Deep F-measure Maximization in Multi-label Classification: A Comparative Study

Stijn Decubber[1,2], Thomas Mortier[2], Krzysztof Dembczyński[3], and Willem Waegeman[2]

[1] ML6, Esplanade Oscar Van De Voorde 1, B-9000 Ghent, Belgium
stijn.decubber@ml6.eu
[2] Department of Data Analysis and Mathematical Modelling, Ghent University, Coupure links 653, B-9000 Ghent, Belgium
thomasf.mortier@ugent.be, willem.waegeman@ugent.be
[3] Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
krzysztof.dembczynski@cs.put.poznan.pl

Abstract In recent years several novel algorithms have been developed for maximizing the instance-wise F_β-measure in multi-label classification problems. However, so far, such algorithms have only been tested in tandem with shallow base learners. In the deep learning landscape, usually simple thresholding approaches are implemented, even though it is expected that such approaches are suboptimal. In this article we introduce extensions of utility maximization and decision-theoretic methods that can optimize the F_β-measure with (convolutional) neural networks. We discuss pros and cons of the different methods and we present experimental results on several image classification datasets. The results illustrate that decision-theoretic inference algorithms are worth the investment. While being more difficult to implement compared to thresholding strategies, they lead to a better predictive performance. Overall, a decision-theoretic inference algorithm based on proportional odds models outperforms the other methods.

Keywords: F_β-measure, Bayes optimal predictions, multi-label image classification, convolutional neural networks

1 Introduction

Amongst other utility measures, the F_β-measure is commonly used as a performance metric for multi-label classification (MLC) problems, especially in the case of imbalanced label occurrences. Given a prediction $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \dots, h_m(\boldsymbol{x}))^T$ of an instance $\boldsymbol{x}$ with m-dimensional binary label vector $\boldsymbol{y} = (y_1, \dots, y_m)^T$, where both $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{y}$ belong to $\{0,1\}^m$, the F_β-measure is usually computed in an instance-wise manner:

$$F_\beta(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \frac{(1+\beta^2)\sum_{i=1}^m y_i h_i(\boldsymbol{x})}{\beta^2 \sum_{i=1}^m y_i + \sum_{i=1}^m h_i(\boldsymbol{x})} \quad \in [0,1] \ , \tag{1}$$

where $0/0 = 1$ by definition. Alternative ways of computing the F$_\beta$-measure are macro-averaging, in which the F$_\beta$-measure is not computed per instance, but per label, and micro-averaging, in which the computation is done over the whole instance-label matrix for a predefined dataset. The instance-wise F$_\beta$-measure will be the focus of this work. It is a very relevant measure for many practical MLC problems.

In recent years, specialized algorithms have been developed for optimizing the instance-wise F$_\beta$-measure. Roughly speaking, existing methods can be subdivided into two categories: utility maximization methods and decision-theoretic approaches. Algorithms in the first category intend to minimize a specific loss during the training phase. Many of those algorithms seek for thresholds on scoring functions [1,2,3,4], but also a few more complicated approaches have been proposed [5,6]. For the related problem of binary classification, F$_\beta$-measure maximization at training time can be achieved via extensions of logistic regression [7], boosting [8] or support vector machines [9,10]. However, F$_\beta$-measure maximization is simpler in binary classification than in multi-label classification, because predictions for subsequent instances are independent, while predictions for subsequent labels are not.

Decision-theoretic methods depart from a different perspective. These methods usually fit a probabilistic model $P(\boldsymbol{y} \,|\, \boldsymbol{x})$ to the data during training, followed by an inference procedure at prediction time. This inference procedure consists of optimizing the following optimization problem:

$$\boldsymbol{h}_F(\boldsymbol{x}) = \operatorname*{arg\,max}_{\boldsymbol{h} \in \{0,1\}^m} \mathbb{E}_{\boldsymbol{Y} \,|\, \boldsymbol{x}} \left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}) \right] = \operatorname*{arg\,max}_{\boldsymbol{h} \in \{0,1\}^m} \sum_{\boldsymbol{y} \in \{0,1\}^m} P(\boldsymbol{y} \,|\, \boldsymbol{x}) \, F_\beta(\boldsymbol{y}, \boldsymbol{h}) \,, \quad (2)$$

in which the ground-truth is a vector of random variables $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_m)$, $\mathbb{E}_{\boldsymbol{Y} \,|\, \boldsymbol{x}}$ denotes the expectation for an underlying probability distribution P over $\{0,1\}^m$, and $\boldsymbol{h}$ denotes a potential prediction. This is a non-trivial optimization problem without closed-form solution. Moreover, a brute-force search requires checking all 2^m combinations of $\boldsymbol{h}$ and summing over an exponential number of terms in each combination and is hence infeasible for moderate values of m [11].

For solving (2), one can distinguish approximate inference algorithms, such as those of [12,13,14,15,16,17], and Bayes optimal methods [18,19,20]. Approximate algorithms depart from the assumption of independence of the Y_i, i.e.,

$$P(\boldsymbol{y} \,|\, \boldsymbol{x}) = \prod_{i=1}^{m} (p_i(\boldsymbol{x}))^{y_i} (1 - p_i(\boldsymbol{x}))^{1-y_i} \,, \quad (3)$$

with $p_i(\boldsymbol{x}) = P(y_i = 1 \,|\, \boldsymbol{x})$. In contrast, exact algorithms do not require the independence assumption, which is not realistic for many MLC problems. Optimization problem (2) seems to require information about the entire joint distribution $P(\boldsymbol{y} \,|\, \boldsymbol{x})$. However, exact algorithms have been proposed that solve the problem in an efficient way, by estimating only a quadratic instead of an exponential (with respect to m) number of parameters of the joint distribution.

The main goal of this article is to provide additional insights on how the instance F$_\beta$-measure can be optimized in the context of (convolutional) neural networks. Multi-label classification methods are commonly used in image analysis, for classical tasks such as tagging, segmentation or edge detection. In such studies the F_β-measure is often reported as a performance measure that reflects the practical performance of a classifier in a realistic way. However, the F_β-measure maximization methods that are discussed above have only been tested on simple MLC problems with shallow base learners that do not involve feature learning. Likewise, deep convolutional neural networks, which dominate the image classification landscape, usually only consider crude solutions when optimizing the F_β-measure. Researchers often stick to simple approaches that are easy to implement, while ignoring the shortcomings of those approximations. In a recent Kaggle competition which involved the multi-label classification of satellite images[4], one could observe that almost all top-scoring submissions applied simple thresholding strategies, which are known to be suboptimal. Only one author in the top ten reported improvement gains by testing something different than thresholding strategies. It is therefore interesting to investigate in a more systematic way how the instance-wise F_β-measure can be maximized in the context of deep neural networks.

This article is organized as follows. In Section 2, we will introduce neural network extensions of different algorithms, including several thresholding strategies, and approximate and exact inference methods. Moreover, we introduce a new model based on proportional odds to estimate the set of parameters of the joint label distribution, required to perform exact inference with existing methods. All those methods have pros and cons, which will be discussed without imposing sympathy for one particular method from the beginning. In Section 3, we present the results of a comparative experimental study on four image classification datasets, illustrating the behavior of the methods that we introduce. Our proportional odds model outperforms the alternatives in almost all scenarios. We end with a few clear conclusions.

2 Algorithms for Deep F_β-measure Maximization

In this section we present six different algorithms that can be applied in tandem with (convolutional) neural networks to optimize the F_β-measure. To this end, we make a major distinction between three utility maximization methods and three decision-theoretic methods.

2.1 Utility maximization methods

When optimizing F_β-measure during training with (deep) neural networks, engineers usually consider thresholding strategies on marginal probabilities via a simple line search. Other existing utility maximization methods usually lead to constrained optimization problems, making them not immediately applicable to neural network training. We present three algorithms that seek to optimize the

[4] https://www.kaggle.com/c/planet-understanding-the-amazon-from-space

F_β-measure by means of applying specific thresholds to the predicted marginal probabilities $p_1(\boldsymbol{x}), \ldots, p_m(\boldsymbol{x})$. To this end, we assume that those marginals are modelled with a (convolutional) neural network with one output neuron per label, obtained via a logistic output layer:

$$p_i(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi}))}{1 + \exp(\boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi}))}\,, \tag{4}$$

in which $\boldsymbol{w}_i$ represents parameter vectors, and $\boldsymbol{\phi}$ denotes the map from the input layer to the one-but-last layer, parameterized by a parameter set $\boldsymbol{\psi}$.

This approach is in multi-label classification often referred to as binary relevance (BR). In the results section, the three BR-inspired algorithms will be referred to as threshold averaging ($\mathrm{BR}_\mathrm{t}^\mathrm{avg}$), global thresholding ($\mathrm{BR}_\mathrm{t}^\mathrm{glob}$) and threshold stacking ($\mathrm{BR}_\mathrm{t}^\mathrm{stack}$), respectively.

Threshold averaging ($\mathbf{BR}_\mathbf{t}^\mathbf{avg}$) The first thresholding approach consists of computing a specific optimal threshold $\theta_*^{(i)}$ for each instance $\boldsymbol{x}^{(i)}$ during training time. The algorithm passes over the data exactly once and considers the marginal probabilities $p_1(\boldsymbol{x}^{(i)}), \ldots, p_m(\boldsymbol{x}^{(i)})$ in decreasing order as candidate thresholds. At test time, the average optimal threshold over the training dataset is applied as a common threshold. Algorithm 1 provides pseudocode for a single instance; the algorithm can be applied on an entire training dataset with $\mathcal{O}(mn)$ time complexity, by vectorizing the counter variables.

Algorithm 1 Threshold Averaging

```
Input: a training instance $(\boldsymbol{x}, \boldsymbol{y})$, predictions $\boldsymbol{p}(\boldsymbol{x}) = (p_1(\boldsymbol{x}), \ldots, p_m(\boldsymbol{x}))$
compute $s^{\boldsymbol{y}} = \sum_{i=1}^m y_i$
$\boldsymbol{p} \leftarrow \mathbf{sort}(\boldsymbol{p}(\boldsymbol{x}))$ s.t. $p_1 \geq p_2 \geq \ldots \geq p_m$, store the sorting indices in a list $L$
set $s^{\boldsymbol{h}} = 0,\ s^{\boldsymbol{yh}} = 0,\ F_{max} = 0,\ \theta_* = p_1$
for $k = 1$ to $m - 1$ do
    $s^{\boldsymbol{yh}} \leftarrow s^{\boldsymbol{yh}} + y_{L[k]}$
    compute $F = (1 + \beta^2)\frac{s^{\boldsymbol{yh}}}{\beta^2 s^{\boldsymbol{y}} + k}$
    if $F > F_{max}$ then
        $F_{max} \leftarrow F$, $\theta_* \leftarrow p_{k+1}$
    end if
end for
return optimal threshold $\theta_*$
```

Global thresholding ($\mathbf{BR}_\mathbf{t}^\mathbf{glob}$) Algorithm 2 directly finds a single global optimal threshold θ_* at training time. The method acts on the entire training data set by concatenating all marginal probabilities $p_1(\boldsymbol{x}), \ldots, p_m(\boldsymbol{x})$ for different $\boldsymbol{x}$, and considering each value as candidate threshold. This second thresholding method seeks to improve over the previous method by considering much more candidate thresholds. However, this comes at the expense of an increasing time complexity. Sorting the vector of all marginals takes $\mathcal{O}(mn\log(mn))$ time, and the computation of the optimal threshold takes $\mathcal{O}(m^2n)$. Each of those two factors might be dominating, depending on m and n. Let us remark that Algorithm 2 could be substantially simplified if the macro or micro F_β-measure would be optimized instead of the instance-wise F_β-measure. For the instance-wise F_β-measure one needs to keep track of the score for every instance individually for different thresholds, resulting in a higher time complexity compared to the micro and macro F_β-measures. Algorithmically, too, threshold-based optimization of the latter two measures is easier.

Algorithm 2 Global Thresholding

```
Input: tr. data $\{(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}),\ldots,(\boldsymbol{x}^{(n)},\boldsymbol{y}^{(n)})\}$, predictions $\{\boldsymbol{p}(\boldsymbol{x}^{(1)}),\ldots,\boldsymbol{p}(\boldsymbol{x}^{(n)}))\}$
$\boldsymbol{q} \leftarrow$ concatenate$(\{\boldsymbol{p}(\boldsymbol{x}^{(1)}),\ldots,\boldsymbol{p}(\boldsymbol{x}^{(n)})\})$
$\boldsymbol{r} \leftarrow$ concatenate$(\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(n)}\})$
$\boldsymbol{q} \leftarrow$ sort$(\boldsymbol{q})$ s.t. $q_1 \geq q_2 \geq \ldots \geq q_{n\times m}$
compute $\boldsymbol{s}^{\boldsymbol{y}} : s^{\boldsymbol{y}}_j = \sum_{i=1}^m y_i^{(j)} \ \forall j \in \{1,\ldots,n\}$
set $\boldsymbol{s}^{\boldsymbol{h}} = \mathbf{0}_n,\ \boldsymbol{s}^{\boldsymbol{yh}} = \mathbf{0}_n,\ F_{max} = 0,\ \theta_* = q_1$
for $k = 1$ to $(n \times m) - 1$ do
    $j \leftarrow$ index of the training instance that corresponds to $q_k$
    $s^{\boldsymbol{h}}_j \leftarrow s^{\boldsymbol{h}}_j + 1,\ s^{\boldsymbol{yh}}_j \leftarrow s^{\boldsymbol{yh}}_j + r_j$
    compute $F = \frac{1}{n}\sum_{i=1}^n (1+\beta^2)\frac{s_i^{\boldsymbol{yh}}}{\beta^2 s_i^{\boldsymbol{y}} + s_i^{\boldsymbol{h}}}$
    if $F > F_{max}$ then
        $F_{max} \leftarrow F,\ \theta_* \leftarrow q_{k+1}$
    end if
end for
return global optimal threshold $\theta_*$
```

Threshold stacking ($\mathbf{BR}_{\mathbf{t}}^{\mathbf{stack}}$) The final thresholding method presented here tries to predict the instance-wise optimal thresholds for each test instance, in an approach similar to stacking, see e.g. [21], [22]. A set of marginal probabilities and optimal thresholds $\{(p(\boldsymbol{x}^{(1)}),\theta_*^{(1)}),\ldots,(p(\boldsymbol{x}^{(n)}),\theta_*^{(n)})\}$ is obtained via Algorithm 1 and serves as training data to learn a mapping from probability vectors to thresholds. As such, one ends up with a stacked model structure:

$$\boldsymbol{x} \mapsto p_1(\boldsymbol{x}),\ldots,p_m(\boldsymbol{x}) \mapsto \theta_*(\boldsymbol{x})\,.$$

The first mapping consists of a (convolutional) neural network that predicts marginal probabilities, and the second mapping will be a ridge regression model that transforms the distribution over marginals to a distribution-specific threshold. The distribution of marginal probabilities depends on $\boldsymbol{x}$, so one can argue that the predicted threshold is instance-specific.

2.2 Decision-theoretic methods

We present in total three algorithms that optimize the F$_\beta$-measure in a decision-theoretic perspective using so-called plug-in classifiers, i.e. classifiers that fit a probabilistic model at training time, followed by an inference phase at test time. We first mention an approach that departs from marginal probabilities and optimizes (2) in an approximate way by assuming label independence. This approach will be referred to as the label independence F$_\beta$ plug-in classifier (LFP). Subsequently, we introduce two methods that do not have this restriction and provide exact solutions for (2). These methods do not require the plugin of m estimated marginal probabilities but rather a set of $m^2 + 1$ parameters of the joint distribution. To this end, we propose a neural network architecture with an output layer that is modified compared to the models that are typically used for BR estimation of marginal probabilities. The two exact methods differ in the hypothesis class that is considered.

All the methods in this section rely on solving (2) via outer and inner maximization. Let H_k denote the space of all possible predictions that contain exactly k positive labels: $H_k = \{\boldsymbol{h} \in \{0,1\}^m \mid \sum_{i=1}^m h_i = k\}$. The inner maximization then solves

$$\boldsymbol{h}_k(\boldsymbol{x}) = \operatorname*{arg\,max}_{\boldsymbol{h} \in H_k} \mathbb{E}_{\boldsymbol{Y} \mid \boldsymbol{x}} \left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}) \right], \tag{5}$$

for each k. Subsequently, the outer maximization seeks to find the F$_\beta$-maximizer $\boldsymbol{h}_F$:

$$\boldsymbol{h}_F(\boldsymbol{x}) = \operatorname*{arg\,max}_{\boldsymbol{h} \in \{\boldsymbol{h}_0(\boldsymbol{x}), \ldots, \boldsymbol{h}_m(\boldsymbol{x})\}} \mathbb{E}_{\boldsymbol{Y} \mid \boldsymbol{x}} \left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}) \right]. \tag{6}$$

The solution to (6) is found by checking all $m + 1$ possibilities. The algorithms discussed below differ in the way they solve the inner maximization (5).

Label independence F$_{\boldsymbol{\beta}}$ plug-in classifier (LFP) By assuming independence of the random variables $Y_1, \ldots, Y_m$, optimization problem (5) can be substantially simplified. It has been shown independently in [12] and [14] that the optimal solution then always contains the labels with the highest marginal probabilities, or no labels at all.

Theorem 1 [12] *Let $Y_1, Y_2, \ldots, Y_m$ be independent Bernoulli variables with parameters $p_1, p_2, \ldots, p_m$ respectively. Then, for all $j, k \in \{1, \ldots, m\}$, $h_{F,j} = 1$ and $h_{F,k} = 0$ implies $p_j \geq p_k$.*

As a consequence, only a few hypotheses $\boldsymbol{h}$ (m+1 instead of 2^m) need to be examined, and the computation of the expected F$_\beta$-measure can be performed in an efficient way. [13], [14], [15] and [16] have proposed exact procedures for computing the F$_\beta$-maximizer under the assumption of label independence. All those methods take as input predicted marginal probabilities $(p_1, p_2, \ldots, p_m)$ with shorthand notation $p_i = p_i(\boldsymbol{x})$, and they all obtain the same solution. In what follows we only discuss the method of [16], which is the most efficient among the four implementations. This method only works for rational β^2; in other cases a less efficient algorithm can be used.

As a starting point, let us assume that the labels are sorted according to the marginal probabilities and let $\boldsymbol{h}_k(\boldsymbol{x})$ be the prediction that returns a one for the labels with the k highest marginal probabilities and zero for the other labels. Furthermore, let $s^{\boldsymbol{y}}_{i:j} = \sum_{l=i}^{j} y_l$, then one can observe that

$$\begin{aligned} \mathbb{E}\left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}_k(\boldsymbol{x}))\right] &= \sum_{\boldsymbol{y}\in\{0,1\}^m} F_\beta(\boldsymbol{y}, \boldsymbol{h}_k(\boldsymbol{x}))P(\boldsymbol{y} \mid \boldsymbol{x}) \qquad (7) \\ &= \sum_{\substack{0\le k_1\le k \\ 0\le k_2\le m-k}} \frac{P(s^{\boldsymbol{y}}_{1:k} = k_1)P(s^{\boldsymbol{y}}_{k+1:m} = k_2)(1+\beta^2)k_1}{k + \beta^2(k_1+k_2)} \\ &= \sum_{k_1=0}^{k} (1+\beta^{-2})k_1 P(s^{\boldsymbol{y}}_{1:k} = k_1) s(k, k\beta^{-2} + k_1)\,, \end{aligned}$$

where $s(k,\alpha) = \sum_{k_2=0}^{m-k} P(s^{\boldsymbol{y}}_{k+1:m} = k_2)/(\alpha + k_2)$. Now observe that

$$P(s^{\boldsymbol{y}}_{k:m} = i) = p_k P(s^{\boldsymbol{y}}_{k+1:m} = i-1) + (1-p_k)P(s^{\boldsymbol{y}}_{k+1:m} = i)\,.$$

As a result, the s-values for different values of k in (7) can be computed recursively:

$$\begin{aligned} s(k-1,\alpha) &= \sum_{k_2=0}^{m-k+1} \frac{P(s^{\boldsymbol{y}}_{k:m} = k_2)}{\alpha + k_2} \\ &= p_k \sum_{k_2=0}^{m-k+1} \frac{P(s^{\boldsymbol{y}}_{k+1:m} = k_2 - 1)}{\alpha + k_2} + (1-p_k) \sum_{k_2=0}^{m-k+1} \frac{P(s^{\boldsymbol{y}}_{k+1:m} = k_2)}{\alpha + k_2} \\ &= p_k \sum_{k_2=0}^{m-k} \frac{P(s^{\boldsymbol{y}}_{k+1:m} = k_2)}{\alpha + k_2 + 1} + (1-p_k) \sum_{k_2=0}^{m-k} \frac{P(s^{\boldsymbol{y}}_{k+1:m} = k_2)}{\alpha + k_2} \\ &= p_k s(k, \alpha+1) + (1-p_k)s(k,\alpha)\,, \end{aligned}$$

with $s(m,\alpha) = 1/\alpha$ and $s(k,\alpha) = 0$ when $k < 0$ or $k > m$. Remark that the transition from the second to the third line follows from an index change.

The recursive formula suggests a dynamic programming implementation with k ranging from $k = m$ to $k = 1$, as given in Algorithm 3. Here we first introduce a list of lists, using double indexing, such that $L[k][j] = P(s^{\boldsymbol{y}}_{1:k} = j)$ with $j \in \{-1, 0, \ldots, k+1\}$. This data structure can also be initialized via dynamic programming:

$$\begin{aligned} L[k][j] &= p_k P(s^{\boldsymbol{y}}_{1:k-1} = j-1) + (1-p_k)P(s^{\boldsymbol{y}}_{1:k-1} = j) \\ &= p_k L[k-1][j-1] + (1-p_k)L[k-1][j] \end{aligned}$$

using $L[1] = [0, (1-p_1), p_1, 0]$ and $L[k][-1] = L[k][k+1] = 0$. After initializing those lists, one can proceed with computing $s(k,\alpha)$ for rational β^2. To this end, we introduce $S[i] = s(k, i/q)$ with $\beta^2 = q/r$, which leads to the implementation given in Algorithm 3. Further speed-ups can be obtained via Taylor series approximations, which might be useful when m becomes very large.

Algorithm 3 LFP – Ye et al. (2012)

```
Input: predictions $\boldsymbol{p}(\boldsymbol{x}) = (p_1(\boldsymbol{x}), \ldots, p_m(\boldsymbol{x}))$, $\beta^2 = q/r$
$\boldsymbol{p} \leftarrow$ sort($\boldsymbol{p}$) s.t. $p_1 \geq \ldots \geq p_m$
initialize $L \leftarrow$ a list of $m$ empty lists
set $L[1] = [0, (1 - p_1), p_1, 0]$
for $k = 2$ to $m$ do
    $L[k] \leftarrow$ a list of $k + 3$ zeros with index starting at $-1$
    for $j = 0$ to $k$ do
        $L[k][j] \leftarrow p_k \times L[k-1][j-1] + (1 - p_k) \times L[k-1][j]$
    end for
end for
For $1 \leq i \leq (q + r)m$: $S[i] \leftarrow q/i$
for $k = m$ to 1 do
    $\mathbb{E}\left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}_k(\boldsymbol{x}))\right] \leftarrow \sum_{k_1=0}^{k} (1 + r/q) k_1 L[k][k_1] S[rk + qk_1]$
    for $i = 1$ to $(q + r)(k - 1)$ do
        $S[i] \leftarrow (1 - p_k) S[i] + p_k S[i + q]$
    end for
end for
$k \leftarrow \arg\max_k \mathbb{E}\left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}^*_k(\boldsymbol{x}))\right]$
return $\boldsymbol{h}_F(\boldsymbol{x})$ by setting $h_i = 1$ for the $k$ labels with the highest $p_i$
```

General F$_{\boldsymbol{\beta}}$ maximizer (GFM) The algorithm that was explained in the previous section assumed that the labels were independent, so that only marginals had to be modelled in order to solve inner problem (5). In what follows we discuss two different extensions of an alternative algorithm that does not assume label independence [19]. The algorithm is Bayes optimal for any probability distribution, but the price one has to pay for this is that more parameters of $P(\boldsymbol{y} \,|\, \boldsymbol{x})$ must be estimated. As a starting point, we introduce the following shorthand notations:

$$s^{\boldsymbol{y}} = s^{\boldsymbol{y}}_{1:m} \quad \text{and} \quad \Delta_{ik} = \sum_{\boldsymbol{y}: y_i = 1} \frac{P(\boldsymbol{y} \,|\, \boldsymbol{x})}{\beta^2 s^{\boldsymbol{y}} + k} \,.$$

By plugging (1) into (5), one can write

$$\boldsymbol{h}_k = \operatorname*{arg\,max}_{\boldsymbol{h} \in H_k} \sum_{\boldsymbol{y} \in \{0,1\}^m} \frac{(1 + \beta^2) \sum_{i=1}^{m} y_i h_i P(\boldsymbol{y} \,|\, \boldsymbol{x})}{\beta^2 s^{\boldsymbol{y}} + k} \,. \tag{8}$$

Swapping the sums in (8) leads to

$$\begin{aligned}\boldsymbol{h}_k &= \operatorname*{arg\,max}_{\boldsymbol{h}\in H_k}(1+\beta^2)\sum_{i=1}^{m} h_i \sum_{\boldsymbol{y}\in\{0,1\}^m} \frac{y_i P(\boldsymbol{y}\,|\,\boldsymbol{x})}{\beta^2 s^{\boldsymbol{y}} + k} \\ &= \operatorname*{arg\,max}_{\boldsymbol{h}\in H_k}(1+\beta^2)\sum_{i=1}^{m} h_i \Delta_{ik}\,. \end{aligned} \tag{9}$$

The inner maximization is solved by setting $h_i = 1$ for the top k values of Δ_{ik}. For each $\boldsymbol{h}_k$, $\mathbb{E}\left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}_k)\right]$ is stored and used to solve the outer maximization. For the specific case of $\boldsymbol{h}_0$, $\mathbb{E}\left[F_\beta(\boldsymbol{Y}, \boldsymbol{h}_0)\right]$ equals $P(\boldsymbol{y} = \boldsymbol{0}\,|\,\boldsymbol{x})$, which needs to be estimated separately. Algorithm 4 provides pseudocode for the complete procedure. This algorithm requires Δ_{ik} for $1 \leq i, k \leq m$ and $P(\boldsymbol{y} = \boldsymbol{0}\,|\,\boldsymbol{x})$, that is, $m^2 + 1$ parameters to obtain $\boldsymbol{h}_F$. With these parameters, the solution can be obtained in $\mathcal{O}(m^2)$ time, i.e., the dominating part of the procedure is the inner maximization: for each k, a selection of the top k elements must be done, which can be accomplished in linear time. Thus, compared to the approach that assumed label independence, more parameters need to be estimated. The advantage of not imposing any distributional assumptions brings a more difficult estimation problem as disadvantage. Depending on the distributional properties of a specific dataset, it can therefore be the case that one algorithm outperforms the other, or the other way around.

Algorithm 4 General F$_\beta$ maximizer (GFM)

```
Input: matrix $\Delta$ with elements $\Delta_{ik}$ and $P(\boldsymbol{y} = \boldsymbol{0})$
set $\mathbb{E}[F_\beta(\boldsymbol{Y}, \boldsymbol{h}_0)] = P(\boldsymbol{y} = \boldsymbol{0})$
for $k = 1$ to $m$ do
    solve inner maximization: $\boldsymbol{h}_k(\boldsymbol{x}) = \arg\max_{\boldsymbol{h}\in H_k} \mathbb{E}[F_\beta(\boldsymbol{Y}, \boldsymbol{h})]$
    by setting $h_i = 1$ for the $k$ labels with the highest $\Delta_{ik}$
    set $\mathbb{E}[F_\beta(\boldsymbol{Y}, \boldsymbol{h}_k)] \leftarrow (1+\beta^2)\sum_{i=1}^{m} h_i \Delta_{ik}$
end for
for $k = 0$ to $m$ do
    solve outer maximization: $\boldsymbol{h}_F = \arg\max_{\boldsymbol{h}\in\{\boldsymbol{h}_0(\boldsymbol{x}),\ldots,\boldsymbol{h}_m(\boldsymbol{x})\}} \mathbb{E}[F_\beta(\boldsymbol{Y}, \boldsymbol{h})]$
end for
return $\boldsymbol{h}_F(\boldsymbol{x})$
```

Estimating $\boldsymbol{\Delta}$ with multinomial regression (GFM$_{\mathbf{MR}}$) [19] proposed the following scheme to estimate the probabilities Δ_{ik}. Let $\boldsymbol{P}$ and $\boldsymbol{W}$ denote two $m \times m$ matrices with elements

$$p_{is} = P(y_i = 1, s^{\boldsymbol{y}} = s\,|\,\boldsymbol{x}), \quad w_{rk} = (\beta^2 r + k)^{-1}\,,$$

respectively. Then, the $m \times m$ matrix $\boldsymbol{\Delta}$ with elements Δ_{ik} can be obtained by

$$\boldsymbol{\Delta} = \boldsymbol{P}\boldsymbol{W}\,.$$

When using simple base learners, one can proceed to estimate $\boldsymbol{P}$ by reducing the problem to m independent problems, each with up to $m+1$ classes. Each subproblem i involves the estimation of

$$P(y = [\![y_i = 1]\!] \cdot s^{\boldsymbol{y}} \,|\, \boldsymbol{x}), \quad \forall\, y \in \{0, \dots, m\}, \tag{10}$$

which sum to one. The subproblems can hence be solved with multinomial regression. For $y = \{1, \dots, m\}$, these probabilities make up the elements of the rows of $\boldsymbol{P}$. Similarly as for the deep neural network that estimated marginal probabilities, we model the i-th row of $\boldsymbol{P}$ via a softmax layer:

$$p_{is}(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_{is}^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi}))}{\sum_{s=0}^{m} \exp(\boldsymbol{w}_{is}^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi}))} \,,$$

with $i = 1, \dots, m$, $\boldsymbol{w}_{is}$ parameter vectors, and $\boldsymbol{\phi}$ the map that originates from the feature learning phase, again parameterized by parameter set $\boldsymbol{\psi}$.

It should be noted that $s^{\boldsymbol{y}}$ equals m only in the worst case where an instance is attributed with all possible labels. This is rarely encountered in practice. Let $s_m = \max_{1 \leq j \leq n} \sum_{i=1}^{m} y_i^{(j)}$, then the total number of output classes for each subproblem (10) can be reduced to $s_m + 1$. Nevertheless, fitting each of multinomial regression problems independently is undesirable when the cost of training the base learners becomes higher, as with deep (convolutional) neural networks, especially for large m. We propose the natural solution of estimating $\boldsymbol{P}$ in its entirety as the output of a single neural network. The two-dimensional final layer of the network should contain m rows of $(s_m + 1)$ output neurons, where a row-wise soft-max transformation is applied. Then the loss to be minimized during training is composed of m cross-entropy losses, which can be minimized using stochastic gradient descent. The m^2 entries required for $\boldsymbol{P}$ can be obtained from the output of the network by discarding the first column and by adding $m - s_m$ columns with zeros.

Estimating $\boldsymbol{\Delta}$ with ordinal regression (GFM$_{\mathbf{OR}}$) Additionally, we propose to reformulate the problem of estimating the elements of Δ as an ordinal regression problem. The key insight is to factorize the probabilities p_{is} as follows:

$$p_{is} = P(y_i = 1, s^{\boldsymbol{y}} = s \,|\, \boldsymbol{x}) = P(s^{\boldsymbol{y}} = s \,|\, y_i = 1, \boldsymbol{x})\, P(y_i = 1 \,|\, \boldsymbol{x}).$$

As before, $P(y_i = 1 \,|\, \boldsymbol{x})$ can be estimated by means of BR. In the conditional probability $P(s^{\boldsymbol{y}} = s \,|\, y_i = 1, \boldsymbol{x})$, $s^{\boldsymbol{y}}$ can take on values from 1 to s_m. By exploiting the ordinal nature of the variable $s^{\boldsymbol{y}}$, one can estimate the conditional probability with proportional odds models, while reducing the number of parameters, compared to GFM$_{\mathrm{MR}}$ [23]. After estimating these conditional probabilities, they can be multiplied with the marginals to obtain the probabilities p_{is} required for GFM.

Taking into account the conditioning on $y_i = 1$, one can choose to estimate m independent proportional odds models. However, we will consider a *global* proportional odds model, consisting of m proportional odds submodels which are optimized jointly in a multi-task learning way. As such, the i-th submodel is characterized by s_m classes, a parameter vector $\boldsymbol{w}_i$ and a vector of bias terms $\boldsymbol{b}^{(i)} = (b_0^{(i)}, b_2^{(i)}, \ldots, b_{s_m}^{(i)})$, subject to

$$b_0^{(i)} < b_1^{(i)} < \cdots < b_{s_m}^{(i)} \,, \tag{11}$$

with $b_0^{(i)} = -\infty$ and $b_{s_m}^{(i)} = \infty$.

Formally speaking, the i-th proportional odds model will estimate the cumulative probabilities

$$P(s^{\boldsymbol{y}} \leq s \,|\, y_i = 1, \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi}) - b_s^{(i)})}{1 + \exp(\boldsymbol{w}_i^T \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi}) - b_s^{(i)})}, \text{ for } i \in \{1, \ldots, m\} \,, \tag{12}$$

where we depart from some learnable feature representation $\boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\psi})$, as in the other methods. Consequently, the conditional distribution of $s^{\boldsymbol{y}}$ can then be retrieved as follows:

$$P(s^{\boldsymbol{y}} = s \,|\, y_i = 1, \boldsymbol{x}) = P(s^{\boldsymbol{y}} \leq s \,|\, y_i = 1, \boldsymbol{x}) - P(s^{\boldsymbol{y}} \leq s - 1 \,|\, y_i = 1, \boldsymbol{x}) \,.$$

Furthermore, we estimate the model parameters in (12) jointly for $i \in \{1, \ldots, m\}$, by minimizing the following log-likelihood function:

$$\operatorname*{arg\,min}_{\boldsymbol{W}, \boldsymbol{B}, \boldsymbol{\psi}} \left(-\sum_n \sum_{i=1}^{m} \sum_{s=1}^{s_m} I_{nis} \, T\Big(P(s^{\boldsymbol{y}} = s \,|\, y_i = 1, \boldsymbol{x}) \Big) \right), \tag{13}$$

with $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m)$, $\boldsymbol{B} = (\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(m)})$ and I_{nis} a binary indicator, which is one when the n-th training instance $(\boldsymbol{x}, \boldsymbol{y})$ has $y_i = 1$ and $s^{\boldsymbol{y}} = s$. T is a transformation function

$$T(z; \epsilon) = \begin{cases} \log \epsilon & \text{if } z \leq 0 \\ \log z & \text{if } z > 0 \end{cases} ,$$

for $\epsilon > 0$, that defines a truncated log-likelihood.
This transformation can be seen as the modified negative log-likelihood of the proportional odds model. It is needed to guarantee numerical stability of the optimization algorithm, in case $P(s^{\boldsymbol{y}} = s \,|\, y_i = 1, \boldsymbol{x})$ becomes negative. This might happen in the early optimization steps, as (11) is not necessarily obeyed. Moreover, when the ordering constraint on the thresholds is not fulfilled, this will be directly penalized by the truncated log-likelihood, provided that ϵ is chosen sufficiently small, e.g. $\epsilon = 1e^{-10}$. The truncated log-likelihood will hence yield a similar effect as logarithmic barrier penalty terms, which are sometimes used to enforce monotonicity as in (11).

Although GFM_{OR} needs less parameters to estimate p_{is} than GFM_{MR}, it requires m values for the marginals as additional input. In case a separate model is used to estimate the marginals (starting from the same feature representation of size d), the parameter requirements for BR + GFM_{OR} are $dm + m + dm + m(s_m - 1)$, which boils down to $m \times (2d + s_m)$. This number will still be lower than the number of parameters required for GFM_{MR}, which can be rewritten as $m \times ((s_m + 1)d + s_m + 1)$.

3 Empirical analysis

3.1 Experimental setup

We compare the discussed methods by means of empirical evaluation on real-world datasets. Estimates of the marginal probability vectors are made by means of BR in the form of a convolutional neural network with m output nodes subject to a sigmoid non-linearity in the output layer, as given in Eqn. 4. Our results include the F_β-measure scores obtained with BR, without any form of F_β-measure maximization. The marginal probabilities for the training data obtained by BR serve as input for the thresholding methods $\mathrm{BR}_\mathrm{t}^\mathrm{avg}$, $\mathrm{BR}_\mathrm{t}^\mathrm{glob}$ and $\mathrm{BR}_\mathrm{t}^\mathrm{stack}$. GFM_MR and GFM_OR estimate the m^2+1 parameters of $P(\boldsymbol{y} \mid \boldsymbol{x})$, required for GFM, with multinomial regression and proportional odds, respectively. The GFM algorithm is then used in tandem with these methods to obtain optimal predictions. Finally, the LFP method starts from the marginal probabilities obtained with BR for the test data.

We report both the F_1 and F_2-measure scores obtained on four publicly available multi-label classification image datasets: PASCAL VOC 2007 [24], PASCAL VOC 2012 [25], Microsoft COCO [26] and the Kaggle Planet dataset [27]. We use the recommended *train-val-test* split for VOC 2007 and perform custom training/validation splits for the other datasets. Table 1 provides some summarizing statistics. All experiments were carried out on a single NVIDIA GTX 1080Ti GPU. All algorithms were implemented in Python using TensorFlow [28], Keras [29] and Pytorch [30].

When it comes to the experiments, for each dataset, the features are vectors of size 512 obtained by resizing the images to 224x224 pixels and passing them through the convolutional part of an entire VGG16 architecture, including a max-pooling operation [31]. The final fully connected classification layers from the original architecture are replaced by a single fully connected layer with 128 neurons (ReLu activation), followed by either a single-layer BR, GFM_MR or GFM_OR classifier, as described in Section 2. The weights and biases of this architecture were set to those obtained by training the network on ImageNet; these are publicly available and accessible through the Keras API. First, the convolutional layers are fixed and the fully connected classification layers are trained until convergence. Then, the entire network is fine-tuned with a lower learning rate. In both stages, early stopping is applied, similarly as before.
Moreover, the BR estimator consists of a single-layer neural network with m output nodes. Likewise, the GFM_BR and GFM_OR models consist of single-layer neural networks parameterized as described in the previous section. A small amount of dropout regularization was applied (dropout probability 0.2) at the input level. All models were trained with the Adam optimization algorithm (learning rate $1e^{-3}$), where early stopping was applied with a five epochs patience counter.

Table 1. Summary statistics for the four datasets. m is the number of labels, s_m the maximum number of labels attributed to a single instance in the training data.

	n_{train}	n_{val}	n_{test}	m	s_m
PLANET	32383	8096	61191	17	9
VOC 2007	2501	2510	4952	20	7
VOC 2012	4859	858	5823	20	6
COCO	65665	16416	40137	80	18

3.2 Experimental results

The results for the conducted experiments are presented in Table 2. As expected, BR without any attempt at maximizing the F_β-measure leads to the worst performance in almost all cases. Rather surprising is the fact that $\text{BR}_\text{t}^\text{avg}$ performs worse than BR for the F_1-measure in several cases, meaning that the average optimal threshold for the training instances is not better than just 0.5 as a threshold. This is especially true for the Planet dataset, which has the smallest m and an imbalanced label distribution. Conversely, this does not occur for the COCO dataset, where, due to a larger numer of labels, more candidate thresholds are considered for each instance by Algorithm 1. $\text{BR}_\text{t}^\text{glob}$ consistently outperforms both BR and $\text{BR}_\text{t}^\text{avg}$. This is as expected, since $\text{BR}_\text{t}^\text{glob}$ considers all $m \times n$ predicted marginal probabilities as candidates. However, this comes at the cost of higher time complexity, as discussed in Section 2.

The performance of $\text{BR}_\text{t}^\text{stack}$ varies across datasets and seems to depend on whether F_1 or F_2 is the measure of interest. For F_1 it performs substantially worse than $\text{BR}_\text{t}^\text{glob}$, whereas it even becomes competitive with the decision-theoretic approaches for F_2. Figure 1 gives further insights w.r.t. the behavior of $\text{BR}_\text{t}^\text{stack}$. It shows for training data the empirical distribution of instance-wise thresholds obtained by Algorithm 1, as well as the thresholds predicted by $\text{BR}_\text{t}^\text{stack}$. One can observe that for all four datasets the two distributions differ substantially, indicating that the threshold stacking method is not always capable of predicting a good threshold. The empirical distribution of instance-wise thresholds obtained by Algorithm 1 is here considered as the ground truth. The dotted line indicates the threshold that will be returned by Algorithm 1 after training.

More generally, the decision-theoretic approaches seem to outperform the thresholding methods on all datasets. The GFM algorithm, which is the only algorithm that does not require the assumption of label independence, is the best algorithm in all but one setting. In almost all cases the proportional odds model outperforms the multinomial regression model, which might indicate that the assumption of ordinality for $s^{\boldsymbol{y}}$ is a valid assumption. However, the differences between both methods are small, so the benefit of a more parsimonious model structure is limited. In addition, the LFP method also yields rather good results. Therefore, we hypothesize that for the analyzed datasets the dependence among

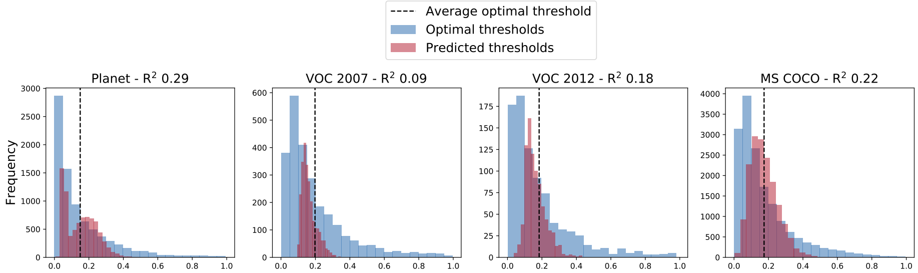

Figure 1. Empirical distributions of instance-wise thresholds obtained by Algorithm 1 (blue, optimal thresholds), as well as the thresholds predicted by $\mathrm{BR}_{\mathrm{t}}^{\mathrm{stack}}$ (red, predicted thresholds). Here, the thresholds for training data are shown, and F_2 is the performance measure. The thresholds are obtained by using the convolutional part of a high-quality pre-trained VGG16 architecture (R^2 indicates the quality of the predictions). The dotted line indicates the mean optimal instance-wise threshold, which is returned by $\mathrm{BR}_{\mathrm{t}}^{\mathrm{avg}}$ after training. See main text for more details.

Table 2. Comparison of the different methods, with training strategy described in Section 3.1.

	Planet		VOC 2007		VOC 2012		COCO	
	F_1	F_2	F_1	F_2	F_1	F_2	F_1	F_2
BR	0.8997	0.8918	0.7398	0.7282	0.7539	0.7405	0.6534	0.6179
$\mathrm{BR}_{\mathrm{t}}^{\mathrm{avg}}$	0.8787	0.9135	0.7241	0.8055	0.7286	0.8064	0.6852	0.7170
$\mathrm{BR}_{\mathrm{t}}^{\mathrm{glob}}$	0.9017	0.9149	0.7701	0.7973	0.7769	0.8020	0.6846	0.7163
$\mathrm{BR}_{\mathrm{t}}^{\mathrm{stack}}$	0.8772	0.9091	0.7121	0.8011	0.7142	0.8007	0.6752	0.7174
$\mathrm{GFM}_{\mathrm{MR}}$	**0.9044**	0.9164	0.7918	0.8108	0.7988	0.8154	0.6955	0.7230
$\mathrm{GFM}_{\mathrm{OR}}$	0.9026	**0.9172**	0.8005	**0.8211**	**0.8035**	**0.8205**	**0.7040**	**0.7316**
LFP	0.9023	0.9107	**0.8007**	0.8177	0.8032	0.8184	0.7011	0.7177

the labels is not very strong. Moreover, even though LFP assumes independence, it requires less parameters than the GFM methods.

4 Conclusion

In this article we introduced extensions of utility maximization and decision-theoretic methods that can optimize the F_β-measure with (convolutional) neural networks. We discussed pros and cons of the different methods and we presented experimental results on several image classification datasets. The results illustrate that decision-theoretic inference algorithms are worth the investment. While being more difficult to implement compared to thresholding strategies, they lead to a superior predictive performance. This is a surprising result, given the popularity of thresholding in deep neural networks. For most of the datasets, the inferior performance of thresholding strategies was remarkable, while also big differences could be observed among the different ways of defining a threshold. Overall, the best performance was obtained with an exact decision-theoretic method based on proportional odds models. This is interesting, because this method is at the same time the most novel among the different methods that were analyzed in this paper.

References

1. Keerthi, S., Sindhwani, V., Chapelle, O.: An efficient method for gradient-based adaptation of hyperparameters in SVM models. In: Advances in Neural Information Processing Systems. Volume 19., MIT Press (2007)
2. Fan, R., Lin, C.: A study on threshold selection for multi-label classification. Technical report, Department of Computer Science, National Taiwan University (2007)
3. Zhang, X., Graepel, T., Herbrich, R.: Bayesian online learning for multi-label and multi-variate performance measures. In: Proceedings of the Conference on Artificial Intelligence and Statistics (AISTATS). (2010) 956–963
4. Lipton, Z., Elkan, C., Naryanaswam, B.: Optimal thresholding of classifiers to maximize F1 measure. In: Proceedings of the European Conference on Machine Learning. (2014)
5. Petterson, J., Caetano, T.: Reverse multi-label learning. In: Advances in Neural Information Processing Systems. Volume 24. (2010)
6. Petterson, J., Caetano, T.: Submodular multi-label learning. In: Advances in Neural Information Processing Systems. Volume 25. (2011)
7. Jansche, M.: Maximum expected F-measure training of logistic regression models. In: Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP). (2005) 736–743
8. Kokkinos, I.: Boundary detection using F-measure-, filter- and feature- (f3) boost. In: Proceedings of the European Conference on Computer Vision (ECCV): Part II. (2010) 650–663
9. Musicant, D., Kumar, V., Ozgur, A.: Optimizing F-measure with support vector machines. In: Proceedings of the International FLAIRS Conference, Haller AAAI Press (2003) 356–360
10. Joachims, T.: A support vector method for multivariate performance measures. In: Proceedings of the International Conference on Machine Learning (ICML). (2005) 377–384
11. Waegeman, W., Dembczyński, K., Jachnik, A., Cheng, W., Hüllermeier, E.: On the bayes-optimality of f-measure maximizers. The Journal of Machine Learning Research **15**(1) (2014) 3333–3388
12. Lewis, D.: Evaluating and optimizing autonomous text classification systems. In: Proceedings of the International ACM Conference on Research and Development in Information Retrieval (SIGIR). (1995) 246–254
13. Chai, K.: Expectation of F-measures: Tractable exact computation and some empirical observations of its properties. In: Proceedings of the International ACM Conference on Research and Development in Information Retrieval (SIGIR). (2005)
14. Jansche, M.: A maximum expected utility framework for binary sequence labeling. In: Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL). (2007) 736–743

15. Quevedo, J., Luaces, O., Bahamonde, A.: Multilabel classifiers with a probabilistic thresholding strategy. Pattern Recognition **45** (2012) 876–883
16. Ye, N., Chai, K., Lee, W., Chieu, H.: Optimizing F-measures: a tale of two approaches. In: Proceedings of the International Conference on Machine Learning. (2012)
17. Dembczyński, K., Kotłowski, W., Koyejo, O., Natarajan, N.: Consistency analysis for binary classification revisited. In: Proceedings of the International Conference on Machine Learning (ICML). Volume 70., PMLR (2017) 961–969
18. Dembczyński, K., Waegeman, W., Cheng, W., Hüllermeier, E.: An exact algorithm for F-measure maximization. In: Advances in Neural Information Processing Systems. Volume 25. (2011)
19. Dembczyński, K., Jachnik, A., Kotlowski, W., Waegeman, W., Hüllermeier, E.: Optimizing the F-measure in multi-label classification: Plug-in rule approach versus structured loss minimization. In: Proceedings of the International Conference on Machine Learning (ICML). (2013)
20. Gasse, M., Aussem, A.: F-measure maximization in multi-label classification with conditionally independent label subsets. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I. (2016) 619–631
21. Wolpert, D.H.: Original contribution: Stacked generalization. Neural Networks **5**(2) (1992) 241–259
22. Cheng, W., Hüllermeier, E.: Combining instance-based learning and logistic regression for multilabel classification. Machine Learning **76**(2-3) (2009) 211–225
23. Agresti, A.: Categorical Data Analysis, Third Edition. John Wiley & Sons (2013)
24. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. `http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html` (2007)
25. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. `http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html` (2012)
26. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: European conference on computer vision, Springer (2014) 740–755
27. Kaggle: Planet: Understanding the amazon from space. `https://www.kaggle.com/c/planet-understanding-the-amazon-from-space` (2017)
28. Abadi, M., Agarwal, A., Barham, P., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org.
29. Chollet, F., et al.: Keras. `https://github.com/keras-team/keras` (2015)
30. Paszke, A., Gross, S., Chintala, S., et al.: Automatic differentiation in pytorch. In: NIPS-W. (2017)
31. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)