad results on other datasets. This can, of course, be explained by the fact that inductive conformal prediction does not intend to maximize a utility function. As explained in Sect. 4, this method rather intends to return sets that contain the true class label with high confidence. This phenomenon is especially visible on the LSHTC and DMOZ datasets, where  $K$  is large. Then, inductive conformal prediction will produce very large sets, when it wants to cover the true class with high probability.

### 5.3 Comparison of exact and approximate algorithms on large datasets

In the final group of experiments, we would like to compare the proposed exact and approximate algorithms by looking at runtime efficiency versus predictive performance.

Additionally, we also compare our proposed work with Fagin's algorithm (Fagin) and Fagin's thresholding algorithm (Fagin<sub>thr</sub>). Table 4 summarizes the results for the SVBOP-Full, SVBOP-Fagin, SVBOP-Fagin<sub>thr</sub>, SVBOP-HSG, and SVBOP-HF approaches, obtained on the five largest datasets (w.r.t. the number of classes). We use the same weights for SVBOP-Full, SVBOP-Fagin, SVBOP-Fagin<sub>thr</sub>, and SVBOP-HSG algorithms. For SVBOP-HF, we consider a predefined hierarchy (HF<sub>p</sub>), if available, and a hierarchy constructed during training by means of hierarchical balanced 2-means on class profiles (HF<sub>c</sub>), as explained in Sect. 3. For each algorithm we optimize two different utility functions:  $u_{F1}$  and  $u_{\delta=2.2, \gamma=1.2}$ . We report train and test time, as well as average utility, recall ( $\mathbb{1}_{y_i \in \hat{Y}(\mathbf{x}_i)}$ ) and size of the predicted sets.

**Table 4** Performance versus runtime for the SVBOP-Full, SVBOP-Fagin, SVBOP-Fagin<sub>hr</sub>, SVBOP-HSG and SVBOP-HF algorithms, tested on five benchmark datasets for F1-measure utility ( $u_{F1}$ ) and credal utility with  $\delta = 2.2$  and  $\gamma = 1.2$  ( $u_{\delta,\gamma}$ )

Dataset	Algo.	$t_{ti, tr}$	$t_{ti, te}$	$t_{train}$	Top-1	$u_{F1}$	R	$ \hat{Y} $	$t_{test}$	$u_{\delta,\gamma}$	R	$ \hat{Y} $	$t_{test}$
ALOI.BIN	Full	–	–	5065	96.43	96.38	97.11	1.03	4.89	96.86	97.68	1.05	4.74
	Fagin			5067	96.43	96.38	97.11	1.03	5.96	96.86	97.68	1.05	6.67
	Fagin <sub>hr</sub>			5067	96.43	96.38	97.11	1.03	6.30	96.86	97.68	1.05	6.45
	HSG			5087	96.41	96.23	96.96	1.04	2.55	96.66	97.54	1.05	3.11
	HF <sub>c</sub>			163	93.15	93.46	95.16	1.06	0.28	93.97	95.44	1.10	0.28
Bacteria	Full		0.30	6303	94.24	94.60	95.55	1.04	3.84	95.12	96.12	1.06	5.22
	Fagin			6314	94.24	94.60	95.55	1.04	52.20	95.12	96.12	1.06	53.06
	Fagin <sub>hr</sub>			6314	94.24	94.60	95.55	1.04	47.00	95.12	96.12	1.06	49.57
	HSG			6323	92.45	93.16	94.11	1.09	1.83	93.56	95.34	1.12	1.92
	HF <sub>p</sub>			360	94.11	94.46	95.53	1.05	0.26	95.56	96.46	1.07	0.28
Proteins	HF <sub>c</sub>			70	93.81	93.38	95.20	1.06	0.14	94.46	96.64	1.11	0.16
	Full		0.21	2192	83.59	83.89	84.74	1.06	13.53	84.54	85.50	1.10	14.28
	Fagin			2198	83.59	83.89	84.74	1.06	55.97	84.54	85.50	1.10	61.92
	Fagin <sub>hr</sub>			2198	83.59	83.89	84.74	1.06	52.77	84.54	85.50	1.10	59.31
	HSG			2672	69.58	69.83	70.61	1.22	10.95	69.46	70.38	1.27	11.35
	HF <sub>p</sub>			77	81.86	81.17	82.54	1.03	0.24	82.36	82.71	1.05	0.25

Table 4 continued

Dataset	Algo.	$t_{ti, tr}$	$t_{ti, te}$	$t_{train}$	Top-1	$u_{F1}$	R	$ \hat{Y} $	$t_{test}$	$u_{\delta, \gamma}$	R	$ \hat{Y} $	$t_{test}$
DMOZ	HF <sub>c</sub>			22	79.99	80.18	80.89	1.04	0.11	80.58	81.09	1.06	0.11
	Full	–	–	82872	40.41	40.91	51.33	3.52	55.04	43.31	52.46	2.73	54.47
	Fagin			82889	40.41	40.91	51.33	3.52	620.54	43.31	52.46	2.73	645.75
	Fagin <sub>thr</sub>			82889	40.41	40.91	51.33	3.52	180.17	43.31	52.46	2.73	238.89
	HSG			83181	39.97	40.13	49.66	3.26	2.67	42.02	50.05	2.36	2.72
LSHTC1	HF <sub>c</sub>			722	38.03	22.79	46.70	7.15	10.94	25.79	45.44	5.32	10.01
	Full	–	–	71509	42.00	42.78	45.38	1.29	46.13	44.49	47.24	1.44	48.37
	Fagin			71515	42.00	42.78	45.38	1.29	672.71	44.49	47.24	1.44	645.89
	Fagin <sub>thr</sub>			71509	42.00	42.78	45.38	1.29	375.24	44.49	47.24	1.44	409.13
	HSG			72361	41.52	42.30	44.86	1.30	8.28	43.99	46.71	1.44	9.71
	HF <sub>p</sub>			557	39.82	40.96	44.79	1.42	0.52	43.20	47.21	1.60	0.60
	HF <sub>c</sub>			338	38.53	39.19	43.36	1.48	1.15	41.38	45.78	1.66	1.24

Notation: R – recall,  $u$  – utility value,  $|\hat{Y}|$  – prediction size,  $t_{ti, tr}$  – CPU time for calculating tf-idf representations on training set in seconds,  $t_{ti, te}$  – CPU time for calculating tf-idf representations on test set in milliseconds / number of test samples,  $t_{train}$  – CPU train time in seconds,  $t_{test}$  – CPU test time in milliseconds / number of test samples,  $p$  – predefined hierarchy,  $c$  – hierarchy built with hierarchical balanced 2-means clustering

We also report the times for calculating the hidden representations during training and testing, except for the datasets where hidden representations are provided (i.e., ALOI.BIN, DMOZ and LSHTC1). Additionally, we also include average recall for top-1 predictions; this is in essence accuracy.

For almost all datasets, SVBOP-Full yields the best predictive performance while being, as expected, always the slowest. For all datasets, SVBOP-HSG achieves a predictive performance that is very close to SVBOP-Full, while being at the same time even a few times faster in inference on DMOZ and LSHTC1 datasets. Unsurprisingly, hierarchical factorization leads to the highest speedup both in training and inference. However, for most datasets, it comes at the expense of predictive performance. Only for datasets where a meaningful natural hierarchy is given (i.e. biological datasets), SVBOP-HF<sub>p</sub> outperforms SVBOP-Full and SVBOP-HF<sub>c</sub>.

In general, we observe that for almost all datasets, both approximate algorithms behave similarly to SVBOP-Full and manage to significantly improve recall with an only small increase in average prediction size. At the same time, the approximate algorithms improve the test times at the cost of predictive performance. This is not very surprising, as one might expect a clear trade-off between the two. Furthermore, one can see that both Fagin's algorithms always yield exactly the same predictive performance as the SVBOP-Full algorithm, yet their runtime is much longer. In practice, the choice of a particular method should depend on the desired trade-off between runtime and predictive performance.

## 6 Conclusion

We introduced a decision-theoretic framework for a general family of set-based utility functions, including most of the measures used in the literature so far, and developed three Bayes-optimal inference algorithms that exploit specific assumptions to improve runtime efficiency. Depending on the concrete dataset, those assumptions may or may not affect predictive performance.

In future work, we plan to extend our decision-theoretic framework toward uncertainty representations more general than standard probability, for example taking up a distinction between so-called aleatoric and epistemic uncertainty recently put forward by several authors (Senge et al. 2014; Kendall and Gal 2017; Depeweg et al. 2018; Nguyen et al. 2018). Finally, some of the proposed algorithms could also be exploited in a set-valued prediction framework for multi-label classification.

**Acknowledgements** For this work Willem Waegeman received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" Programme.

## A Regret bounds for the utility functions

In this part we present a short theoretical analysis that relates the Bayes optimal solution for the set-based utility functions to the solution obtained on the probabilities given by a trained model. The goal is to upper bound the regret of the set-based utility

functions by the  $L_1$  error of the class probability estimates. The analysis is performed on the level of a single  $\mathbf{x}$ .

Let  $\hat{P}(\mathbf{x})$  be the estimate of the true underlying distribution  $P(\mathbf{x})$ . Let  $U^*(P, u)$  denote the optimal utility for  $P$  obtained by the optimal solution  $\hat{Y}^*$  (this solution does not have to be unique). Now, let  $\hat{Y}$  denote the optimal solution with respect to  $\hat{P}(\mathbf{x})$ . We define the regret of  $\hat{Y}$  as:

$$\begin{aligned} \text{reg}_u(\hat{Y}) &= U^*(P, u) - U(\hat{Y}, P, u) \\ &= \sum_{c \in \mathcal{Y}} u(c, \hat{Y}^*) P(c | \mathbf{x}) - \sum_{c \in \mathcal{Y}} u(c, \hat{Y}) P(c | \mathbf{x}) \\ &= \sum_{c \in \mathcal{Y}} \left( u(c, \hat{Y}^*) - u(c, \hat{Y}) \right) P(c | \mathbf{x}) \end{aligned}$$

We bound  $\text{reg}_u(\hat{P}(\mathbf{x}))$  in terms of the  $L_1$ -estimation error, i.e.:

$$\sum_{c \in \mathcal{Y}} |P(c | \mathbf{x}) - \hat{P}(c | \mathbf{x})|$$

Note that if  $\hat{Y}^* = \hat{Y}$  the regret is 0. Otherwise, we need to have

$$U(\hat{Y}, \hat{P}, u) \geq U(\hat{Y}^*, \hat{P}, u)$$

Thus, we can write

$$\begin{aligned} \text{reg}_u(\hat{Y}) &\leq U^*(P, u) - U(\hat{Y}, P, u) + U(\hat{Y}, \hat{P}, u) - U(\hat{Y}^*, \hat{P}, u) \\ &= \sum_{c \in \mathcal{Y}} \left( u(c, \hat{Y}^*) - u(c, \hat{Y}) \right) P(c | \mathbf{x}) + \sum_{c \in \mathcal{Y}} \left( u(c, \hat{Y}) - u(c, \hat{Y}^*) \right) \hat{P}(c | \mathbf{x}) \\ &= \sum_{c \in \mathcal{Y}} u(c, \hat{Y}^*) \left( P(c | \mathbf{x}) - \hat{P}(c | \mathbf{x}) \right) + \sum_{c \in \mathcal{Y}} u(c, \hat{Y}) \left( \hat{P}(c | \mathbf{x}) - P(c | \mathbf{x}) \right) \\ &\leq \sum_{c \in \mathcal{Y}} u(c, \hat{Y}^*) \left| P(c | \mathbf{x}) - \hat{P}(c | \mathbf{x}) \right| + \sum_{c \in \mathcal{Y}} u(c, \hat{Y}) \left| \hat{P}(c | \mathbf{x}) - P(c | \mathbf{x}) \right| \quad (14) \\ &= \sum_{c \in \mathcal{Y}} \left( u(c, \hat{Y}^*) + u(c, \hat{Y}) \right) \left| P(c | \mathbf{x}) - \hat{P}(c | \mathbf{x}) \right| \\ &\leq 2 \sum_{c \in \mathcal{Y}} \left| P(c | \mathbf{x}) - \hat{P}(c | \mathbf{x}) \right| \quad (15) \end{aligned}$$

The inequality in (14) follows from the properties of the absolute function,  $a \leq |a|$ , while the one in (15) holds because the utility functions are from the bounded interval,  $u(\cdot, \cdot) \in [0, 1]$ . We clearly see that the regret is upper bounded by the quality of the estimated probability distribution.

## B Generalized reject option utility and parameter bounds

In this part we analyze which values  $\alpha$  and  $\beta$  can take so that the  $g_{\alpha,\beta}$  family is lower bounded by precision. This family is visualized in Fig. 3. For a given  $K$ , the following inequality must hold  $\forall s \in \{1, \dots, K\}$ , such that  $g_{\alpha,\beta}(s)$  is lower bounded by precision:

$$g_{\alpha,\beta}(s) \geq g_P(s),$$

with utilities:

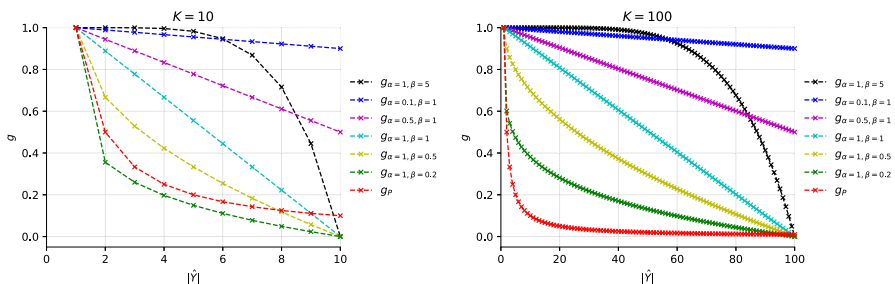
$$g_{\alpha,\beta}(s) = 1 - \alpha \left( \frac{s-1}{K-1} \right)^\beta, \quad g_P(s) = \frac{1}{s}.$$

When looking at the boundary cases (i.e.,  $s = 1, s = K$ ), we find that:

$$\alpha \leq \frac{K-1}{K}.$$

By fixing  $\alpha = \frac{K-1}{K}$ , the above inequality can be rewritten,  $\forall s \in \{2, \dots, K-1\}$ , as:

$$\begin{aligned} 1 - \left( \frac{K-1}{K} \right) \left( \frac{s-1}{K-1} \right)^\beta &\geq \frac{1}{s} \\ \Leftrightarrow \left( \frac{s-1}{K-1} \right)^\beta &\leq \frac{K}{s} \left( \frac{s-1}{K-1} \right) \\ \Leftrightarrow \beta &\geq \log_{\frac{s-1}{K-1}} \frac{K}{s} + 1 \\ \Rightarrow \beta &\geq \log_{\frac{1}{K-1}} \frac{K}{2} + 1 \end{aligned}$$



**Fig. 3** A visualization of  $g_{\alpha,\beta}$  in function of different values of  $|\hat{Y}|$  and  $K$

Note that in the limit, when  $K \rightarrow \infty$ , we obtain the following upper and lower bound for  $\alpha$  and  $\beta$ , respectively:

$$\lim_{K \rightarrow \infty} \frac{K-1}{K} = 1, \quad \lim_{K \rightarrow \infty} \log_{\frac{1}{K-1}} \frac{K}{2} + 1 = 0.$$

## C Experimental setup

For all image datasets, except ALOI.BIN, we use hidden representations obtained by convolutional neural networks, whereas for the text datasets (bottom part of Table 2) tf-idf representations are used. The dimensionality of the representations are denoted by  $D$ . For the MNIST dataset we use a convolutional neural network with three consecutive convolutional, batchnorm and max-pool layers, followed by a fully connected dense layer with 32 hidden units. We use ReLU activation functions and optimize the categorical cross-entropy loss by means of Adam optimization with learning rate  $\eta = 1e - 3$ . For the VOC 2006,<sup>3</sup> VOC 2007,<sup>3</sup> Caltech-101 and Caltech-256, the hidden representations are obtained by resizing images to 224x224 pixels and passing them through the convolutional part of an entire VGG16 architecture, including a max-pooling operation (Simonyan and Zisserman 2014). The weights are set to those obtained by training the network on ImageNet. For all convolutional neural networks, the number of epochs are set to 100 and early stopping is applied with a patience of five iterations. For ALOI.BIN, we use the ALOI<sup>4</sup> dataset with precalculated random binning features (Rahimi and Recht 2008). Training is performed end-to-end on a GPU, by using the PyTorch library (Paszke et al. 2017) and infrastructure with the following specifications:

- **CPU:** i7-6800K 3.4 GHz (3.8 GHz Turbo Boost) – 6 cores / 12 threads.
- **GPU:** 2x Nvidia GTX 1080 Ti 11GB + 1x Nvidia Tesla K40c 11GB.
- **RAM:** 64GB DDR4-2666.

For the bacteria dataset, tf-idf representations are calculated by using 3-, 4-, and 5-grams extracted from each 16S rRNA sequence in the dataset (Fiannaca et al. 2018). For the proteins dataset, we consider 3-grams in order to calculate the tf-idf representation for each protein sequence. To comply with literature, we concatenate the tf-idf representations with functional domain encoding vectors, which provide distinct functional and evolutionary information about the protein sequence. For more information about the functional domain encodings, we refer the reader to (Li et al. 2018). Precalculated tf-idf representations were provided with the DMOZ and LSHTC1 dataset.<sup>5</sup>

Finally, we use the learned hidden representations for the image datasets and calculate tf-idf representations for the text datasets to train the probabilistic models using a dual L2-regularized logistic regression model. For the DMOZ and LSHTC1 dataset we enforce sparsity by clipping all the learned weights less than a threshold  $\eta = 0.1$

<sup>3</sup> The multi-label VOC datasets are transformed to multi-class by removing instances with more than one label.

<sup>4</sup> LIBSVM datasets collection <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

<sup>5</sup> <http://lshtc.iit.demokritos.gr/>.

to zero (Babbar and Schölkopf 2017). We implemented all SVBOP algorithms in C++ using the LIBLINEAR library (Fan et al. 2008) and H-NSW implementation from NMSLIB (Naidan and Boytsov 2015). All experiments were conducted on Intel Xeon E5-2697 v3 2.60GHz (14 cores) with 64GB RAM. We include detail information about selection of hyperparameters for all the models in the next section.

## D Hyperparameters

For the LIBLINEAR library, used for the implementations of all SVBOP algorithms, as well as the baselines, we tuned two parameters:  $C$  – inverse of the regularization strength and  $\epsilon_l$  – tolerance of termination criterion. For SVBOP-HSG and the underlying H-NSW indexing method, we tuned four parameters:  $M$  – the maximum number of neighbors in the layers of H-NSW index,  $ef_c$  – size of the dynamic candidate list during H-NSW index construction, and  $k$  – initial size of the query to H-NSW index and  $ef_s$  – size of the dynamic candidate list during H-NSW index query, that both were always set to the same value. For balanced 2-means tree building, we tuned two parameters:  $l$  – the maximum number of leaves on the last level of a tree and  $\epsilon_c$  – tolerance of termination criterion of the 2-means algorithm. We list all the hyperparameters we used to obtained all the results presented in Sects. 5.2 and 5.3 in Table 5.

**Table 5** Values of hyperparameters used for SVBOP-Full, SVBOP-HSG, and SVBOP-HF algorithms for different datasets

Dataset	Full		HSG			HF			
	$C$	$\epsilon_l$	$M$	$ef_c$	$k/ef_s$	$C$	$\epsilon_l$	$l$	$\epsilon_c$
VOC 2006	100	0.1	–	–	–	–	–	–	–
VOC 2007	100	0.1	–	–	–	–	–	–	–
Caltech-101	100	0.1	–	–	–	–	–	–	–
Caltech-256	100	0.1	–	–	–	–	–	–	–
ALOI.BIN	100	0.1	10	50	10	500	0.1	20	0.001
DBpedia	$10^5$	0.1	–	–	–	–	–	–	–
Bacteria	$10^6$	0.1	50	200	100	$10^6$	0.1	20	0.001
Proteins	$10^6$	0.1	50	200	200	$10^9$	0.1	20	0.001
Dmoz	1000	0.1	20	100	100	50	0.1	100	0.001
LSHTC1	1000	0.1	20	100	100	50	0.1	100	0.001



## References

- Babbar R, Dembczyński K (2018) Extreme classification for information retrieval. Tutorial at ECIR 2018, <http://www.cs.put.poznan.pl/kdembczynski/xmlc-tutorial-ecir-2018/xmlc4ir-2018.pdf>
- Babbar R, Schölkopf B (2017) Dismec: Distributed sparse machines for extreme multi-label classification. Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, DOI 10(1145/3018661):3018741
- Balasubramanian V, Ho S, Vovk V (eds) (2014) Conformal Prediction for Reliable Machine Learning: Theory. Morgan Kaufmann, Adaptations and Applications
- Beygelzimer A, Langford J, Lifshits Y, Sorkin G, Strehl A (2009) Conditional probability tree estimation analysis and algorithms. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, AUAI Press, Arlington, Virginia, United States, UAI '09, pp 51–58
- Bi W, Kwok J (2015) Bayes-optimal hierarchical multilabel classification. IEEE Trans Knowl Data Eng 27:1–1
- Corani G, Zaffalon M (2008) Learning reliable classifiers from small or incomplete data sets: the naive credal classifier 2. J Mach Learn Res 9:581–621
- Corani G, Zaffalon M (2009) Lazy naive credal classifier. In: Proceedings of the 1st ACM SIGKDD Workshop on Knowledge Discovery from Uncertain Data, ACM, pp 30–37
- Del Coz JJ, Díez J, Bahamonde A (2009) Learning nondeterministic classifiers. J Mach Learn Res 10:2273–2293
- Dembczyński K, Waegeman W, Cheng W, Hüllermeier E (2012) An analysis of chaining in multi-label classification. In: Proceedings of the European Conference on Artificial Intelligence
- Dembczyński K, Kotłowski W, Waegeman W, Busa-Fekete R, Hüllermeier E (2016) Consistency of probabilistic classifier trees. In: ECML/PKDD
- Denis C, Hebiri M (2017) Confidence sets with expected sizes for multiclass classification. J Mach Learn Res 18:102–128
- Depeweg S, Hernández-Lobato JM, Doshi-Velez F, Udluft S (2018) Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. ICML, PMLR, Proceedings of Machine Learning Research 80:1192–1201
- Everingham M, Eslami ASM, Gool LV, Williams CKI, Winn J, Zisserman A (2006) The pascal visual object classes challenge 2006 (VOC2006) results. Int J comput vision 111(1):98–136
- Everingham M, Gool LV, Williams CKI, Winn J, Zisserman A (2007) The PASCAL visual object classes challenge 2007 (VOC2007) results
- Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: a library for large linear classification. J Mach Learn Res 9:1871–1874
- Fiannaca A, Paglia LL, Rosa ML, Bosco GL, Renda G, Rizzo R, Gaglio S, Urso A (2018) Deep learning models for bacteria taxonomic classification of metagenomic data. BMC Bioinform 19:61–76
- Fox J (1997) Applied regression analysis, linear models, and related methods. Sage,
- Frank E, Kramer S (2004) Ensembles of nested dichotomies for multi-class problems. In: Proceedings of the Twenty-first International Conference on Machine Learning, ACM, New York, NY, USA, ICML '04, pp 39
- Freitas A (2007) A tutorial on hierarchical classification with applications in bioinformatics. In: Research and Trends in Data Mining Technologies and Applications, pp 175–208
- Geusebroek JM, Burghouts G, Smeulders A (2005) The amsterdam library of object images. Int J Comput Vision 61(1):103–112
- Griffin G, Holub A, Perona P (2007) Caltech-256 object category dataset. Tech Rep 7694, California Institute of Technology
- Hüllermeier E, Waegeman W (2019) Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. [arXiv:1910.09457](https://arxiv.org/abs/1910.09457)
- Jansche M (2007) A maximum expected utility framework for binary sequence labeling. In: Association for Computational Linguistics, pp 736–743
- Johnson J, Douze M, Jégou H (2017) Billion-scale similarity search with gpus. [arXiv:1702.08734](https://arxiv.org/abs/1702.08734)
- Kendall A, Gal Y (2017) What uncertainties do we need in Bayesian deep learning for computer vision? Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017. Long Beach, CA, USA, pp 5580–5590

- LeCun Y, Cortes C (2010) MNIST handwritten digit database. Tech rep Courant Institute, Google Labs, <http://yann.lecun.com/exdb/mnist/>
- Li FF, Andreetto M, Ranzato MA (2003) Caltech101 image dataset. Tech. rep, California Institute of Technology
- Li Y, Wang S, Umarov R, Xie B, Fan M, Li L, Gao X (2018) Deepre: sequence-based enzyme EC number prediction by deep learning. *Bioinformatics* 34(5):760–769
- Malkov YA, Yashunin DA (2018) Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp 1–1
- Melnikov V, Hüllermeier E (2018) On the effectiveness of heuristics for learning nested dichotomies: an empirical analysis. *Mach Learn* 107(8–10):1537–1560
- Mena D, Montañés E, Quevedo JR, del Coz JJ (2017) A family of admissible heuristics for A\* to perform inference in probabilistic classifier chains. *Mach Learn* 106(1):143–169
- Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ (eds) *Advances in Neural Information Processing Systems 26*, Curran Associates, Inc., pp 3111–3119
- Morin F, Bengio Y (2005) Hierarchical probabilistic neural network language model. In: *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, Society for Artificial Intelligence and Statistics, pp 246–252
- Naidan B, Boytsov L (2015) Non-metric space library manual. CoRR [arXiv:1508.05470](https://arxiv.org/abs/1508.05470)
- Navarro G (2002) Searching in metric spaces by spatial approximation. *VLDB J* 11(1):28–46. <https://doi.org/10.1007/s007780200060>
- Nguyen V, Destercke S, Masson M, Hüllermeier E (2018) Reliable multi-class classification based on pairwise epistemic and aleatoric uncertainty. In: *IJCAI*, [ijcai.org](http://ijcai.org), pp 5089–5095
- Ofer D (2019) Dbpedia classes. <https://www.kaggle.com/danofer/dbpedia-classes/metadata>
- Oh S (2017) Top-k hierarchical classification. In: *AAAI*, AAAI Press, pp 2450–2456
- Papadopoulos H (2008) Inductive conformal prediction: theory and application to neural networks. *Tools Artif Intel* 18(2):315–330
- Partalas I, Kosmopoulos A, Baskiotis N, Artières T, Paliouras G, Gaussier É, Androutsopoulos I, Amini M, Gallinari P (2015) LSHTC: A benchmark for large-scale text classification. CoRR [arXiv:1503.08581](https://arxiv.org/abs/1503.08581)
- Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch. In: *NIPS-W*
- Prabhu Y, Varma M (2014) Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In: *KDD*
- Prabhu Y, Kag A, Harsola S, Agrawal R, Varma M (2018) Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In: *Proceedings of the International World Wide Web Conference*
- Rahimi A, Recht B (2008) Random features for large-scale kernel machines. *Adv Neural Inform Process Syst* 20:1177–1184
- Ramaswamy HG, Tewari A, Agarwal S (2015) Consistent algorithms for multiclass classification with a reject option. CoRR [arXiv:1505.04137](https://arxiv.org/abs/1505.04137)
- Rangwala H, Naik A (2017) Large scale hierarchical classification: foundations, algorithms and applications. In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*
- RIKEN (2013) Genomic-based 16s ribosomal rna database. <https://metasystems.riken.jp/grd/download.html>
- Senge R, Bösnér S, Dembczyński K, Haasenritter J, Hirsch O, Donner-Banzhoff N, Hüllermeier E (2014) Reliable classification: Learning classifiers that distinguish aleatoric and epistemic uncertainty. *Inf Sci* 255:16–29
- Shafer G, Vovk V (2008) A tutorial on conformal prediction. *J Mach Learn Res* 9:371–421
- Shrivastava A, Li P (2014) Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, MIT Press, Cambridge, MA, USA, NIPS'14, pp 2321–2329
- Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. *arXiv preprint* [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)

- Stock M, Dembczynski K, Baets BD, Waegeman W (2016) Exact and efficient top-k inference for multi-target prediction by querying separable linear relational models. *Data Min Knowl Discov* 30(5):1370–1394. <https://doi.org/10.1007/s10618-016-0456-z>
- Syed S (2016) Submodularity in machine learning. MLRG Summer School, [https://www.stat.ubc.ca/~saif.syed/papers/mlrg\\_submodularity.pdf](https://www.stat.ubc.ca/~saif.syed/papers/mlrg_submodularity.pdf)
- Vondrak J (2019) Optimization of submodular functions tutorial. <https://theory.stanford.edu/~jvondrak/data/submod-tutorial-1.pdf>
- Vovk V, Gammerman A, Shafer G (2003) *Algorithmic Learning in a Random World*. Springer-Verlag,
- Waegeman W, Dembczyński K, Jachnik A, Cheng W, Hüllermeier E (2014) On the Bayes-optimality of F-measure maximizers. *J Mach Learn Res* 15:3333–3388
- Yagnik J, Strelow D, Ross DA, sung Lin R (2011) The power of comparative reasoning. In: 2011 International Conference on Computer Vision, pp 2431–2438
- Yang G, Destercke S, Masson MH (2017a) Cautious classification with nested dichotomies and imprecise probabilities. *Soft Comput* 21:7447–7462
- Yang G, Destercke S, Masson MH (2017b) The costs of indeterminacy: how to determine them? *IEEE Transact Cybernet* 47:4316–4327
- Ye N, Chai K, Lee WS, Chieu HL (2012) Optimizing f-measures: a tale of two approaches. In: *Proceedings of the International Conference on Machine Learning*
- Zaffalon M, Giorgio C, Mauá DD (2012) Evaluating credal classifiers by utility-discounted predictive accuracy. *Int J Approx Reasoning* 53:1282–1301
- Ziyin L, Wang Z, Liang PP, Salakhutdinov R, Morency LP, Ueda M (2019) Deep gamblers: Learning to abstain with portfolio theory. [arXiv:1907.00208](https://arxiv.org/abs/1907.00208)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.