Meta–Gradient Boosted Decision Tree Model for Weight and Target Learning

Yury Ustinovskiy YURAUST@YANDEX-TEAM.RU
Valentina Fedorova VALYA17@YANDEX-TEAM.RU
Gleb Gusev GLEB57@YANDEX-TEAM.RU
Pavel Serdyukov PAVSER@YANDEX-TEAM.RU
Yandex, Moscow, Russia

Abstract

Labeled training data is an essential part of any supervised machine learning framework. In practice, there is a trade-off between the quality of a label and its cost. In this paper, we consider a problem of learning to rank on a large-scale dataset with low-quality relevance labels aiming at maximizing the quality of a trained ranker on a small validation dataset with high-quality ground truth relevance labels. Motivated by the classical Gauss-Markov theorem for the linear regression problem, we formulate the problems of (1) reweighting training instances and (2) remapping learning targets. We propose meta–gradient decision tree learning framework for optimizing weight and target functions by applying gradient-based hyperparameter optimization. Experiments on a large-scale real-world dataset demonstrate that we can significantly improve state-of-the-art machine-learning algorithms by incorporating our framework.

1. Introduction

The fundamental issue faced by many of supervised machine learning algorithms is the presence of noise in labels and features. Clearly, label noise is likely to have a negative impact on both the quality of the trained models and the evaluation accuracy. In this paper, we address the problem of label noise from the perspective of the quality of the trained model.

Traditionally, this problem is dealt with by various *noise reduction* techniques (Frénay & Verleysen, 2014; Strutz, 2010). For example, common approaches to noise reduction include *cleansing* (Brodley & Friedl, 1999) and *weighting* techniques. Noise cleansing techniques are similar to outlier detection and amount to filtering out samples which 'look like' mislabeled for some reason. With the weighting approach none of the samples are completely discarded, while their impact on a machine learning algorithm is controlled by *weights*, representing our confidence in a particular label. In both approaches, one has to use certain heuristics (Verbaeten & Van Assche, 2003) or assume some underlying model for the noise generation (Li et al., 2007).

With this paper we study a general problem of supervised learning in the presence of noise in labels (noisy supervised learning, or **NSL** for short). Unlike the classical approaches (Hastie et al., 2009, §3.2.2, Lawrence & Schölkopf, 2001), we do not assume any prior knowledge on the noise nature and its distribution. Moreover, we let the labels in the training dataset (1) to be biased relative to the implicit ground-truth labels; (2) to have different noise variance.

Let us discuss several important examples of such training tasks. **Click prediction problem** is an important task in online advertising and Web search personalization (Graepel et al., 2010; Shen et al., 2005). Typically, samples in the training datasets for these tasks are labeled using *implicit feedback*, i.e., labels are inferred from user behavior, such as clicks on documents/advertisements, time spent on a document, fact of transaction after an advertisement click, etc. It is well-known that such labels are liable to positional and presentational bias (Ciaramita et al., 2008). A possible common solution to this problem is to reduce the noise by samples *relabeling* technique. For example, one might assign positive value only to *satisfied clicks*, assign different values to clicks with long/short dwell-time or discount the click value by snippet/advertisement attractiveness. Another problem is **learning with crowdsourced labels**. Recent rapid take-up of crowdsourcing marketplaces, e.g., Amazon MTurk[1], provides a cheap way of collecting large supervised datasets (Muhammadi et al., 2013). However, due to a lack of expertise and presence of spammer workers, such labels often hugely vary in quality. To overcome this issue, employers assign each item to multi-

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

[1] http://www.mturk.com/

ple workers. Afterwards, item's multiple labels are aggregated into a consensus label with the use of a certain consensus algorithm, e.g., majority voting, label averaging, etc (see, e.g. Zhou et al., 2014). Various consensus models are known to significantly improve the precision of raw crowd labels.

The examples above share one common property, which is crucial for the algorithm we propose in this paper. Namely, for each item in the training dataset, besides a noisy label itself, we observe several additional variables, which altogether are further referred to as **label features**. Examples of label features are: fact of a click, click position, and dwell time in the click prediction problem; worker's experience, outputs of various consensus models in the problem of learning with crowdsourced labels.

In this paper, we propose a method which addresses NSL problem by utilizing label features directly within a machine learning framework. Our method assigns to each learning sample (1) learning target and (2) its weight, which captures the confidence in the value of the learning target. This approach is motivated by the classical Gauss-Markov theorem and its generalizations to the cases of correlated heteroscedastic errors (Section 2). We model target t and weight w as functions of label features. Training dataset equipped with learning targets and weights is fed to the Generalized Least Squares (GLS) algorithm. Our goal is to tune weights and labels in such a way that the output of GLS algorithm has the maximum quality on a held-out validation dataset. We treat this maximization problem as a separate supervised machine learning problem, which learns target and weight functions t and w on the validation dataset (Section 3). This interpretation resembles gradient-based hyperparameter optimization in (Maclaurin et al., 2015). In the case of a smooth loss function $\mathcal{L}$, we propose an efficient gradient descent algorithm, which learns t and w as ensembles of decision trees (Section 4). In the context of click prediction problem, a weight optimization framework was proposed in (Ustinovskiy et al., 2015). Unlike this paper we (1) are optimizing both target and weight functions; (2) what is more important train these functions as ensembles of decision trees, significantly advancing the effectiveness of our framework.

In the experimental part of the paper (Section 7) we evaluate our framework within learning with crowdsourced labels task. We conduct experiments on a large-scale dataset shared with us by a commercial search engine (Section 6).

2. Motivation

In this section we recall one classical learning task in the presence of noise, that is *linear regression model* with unbiased errors. Under this model we observe a vector $\boldsymbol{z} = \boldsymbol{X}\boldsymbol{b} + \boldsymbol{\epsilon}$, where $\boldsymbol{z} \in \mathbb{R}^S$ is a vector of observables, $\boldsymbol{X}$ is non-random known design matrix, $\boldsymbol{b} \in \mathbb{R}^N$ is unobservable vector of parameters and $\boldsymbol{\epsilon} \in \mathbb{R}^S$ is a vector of noise components. Assume that (1) vector $\boldsymbol{z}$ is unbiased, i.e., expectation of $\boldsymbol{\epsilon}$ is zero; (2) components of $\boldsymbol{\epsilon}$ have finite variance; (3) $\operatorname{cov}(\epsilon_i, \epsilon_j) = 0$, i.e., the error terms are uncorrelated.

Then the best linear unbiased estimator (BLUE) of the vector $\boldsymbol{b}$, i.e., estimator with the minimal component-wise variance, is given by Generalized Least Squares estimator:

$$\widehat{\boldsymbol{b}} := \operatorname*{argmin}_{\boldsymbol{\beta} \in \mathbb{R}^N} \sum_{i=1}^{S} \frac{1}{\sigma_i^2} (z_i - \boldsymbol{x}_i \cdot \boldsymbol{\beta})^2, \tag{1}$$

where σ_i^2 is the variance of i-th component of $\boldsymbol{\epsilon}$ and $\boldsymbol{x}_i$ is the i-th row of matrix $\boldsymbol{X}$. Solution of this minimization problem is known as Aitken estimator in the generalized version of Gauss-Markov Theorem (Aitken, 1935, §2). Intuitively, the noisier the i-th observation, the less impact it should have on the loss function. This impact is controled by a *weight* $w_i = 1/\sigma_i^2$.

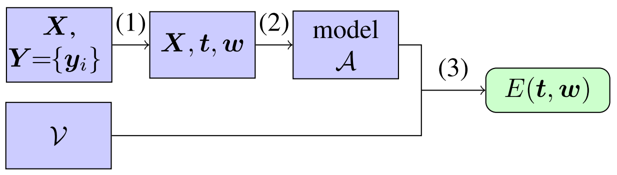
Condition (3) above could be relaxed, i.e., Aitken estimator has a generalization to the case of correlated errors. Then in minimization problem (1) one has to substitute components z_i with certain linear combinations $z_i' = \sum_{j=1}^{S} c_{ij} z_j$ and change matrix $\boldsymbol{X}$ and error vector $\boldsymbol{\epsilon}$ accordingly, (see Aitken, 1935, §5). That is instead of predicting observables themselves, we predict linear combinations of observables. Throughout this paper we refer to the quantities estimated within machine learning algorithm (z_i or z_i' in the example above) as *targets*.

Unfortunately, it is hard to apply the above methods in practice, since (1) the correlations $\operatorname{cov}(\epsilon_i, \epsilon_j)$ are not known; (2) we are not interested in minimization of the variance of estimator $\boldsymbol{b}$ (equivalently weighted sum of squared residuals), but would like to optimize some other quantities, e.g., minimize variances of $\log b_i$, or predict correctly the mutual order of components of $\boldsymbol{b}$. Motivated by the initial theoretical result and taking in account these issues, we suggest tuning weights and targets in Equation (1) in order to optimize some arbitrary loss function.

3. Problem formulation

In this section we give a general description of the problem we are solving. In the subsequent sections we restrict the scope to a certain class of machine learning algorithms and provide a rigorous formulation within this specific.

Let $\mathcal{X}$ be a training dataset with the corresponding $S \times N$ design matrix $\boldsymbol{X}$, where S is the number of items in $\mathcal{X}$ and N is the number of features. That is the i-th training sample is represented by a row feature vector $\boldsymbol{x}_i \in \mathbb{R}^N$.


$\boldsymbol{X}$, $\boldsymbol{Y}$={$\boldsymbol{y}_i$}
(1)
$\boldsymbol{X}, \boldsymbol{t}, \boldsymbol{w}$
(2)
model $\mathcal{A}$
(3)
$E(\boldsymbol{t}, \boldsymbol{w})$
$\mathcal{V}$

(1) Target and weight optimization
(2) Training machine learning algorithm $\mathcal{A}$
(3) Evaluation of the model $\mathcal{A}$ on the validation dataset

Figure 1. General framework for noisy supervised learning.

Let $\mathcal{A}$ be some fixed machine learning algorithm. Further this algorithm is sometimes referred to as *background algorithm*, since our framework is an additional supervised machine learning algorithm on the top of $\mathcal{A}$. Assume that $\mathcal{A}$ utilizes both learning targets and weights of training samples in a known way. That is, given a column-vector of targets $\boldsymbol{t} = (t_1, \ldots, t_S)$, a column-vector of weights $\boldsymbol{w} = (w_1, \ldots, w_S)$ and a design matrix $\boldsymbol{X}$, algorithm $\mathcal{A}$ yields a function (slightly abusing notation we use the same symbol for it):

$$\mathcal{A}(\boldsymbol{t}, \boldsymbol{w}; \cdot) \colon \boldsymbol{x} \mapsto \mathcal{A}(\boldsymbol{t}, \boldsymbol{w}; \boldsymbol{x}) \in \mathbb{R},$$

assigning to a sample with a feature vector $\boldsymbol{x}$ the output of the trained function.

Let $\mathcal{V}$ be a validation dataset consisting of S_v samples with i-th sample represented by a row feature vector $\boldsymbol{v}_i$. Finally, assume we are given a *loss function* $\mathcal{L} \colon \mathbb{R}^{S_v} \to \mathbb{R}$, which assigns to the vector of outputs of a function $\mathcal{A}(\boldsymbol{t}, \boldsymbol{w})$, applied to $\mathcal{V}$, the following loss:

$$E(\boldsymbol{t}, \boldsymbol{w}) := \mathcal{L}(\mathcal{A}(\boldsymbol{t}, \boldsymbol{w}; \boldsymbol{v}_1), \ldots, \mathcal{A}(\boldsymbol{t}, \boldsymbol{w}; \boldsymbol{v}_{S_v})) \in \mathbb{R}. \quad (2)$$

Our goal is to minimize the loss on the validation dataset:

$$\widehat{\boldsymbol{t}}, \widehat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{t}, \boldsymbol{w}} E(\boldsymbol{t}, \boldsymbol{w}). \quad (3)$$

We assume that each item $i \in \mathcal{X}$ is equipped with an M-dimensional label feature vector $\boldsymbol{y}_i \in \mathbb{R}^M$. We consider only target and weight vectors $\boldsymbol{t}$ and $\boldsymbol{w}$ of the form $(t(\boldsymbol{y}_1), \ldots, t(\boldsymbol{y}_S))$, $(w(\boldsymbol{y}_1), \ldots, w(\boldsymbol{y}_S))$, where $t(\cdot)$ and $w(\cdot)$ are some functions of label features.

The outline of the global learning scheme is depicted in Figure 1. We focus on step (1), namely, given a fixed learning to rank algorithm at step (2) and a fixed evaluation metric at step (3) we aim at direct minimization of the loss by tuning target and weight functions.

To solve minimization problem (3) among the functions of label features, we develop a *meta–gradient boosted decision tree model* (MGBDT). The term *meta* comes from the fact that vectors $\boldsymbol{t}$ and $\boldsymbol{w}$ can be treated as hyperparameters of algorithm $\mathcal{A}$ and we are optimizing these hyperparameters. To perform meta–gradient descent we will need the gradients

$$\begin{aligned} \frac{\partial E(\boldsymbol{t}, \boldsymbol{w})}{\partial \boldsymbol{t}} &:= \left\{ \frac{\partial E(\boldsymbol{t}, \boldsymbol{w})}{\partial t_i} \right\}_{i=1}^{S}, \\ \frac{\partial E(\boldsymbol{t}, \boldsymbol{w})}{\partial \boldsymbol{w}} &:= \left\{ \frac{\partial E(\boldsymbol{t}, \boldsymbol{w})}{\partial w_i} \right\}_{i=1}^{S}. \end{aligned} \quad (4)$$

In the next section, we derive the explicit formulas for these gradients under additional assumption on the background algorithm $\mathcal{A}$, provided the loss function $\mathcal{L}$ is smooth. Also we give some upper bounds on the computational complexity of our framework.

4. Meta–gradient boosted decision tree model

4.1. Meta–gradient descent

In this subsection, we compute gradients (4). For a function $\boldsymbol{f}(\boldsymbol{z})$ with multivariate inputs and outputs, the gradients $\partial \boldsymbol{f} / \partial \boldsymbol{z}$ are Jacobian matrices. We summarize notations in Table 1. Since it will be important to distinguish column and row vectors, we also report this information. Using the chain rule we rewrite (4) as:

Table 1. List of notations.

$\mathcal{X}$	training dataset
$\mathcal{V}$	validation dataset
N	number of ordinary features for samples in $\mathcal{X}$ and $\mathcal{V}$
M	number of label features for samples in $\mathcal{X}$
S	number of samples in $\mathcal{X}$
S_v	number of samples in $\mathcal{V}$
$\boldsymbol{x}_i$	row vector of features of a sample $i \in \mathcal{X}$
$\boldsymbol{v}_i$	row vector of features of a sample $i \in \mathcal{V}$
$\boldsymbol{y}_i$	row vector of label features of a sample $i \in \mathcal{X}$
$\mathcal{A}$	background learning to rank algorithm, used at step (2) in Figure 1
$\mathcal{A}(\boldsymbol{t}, \boldsymbol{w}; \cdot)$	function trained via $\mathcal{A}$
$\boldsymbol{w}$	S-dimensional column vector of training samples' weights
$\boldsymbol{t}$	S-dimensional column vector of training samples' targets
$\boldsymbol{X}$	$S \times N$ design matrix for $\mathcal{X}$
$\boldsymbol{Y}$	$S \times M$ matrix of label features
$\boldsymbol{W}$	$S \times S$ diagonal matrix with weights w_i
$\boldsymbol{V}$	$S_v \times N$ design matrix for $\mathcal{V}$
$\widehat{\boldsymbol{b}}$	N-dimensional column vector of optimal parameters of a GLS model
$\boldsymbol{s}$	S_v-dimensional column vector of validation samples' scores

$$\frac{\partial E(\boldsymbol{t},\boldsymbol{w})}{\partial \boldsymbol{t}} = \sum_{j\in\mathcal{V}} \frac{\partial \mathcal{A}(\boldsymbol{t},\boldsymbol{w};\boldsymbol{v}_j)}{\partial \boldsymbol{t}} \frac{\partial \mathcal{L}}{\partial s_j}; \qquad (5)$$
$$\frac{\partial E(\boldsymbol{t},\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{j\in\mathcal{V}} \frac{\partial \mathcal{A}(\boldsymbol{t},\boldsymbol{w};\boldsymbol{v}_j)}{\partial \boldsymbol{w}} \frac{\partial \mathcal{L}}{\partial s_j},$$

where $s_j = \mathcal{A}(\boldsymbol{t},\boldsymbol{w};\boldsymbol{v}_j)$ is the score assigned by a model $\mathcal{A}$ to the j-th sample in $\mathcal{V}$. To compute these gradients we will need to assume that the following two conditions are satisfied:

C1: Loss function $\mathcal{L}\colon \mathbb{R}^{S_v} \to \mathbb{R}$ is differentiable with respect to the scores assigned to the individual samples in $\mathcal{V}$;

C2: The outputs of the function $\mathcal{A}(\boldsymbol{t},\boldsymbol{w};\cdot)$ trained via background algorithm $\mathcal{A}$ are differentiable with respect to the targets and weights t_i and w_i of the individual samples of $\mathcal{X}$.

Derivatives in C2 do not necessarily exist and even their finite-difference approximation might be computationally expensive, since it requires additional trainings of the algorithm $\mathcal{A}$. Since the computational complexity turns out to be an important concern, we require both sets of derivatives to be *efficiently* computable. Property C1 is satisfied, whenever the loss functions has a gradient. This is the case for the residual sum of squares, logistic loss, cross entropy loss, etc. We are not aware of property C2 for a general background machine learning algorithm $\mathcal{A}$. However, we are able to explicitly compute derivatives $\partial\mathcal{A}(\boldsymbol{t},\boldsymbol{w};\boldsymbol{v}_j)/\partial w_i$ and $\partial\mathcal{A}(\boldsymbol{t},\boldsymbol{w};\boldsymbol{v}_j)/\partial t_i$ in a special case of the **Generalized Least Squares** (GLS) model, see Proposition 2 below.

Definition 1. *GLS model* is a linear model, which minimizes weighted residual sum of squares:

$$\mathrm{GLS}(\boldsymbol{t},\boldsymbol{w};\boldsymbol{x}) = \boldsymbol{x}\cdot\widehat{\boldsymbol{b}}, \text{ where}$$
$$\widehat{\boldsymbol{b}} = \mathrm{argmin}_{\boldsymbol{b}} \sum_{i=1}^{S} w_i(\boldsymbol{x}_i\cdot\boldsymbol{b} - t_i)^2. \qquad (6)$$

Inputs of GLS model are design matrix $\boldsymbol{X}$, vector of targets $\boldsymbol{t}$ and vector of weights $\boldsymbol{w}$. The output of GLS model is a function $\mathrm{GLS}(\boldsymbol{t},\boldsymbol{w};\cdot)$.

Remark. It is well-known that simple linear models often underperform in real-life applications. Note, however, that this shortcoming could be treated by Generalized Additive Models (Hastie et al., 2009, §9). In the experimental part we use this idea and expand feature space by adding the outputs of pre-learned decision trees as additional features.

Proposition 2. *Let $\mathcal{A}$ = GLS be a Generalized Least Squares model (6) trained on the dataset $\mathcal{X}$ with target and weight vectors $\boldsymbol{t}$ and $\boldsymbol{w}$ respectively. Then condition C2 is satisfied, i.e., derivatives $\partial GLS(\boldsymbol{t},\boldsymbol{w};\boldsymbol{x})/\partial\boldsymbol{t}$ and $\partial GLS(\boldsymbol{t},\boldsymbol{w};\boldsymbol{x})/\partial\boldsymbol{w}$, where $\boldsymbol{x}\in\mathbb{R}^N$ is a row feature vector, exist. Specifically:*

$$\frac{\partial GLS(\boldsymbol{t},\boldsymbol{w};\boldsymbol{x})}{\partial\boldsymbol{t}} = \boldsymbol{x}\boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{W}, \qquad (7)$$
$$\frac{\partial GLS(\boldsymbol{t},\boldsymbol{w};\boldsymbol{x})}{\partial\boldsymbol{w}} = \boldsymbol{x}\boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{T},$$

where $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{w})$ is the diagonal $S\times S$ matrix, $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$, and $\boldsymbol{T} = \mathrm{diag}(\boldsymbol{t} - \boldsymbol{X}\,\widehat{\boldsymbol{b}})$.

Proof. The core property of GLS model is that the minimizer (6) admits a closed-form presentation:

$$\widehat{\boldsymbol{b}} = (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{t}. \qquad (8)$$

Let $\widehat{\boldsymbol{t}} := \boldsymbol{X}\cdot\widehat{\boldsymbol{b}}$ be the column vector of values of the GLS model on $\mathcal{X}$. Differentiating the equality $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{t} = \boldsymbol{Z}\widehat{\boldsymbol{b}}$ with respect to w_j one gets:

$$\boldsymbol{X}^T\frac{\partial\boldsymbol{W}}{\partial w_j}\boldsymbol{t} = \frac{\partial\boldsymbol{Z}\widehat{\boldsymbol{b}}}{\partial w_j} = \frac{\partial\boldsymbol{Z}}{\partial w_j}\widehat{\boldsymbol{b}} + \boldsymbol{Z}\frac{\partial\widehat{\boldsymbol{b}}}{\partial w_j} =$$
$$= \boldsymbol{X}^T\frac{\partial\boldsymbol{W}}{\partial w_j}\boldsymbol{X}\widehat{\boldsymbol{b}} + \boldsymbol{Z}\frac{\partial\widehat{\boldsymbol{b}}}{\partial w_j} = \boldsymbol{X}^T\frac{\partial\boldsymbol{W}}{\partial w_j}\widehat{\boldsymbol{t}} + \boldsymbol{Z}\frac{\partial\boldsymbol{b}}{\partial w_j}. \qquad (9)$$

Finally, a little manipulation with Equation (9) yields:

$$\frac{\partial\widehat{\boldsymbol{b}}}{\partial w_j} = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\frac{\partial\boldsymbol{W}}{\partial w_j}(\boldsymbol{t} - \widehat{\boldsymbol{t}}). \qquad (10)$$

Collecting these expressions over all components of $\boldsymbol{w}$ into one vector and recalling that $\partial\boldsymbol{W}/\partial w_j = E_{jj}$ is an elementary matrix with one nonzero entry, one gets:

$$\frac{\partial\widehat{\boldsymbol{b}}}{\partial\boldsymbol{w}} = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\mathrm{diag}(\boldsymbol{t} - \widehat{\boldsymbol{t}}) = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{T}.$$

Computation of derivatives of $\widehat{\boldsymbol{b}}$ with respect to $\boldsymbol{t}$ is a bit simpler, since only factor $\boldsymbol{t}$ depends on $\boldsymbol{t}$:

$$\frac{\partial\widehat{\boldsymbol{b}}}{\partial\boldsymbol{t}} = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{W}\frac{\partial\boldsymbol{t}}{\partial\boldsymbol{t}} = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{W}. \qquad (11)$$

Combining equations (10) and (11) with the definition of GLS (6) we obtain required formulas (7). □

Remark. Matrix $\boldsymbol{Z}$ is not necessarily invertible. Partly for this reason, we follow a standard trick and use L_2-regularized version of GLS model: $\widehat{\boldsymbol{b}} = \mathrm{argmin}_{\boldsymbol{b}}\sum_{i=1}^{S} w_i(\boldsymbol{x}_i\cdot\boldsymbol{b} - t_i)^2 + \mu||\boldsymbol{b}||^2$, where $\mu > 0$ is a constant. It is straightforward to check that in this case matrix $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$ is just substituted with $\boldsymbol{Z} + \mu\boldsymbol{I}_N$. For a generic choice of μ this matrix is invertible.

With the same notation as in Proposition 2 we have the following immediate corollary.

Corollary 3. *Assume that the loss function $\mathcal{L}(\boldsymbol{s})$ satisfies C1, i.e, it is differentiable with respect to the vector $\boldsymbol{s}$ (scores of samples in $\mathcal{V}$). Let $\mathcal{A} = GLS$ be the background machine learning algorithm. Then the gradients $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{t}$ and $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{w}$ exist and are given by the formulas:*

$$\begin{aligned} \frac{\partial E(\boldsymbol{t}, \boldsymbol{w})}{\partial \boldsymbol{t}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}} \boldsymbol{V} \boldsymbol{Z}^{-1} \boldsymbol{X}^T \boldsymbol{W}, \\ \frac{\partial E(\boldsymbol{t}, \boldsymbol{w})}{\partial \boldsymbol{w}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}} \boldsymbol{V} \boldsymbol{Z}^{-1} \boldsymbol{X}^T \boldsymbol{T}. \end{aligned} \tag{12}$$

Theorem 4. *Under the assumptions of Corollary 3, given the derivative $\partial\mathcal{L}/\partial \boldsymbol{s}$, the computational complexity of the gradients (12) is bounded above by $O(NS_v) + O(N^2S)$ arithmetic operations.*

Proof. Let us bound computational complexity for the gradient $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{w}$. Matrices $\partial\mathcal{L}/\partial \boldsymbol{s}$, $\boldsymbol{V}$, $\boldsymbol{Z}^{-1}$, $\boldsymbol{X}^T$ have sizes $1 \times S_v$, $S_v \times N$, $N \times N$, $N \times S$ respectively. First, let us recall that given two matrices of sizes $p \times q$ and $q \times r$ computation of their product requires $O(pqr)$ arithmetic operations. Hence, computation of matrices $\boldsymbol{T}$ and $\boldsymbol{Z}$ (see Proposition 2) requires $O(NS)$ and $O(N^2S)$ operations respectively.

Multiplication in the order $(((\partial\mathcal{L}/\partial \boldsymbol{s}\, \boldsymbol{V})\boldsymbol{Z}^{-1})\boldsymbol{X}^T)\boldsymbol{T}$ requires $O(NS_v)$ operation for the first operation, $O(N^2)$ operations for the second multiplication with $\boldsymbol{Z}^{-1}$ (which is equivalent to solving the system of linear equations, so matrix $\boldsymbol{Z}^{-1}$ does not need to be computed explicitly), $O(NS)$ and $O(S)$ for the last two multiplications (the last $O(S)$ comes from the fact that $\boldsymbol{T}$ is a diagonal matrix). All in all this gives a bound $O(NS_v) + O(N^2S)$.

Analogously, one bounds the computational complexity of $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{t}$. □

Remark. The most resource-intensive computation is the calculation of matrix $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$ (or $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} + \mu \boldsymbol{I}_N$ in the L_2-regularized case).

4.2. Decision tree fitting

The gradients $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{t}$ and $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{w}$ computed in the previous section allow to directly minimize the loss on the validation dataset $\mathcal{V}$ by learning target and weight functions of label features $t(\cdot)$ and $w(\cdot)$. For the majority of machine learning algorithm it suffice to have these gradients to perform gradient descent. Ensembles of decision tress are known to generate quite general and effective models (Friedman, 2001), so in this paper we focus on boosted decision trees models. Specifically, we learn the target function in the form:

$$t(\boldsymbol{y}) = \sum_{j=1}^{J} h_j^t(\boldsymbol{y}),$$

Algorithm 1 Meta–gradient steepest descend optimization of weight and target functions.

```
1:  Input training dataset $\mathcal{X}$ and validation dataset $\mathcal{V}$
2:  Parameters number of iterations $J$, step size regularization $\epsilon$, parameter of $L_2$ regularization $\mu$ for GLS, maximum depth of a decision tree $d$
3:  Initialization $t(\cdot) = w^{\text{raw}}(\cdot) = 0$,
    $w(\cdot) = \sigma(w^{\text{raw}}(\cdot)) = 1/2$
4:  for $j = 0$ to $J$ do
5:      $h_j^t = \operatorname{argmin}_{h(\cdot)} \sum_{j \in \mathcal{V}} (h(\boldsymbol{y}_j) - \partial E(\boldsymbol{t}, \boldsymbol{w})/\partial t_j)^2$
6:      $h_j^w = \operatorname{argmin}_{h(\cdot)} \sum_{j \in \mathcal{V}} (\sigma'(w_j)h(\boldsymbol{y}_j) - \partial E(\boldsymbol{t}, \boldsymbol{w})/\partial w_j)^2$
7:      update functions $t, w^{\text{raw}}, w$:
        $t(\cdot) := t(\cdot) + \epsilon h_j^t(\cdot)$
        $w^{\text{raw}}(\cdot) := w^{\text{raw}}(\cdot) + \epsilon h_j^w(\cdot)$
        $w(\cdot) := \sigma(w^{\text{raw}}(\cdot))$
8:      update $\boldsymbol{t}, \boldsymbol{w}, \boldsymbol{T}, \boldsymbol{W}, \boldsymbol{Z}$ accordingly
9:  end for
10: return functions $t(\cdot)$, $w(\cdot)$
```

where $\boldsymbol{y} \in \mathbb{R}^M$ is a vector of label features and $h_j^t(\boldsymbol{y})$ are weak learners — *decision trees*, (cf. Breiman et al., 1984).

For the weight function we additionally pass an ensemble of decision trees through the sigmoid transform $\sigma(\lambda) = 1/(1 + e^{-\lambda})$ to ensure that all weights are nonegative (one might use different transforms as well): $w(\boldsymbol{y}) = \sigma(w^{\text{raw}}(\boldsymbol{y}))$, where $w^{\text{raw}}(\boldsymbol{y}) = \sum_{j=1}^{J} h_j^w(\boldsymbol{y})$.

Functions $t(\cdot)$ and $w(\cdot)$ are iteratively constructed by fitting new weak learners $h_j^t(\cdot)$ and $h_j^w(\cdot)$ to the gradients $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{t}$ and $\partial E(\boldsymbol{t}, \boldsymbol{w})/\partial \boldsymbol{w}$ at each step of gradient descent. The algorithm for training target and weight functions is summarized in pseudocode in Algorithm 1. As usual with gradient boosted decision trees, minimization problem in lines 5 and 6 is solved within decision trees of fixed depth d. Note that there is an additional factor $\sigma'(w_j)$ in line 6, since we are fitting not ensemble of decision trees itself, but its sigmoid transform. In the experimental part we use DecisionTreeRegressor implementation from scikit-learn package (Pedregosa et al., 2011) to solve regression problems at lines 5 and 6 of Algorithm 1. Fitting a single tree with this implementation has complexity $O(NS \log S)$[2].

5. Learning to rank task with crowdsourced labels

In this section we explain how our framework is adopted to the problem of learning to rank with crowdsourced labels. This is the focus of the experimental part of the paper.

[2]http://scikit-learn.org/stable/modules/tree.html#complexity

5.1. Learning to rank

In the classical learning to rank task training and validation datasets consist of query-document pairs with each label reflecting the relevance of a document to a query. The goal is to train a ranking function, which sorts all documents within each query according to the outputs of the ranking function. The common ranking evaluation measures include Mean Average Precision (MAP), Discounted Cumulative Gain (DCG), Expected Reciprocal Rank (ERR) (Sakai, 2014).

The method of Section 4 is not applicable directly to learning to rank task, since most of the ranking evaluation measures do not satisfy condition C1. Indeed, these measures are discontinuous and locally constant (at a generic point) functions of the outputs of the ranking function, so the gradient $\partial\mathcal{L}/\partial\boldsymbol{s}$ is either zero, or not defined. There are several approaches to handle this issue preforming *smoothing procedure* (Taylor et al., 2008) or introducing *pseudo-gradient* (Burges et al., 2007). In this case, even though objective functions, e.g., MAP, DCG, ERR, do not satisfy condition C1, we are able to apply Algorithm 1 by plugging in smoothed or pseudo gradient .

In the experimental part we fix DCG evaluation measure and use LambdaRank pseudo gradient (Burges, 2010) as a substitute for the non-existing "genuine gradient" $\partial\mathcal{L}/\partial\boldsymbol{s}$ in (12). Computational complexity of lambda gradient per one query is $O(k^2)$, where k is the number of documents (cf. Burges, 2010, §3). Hence, assuming validation dataset has k documents per each query, the computation of the LambdaRank pseudo gradient requires $S_v/k \cdot O(k^2) = O(kS_v)$ operations. Hence, the computation of the gradients $\partial E(\boldsymbol{t},\boldsymbol{w})/\partial\boldsymbol{t}$ and $\partial E(\boldsymbol{t},\boldsymbol{w})/\partial\boldsymbol{w}$ (see Equations (12)) in the learning-to-rank task takes $O(kS_v) + O(NS_v) + O(N^2S)$ operations. Adding the complexity of tree fitting, we get an upper bound on the total computational complexity of one iteration of Algorithm 1:

$$O(kS_v) + O(NS_v) + O(N^2S) + O(NS\log S)$$

arithmetic operations.

5.2. Crowdsourced relevance labels

The problem of learning to rank with crowdsourced labels is not very different from a classical learning to rank task, however, the origin of the relevance labels imply several modifications of the training and validation processes.

Labels collected via crowdsourcing have a number of serious shortcomings: (S1) crowd workers are usually not provided with detailed instructions like those compiled for professional assessors, since the majority of them would either refuse or fail to follow complicated guidelines; (S2) individual workers vary greatly in the quality of their assessments. The first property forces to use coarse relevance scale, e.g., binary, which is different from the common fine graded scales, e.g., in DCG. Second property urges employers to modify the labeling process in order to gain some evidence for each label being correct. Namely, the employers often: (P1) place 'honeypot' tasks, i.e., tasks with a known true label, to filter out workers severely underperforming on these tasks; (P2) assign each task to multiple workers in order to evaluate and aggregate their answers.

Each query-document pair in $\mathcal{X}$ is labeled by several crowd workers, so to learn a ranker we might either use all crowd labels separately, or aggregate them into one consensus label (see, e.g., Dawid & Skene, 1979, Lee et al., 2010, Whitehill et al., 2009). Since crowd labels are noisy and coarse, we assume that validation dataset $\mathcal{V}$ is, on contrary, labeled by professional assessors with the standard 5-graded relevance scheme making our evaluation methodology more precise and comprehensive.

6. Data

In this section we describe the datasets, provided by a commercial search engine. The structure of these datasets and its features reveal the motivation for the particular problem we are solving and clarify the main ideas behind our approach.

The first dataset consists of 132K query-document pairs. Each query-document pair is assigned three *binary* relevance labels from three crowd workers at a crowdsourcing platform. With these query-document pairs and crowd labels a dataset $\mathcal{X}$ consisting of $132\text{K} \times 3 = 396\text{K}$ samples is formed. One sample in $\mathcal{X}$ is a query-document pair together with one label assigned by one crowd worker. Besides these query-document pairs, there are 1900 *honeypot* tasks completed by the same workers (each task is completed by several workers). These are query-document pairs labeled by professional assessors with the same instructions. Usually, honeypots are used to detect and penalize spammer workers. Query-document pairs corresponding to honeypots do not get into $\mathcal{X}$.

The second dataset $\mathcal{V}^L$ (large validation dataset) is collected in a standard manner for a learning to rank task. Namely, every query-document pair in $\mathcal{V}^L$ is judged once by one professional assessor and is endowed with a label, corresponding to one of the relevance classes {*Perfect, Excellent, Good, Fair, Bad*}. Parts of this dataset are intended for evaluation, supervised training and tuning of our framework.

Every query-document pair in both datasets $\mathcal{V}^L$ and $\mathcal{X}$ is represented by a feature vector $\boldsymbol{x} \in \mathbb{R}^N$. These are standard *ranking features*, including text and link relevance, query characteristics, document quality, user behavior fea-

Table 2. List of label features

1	$cl^{(w)}$	the crowd label assigned by the worker
2-5	$\pi_{ij}^{(w)}$	confusion matrix of the worker in DS model
6	$a^{(w)}$	worker's parameter in GLAD model
7-8	$\mathrm{P}(cl^{(w)} = t_i)$	probability the crowd label is correct under DS/GLAD model
9-10	$\mathrm{P}(cl^{(w)} = 1)$	probability of positive label under DS/GLAD model
11	$\log(n^{(w)})$	logarithm of the number of tasks, completed by the worker
12	$n_0^{(w)}/n^{(w)}$	fraction of negative labels, assigned by the worker
13	$f_{\text{hp}}^{(w)}$	fraction of correctly-labeled honeypots by a worker

tures, etc. The particular choice of ranking features is not important for our framework and is out of the scope of the paper. We do not disclose particular features due to the proprietary nature of the dataset.

Besides ranking features, the samples in the crowdsourced dataset $\mathcal{X}$ are endowed with a number of *label features*. These features comprise numerical information about the worker who assigned the label as well as about the task, and the label itself. While a specific choice of label features is also not important for our framework, we describe them in details, since they reveal motivation behind learning new targets and weights. Intuitively, the purpose of label features is to provide evidence of the crowd label being correct.

To generate features for labels we utilize two classical consensus models: (DS) (Dawid & Skene, 1979) and Generative model of Labels, Abilities, and Difficulties (GLAD) (Whitehill et al., 2009). Both these models iteratively update intrinsic parameters via EM-algorithm. We use outputs of these models as well as the intrinsic parameters as label features. Besides the outputs of these two models we use several simple statistics on tasks and workers. For a worker w, let $n^{(w)}$ be the number of completed tasks, $n_0^{(w)}$ the number of zero-labeled documents. The complete set of label features is listed in Table 2. Some important details on the dataset are provided in Table 3

7. Experiments

There are no large-scale publicly available crowdsourced learning to rank datasets, so for the experiments we use the proprietary dataset described in Section 6. It is used to evaluate various baselines and compare them with our process-

Table 3. Statistics on the commercial dataset.

$\mathcal{X}$	# of queries	7200
	# of query-document pairs	132K
	# of samples	396K
	# of workers	1720
	average # of tasks per worker	233
	# of unique honeypot tasks	1900
	# of completed honeypot tasks	43300
$\mathcal{V}$	# of queries	6600
	# of query-document pairs (samples)	90K

ing framework. In all the experiments we use *discounted cumulative gain* measure at positions $k = 1, 5, 10$, denoted by DCG@k (Järvelin & Kekäläinen, 2002), as the ranking quality metric. Graded relevance labels are mapped into the standard numerical gains $\{15, 7, 3, 1, 0\}$.

Experiment design To evaluate our algorithm along with a number of baselines we perform 5-fold cross validation. Each time, the large validation set $\mathcal{V}^L$ is randomly split into 3 parts $\mathcal{V}$, $\mathcal{V}^{\text{validate}}$ and $\mathcal{V}^{\text{test}}$ on the basis of the unique query id. The part $\mathcal{V}$ is used to train Algorithm 1, specifically, to compute the LambdaRank pseudo gradients. The part $\mathcal{V}^{\text{validate}}$ is used to tune various hyperparameters of the background machine learning algorithm $\mathcal{A}$ and Algorithm 1, i.e., L_2-regularization parameter μ; J and ϵ in our processing algorithm. The output of Algorithm 1 is a pair of functions $t(\cdot)$ and $w(\cdot)$. We compute vectors $\boldsymbol{t}$, $\boldsymbol{w}$ and train GLS model on $\mathcal{X}$, using these targets and weights. Finally, the third part $\mathcal{V}^{\text{test}}$ is used to evaluate this ranker trained via GLS model (our approach) and the baseline rankers.

Due to the proprietary nature of the data, we do not disclose the actual values of DCG@k metrics on the test dataset $\mathcal{V}^{\text{test}}$. Instead, we report the *relative improvement* over the 'majority vote' baseline, described further. We denote this relative improvement by ΔDCG@k.

As we mention in the remark after Definition 1, we expand the feature space to reinforce GLS model. To construct the additional features, we train once an ensemble of the gradient boosted decision trees on $\mathcal{X}$ in the regression regime. Each individual tree in this ensemble (after standardization) is treated as a single extended ranking feature. The optimal number of extended features is estimated on $\mathcal{V}^{\text{validate}}$ and equals $T = 200$.

Baselines Since the specific problem we formulate in this paper is novel, there are no baselines intended specifically for our setting. For this reason, we adopt various common approaches to noise reduction and consensus modeling. As the background algorithm for the baselines RD and Lamb-

Table 4. Comparsion of various baselines with our framework. † denotes a significant improvement over the best baseline with $p < 0.05$.

Metric	Baselines								Our approach		
	Reg.	LambdaMART				Linear					
	RD	MV	Av	DS	GLAD	**W**	**T**	**W+T**	**W**	**T**	**W+T**
ΔDCG@1	-14.69%	0.00%	-0.79%	0.12%	-1.19%	-7.55%	-5.99%	-4.92%	-1.31%	†6.40%	†**6.98%**
ΔDCG@5	-6.92%	0.00%	0.01%	0.27%	-0.55%	-2.46%	-1.44%	-0.83%	†1.36%	†**5.41%**	†5.10%
ΔDCG@10	-5.39%	0.00%	0.21%	0.34%	-0.34%	-1.60%	-0.41%	-0.32%	-0.09%	†4.41%	†**4.56%**

daMART baselines (see below), we use the **xgboost**[3] implementation of the gradient boosted ensemble of decision trees. We implement the state-of-the-art *listwise* algorithm LambdaMART (Burges et al., 2011) and train it on $\mathcal{X}$.

We use the following methods to generate training targets as baselines:

1. MV, assigning the majority vote to every query-document pair;
2. Av, assigning the average of 3 crowd labels to every query-document pair;

3-4. DS/GLAD, assigning the most likely label to each task according to DS/GLAD model;

Besides these baselines, we experiment with the classical noise reduction technique, *reweighting by deviations* (Frénay & Verleysen, 2014, §3.3.2), denoted by RD. This reweighting method assigns to the i-th sample the weight $\min(1/\delta^2, 1/\Delta_i^2)$, where $\Delta_i = |l_i - \widehat{l_i}|$ is the absolute error of the background learning algorithm on the initial dataset $\mathcal{X}$ and $\delta \geq 0$ is a parameter. The design of LambdaMART method does not expect documents to have weights, so instead we train **xgboost** with RD weights in the standard regression regime.

Finally, we use the method for weight training proposed in (Ustinovskiy et al., 2015) in the context of click prediction problem. Following the scheme designed in the present paper (Section 4), we also extended the method from (Ustinovskiy et al., 2015) to the target training. The core difference of this baseline from our approach is that the weight and target functions are trained as (sigmoid transforms of) *linear* combinations of label features, while we train ensembles of decision trees. Below this baseline is referred as *Linear*. The background algorithm for this method is GLS.

Results We evaluate all the baselines together with our approach on $\mathcal{V}^{\text{test}}$. We consider three versions of our framework: (1) **W** which trains weight function w in Algorithm 1; (2) **T** which trains only function t; (3) their combination **W+T** which tunes both w and t. Similarly we compute three versions of the *Linear* baseline.

[3]https://github.com/dmlc/xgboost

Table 5. Feature importances.

W		***T***	
$\text{P}^\text{G}(\text{cl}^{(\text{w})} = 1)$	0.7343	$\text{P}^\text{G}(\text{cl}^{(\text{w})} = 1)$	0.6557
$\text{P}^\text{D}(\text{cl}^{(\text{w})} = 1)$	0.2474	$\text{P}^\text{D}(\text{cl}^{(\text{w})} = 1)$	0.3327
$\log(n^{(w)})$	0.0110	$a^{(w)}$	0.0104
$f_{\text{hp}}^{(w)}$	0.0023	$\pi_{12}^{(w)}$	0.0006
$\pi_{22}^{(w)}$	0.0023	$\text{P}^\text{G}(cl^{(w)} = t_i)$	0.0005

Relative improvements of our methods over the majority vote baseline (MV) are shown in Table 4. Approach **W** is significantly outperformed by **T**, and the latter is on par with their combination **W+T**. Note that according to all three evaluation measures, **T** and **W+T** statistically significantly (with $p < 0.05$) outperform the best performing baseline, namely, LambdaMART trained on DS consensus labels.

We also report the importance of the label features. To compute it we sum Gini importances over all J trees in our model for $t(\cdot)$ or $w(\cdot)$ (normalized to the unit sum for convenience). Top-5 performing label features together with their importances for $\boldsymbol{W}$ and $\boldsymbol{T}$ methods are reported in Table 5; superscripts D/G refer to DS/GLAG model.

8. Conclusion

In this paper we address the problem of supervised machine learning in the presence of label noise. Unlike the existing approaches, we explicitly utilize *label features* directly optimizing the quality of the trained model on a validation dataset. Its success prompts one to collect as much information about each sample in the training dataset as possible. By transforming this information into a large label features vector $\boldsymbol{y}$, we are able to learn more refined weights and relevance labels. These, in turn, being fed to a background machine learning algorithm, will most likely lead to a better model.

Our framework works for any smooth loss function, so in the future we intend to apply our methodology to other machine learning problems, e.g., regression and classification problems. Also it is interesting to extend our framework to more complicated background algorithms, than GLS model, e.g., to neural networks.

References

Aitken, A. C. On least squares and linear combination of observations. *Proceedings of the Royal Society of Edinburgh*, 55:42–48, 1935.

Breiman, Leo, Friedman, Jerome, Stone, Charles J, and Olshen, Richard A. *Classification and regression trees*. CRC press, 1984.

Brodley, Carla E. and Friedl, Mark A. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, pp. 131–167, 1999.

Burges, Christopher, Rango, Roberto, and Viet Le, Quoc. Learning to rank with nonsmooth cost functions. *Proceedings of the Advances in Neural Information Processing Systems*, 19:193–200, 2007.

Burges, Christopher, Svore, Krysta, Bennett, Paul, Pastusiak, Andrzej, Wu, Qiang, Chapelle, Olivier, Chang, Yi, and yan Liu, Tie. Learning to rank using an ensemble of lambda-gradient models. JMLR, pp. 25–35, 2011.

Burges, Christopher J. C. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010.

Ciaramita, Massimiliano, Murdock, Vanessa, and Plachouras, Vassilis. Online learning from click data for sponsored search. In *Proceedings of the 17th international conference on World Wide Web*, pp. 227–236. ACM, 2008.

Dawid, Alexander Philip and Skene, Allan M. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, pp. 20–28, 1979.

Frénay, Benoît and Verleysen, Michel. Classification in the presence of label noise: a survey. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(5):845–869, 2014.

Friedman, Jerome H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Graepel, Thore, Candela, Joaquin Q, Borchert, Thomas, and Herbrich, Ralf. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 13–20, 2010.

Hastie, Trevor J., Tibshirani, Robert John, and Friedman, Jerome H. *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics. 2009.

Järvelin, Kalervo and Kekäläinen, Jaana. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Lawrence, Neil D and Schölkopf, B. Estimating a kernel fisher discriminant in the presence of label noise. Citeseer, 2001.

Lee, Kyumin, Caverlee, James, and Webb, Steve. The social honeypot project: Protecting online communities from spammers. WWW '10, pp. 1139–1140. ACM, 2010.

Li, Yunlei, Wessels, Lodewyk FA, de Ridder, Dick, and Reinders, Marcel JT. Classification in the presence of class noise using a probabilistic kernel fisher method. *Pattern Recognition*, 40(12):3349–3357, 2007.

Maclaurin, Dougal, Duvenaud, David, and Adams, Ryan P. Gradient-based hyperparameter optimization through reversible learning. *arXiv preprint arXiv:1502.03492*, 2015.

Muhammadi, Jafar, Rabiee, Hamid Reza, and Hosseini, Abbas. Crowd labeling: a survey. *arXiv preprint arXiv:1301.2774*, 2013.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Sakai, Tetsuya. *Bridging Between Information Retrieval and Databases: PROMISE Winter School 2013, Bressanone, Italy, February 4-8, 2013. Revised Tutorial Lectures*, chapter Metrics, Statistics, Tests, pp. 116–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

Shen, Xuehua, Tan, Bin, and Zhai, ChengXiang. Context-sensitive information retrieval using implicit feedback. SIGIR '05, pp. 43–50, New York, NY, USA, 2005. ACM. ISBN 1-59593-034-5.

Strutz, Tilo. Data fitting and uncertainty. *A practical introduction to weighted least squares and beyond. Vieweg+ Teubner*, 2010.

Taylor, Michael, Guiver, John, Robertson, Stephen, and Minka, Tom. Softrank: Optimizing non-smooth rank metrics. WSDM '08, pp. 77–86, 2008.

Ustinovskiy, Yury, Gusev, Gleb, and Serdyukov, Pavel. An optimization framework for weighting implicit relevance labels for personalized web search. WWW '15, pp. 1144–1154, 2015.

Verbaeten, Sofie and Van Assche, Anneleen. Ensemble methods for noise elimination in classification problems. In *Multiple classifier systems*, pp. 317–325. Springer, 2003.

Whitehill, Jacob, Wu, Ting-fan, Bergsma, Jacob, Movellan, Javier R, and Ruvolo, Paul L. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. NIPS '09, pp. 2035–2043, 2009.

Zhou, Dengyong, Liu, Qiang, Platt, John, and Meek, Christopher. Aggregating ordinal labels from crowds by minimax conditional entropy. ICML '14, pp. 262–270, 2014.