Scaling up the learning-from-crowds GLAD algorithm using instance-difficulty clustering

Enrique G. Rodrigo, Juan A. Aledo, Jose A. Gámez

Escuela Superior de Ingeniería Informática.

Universidad de Castilla-La Mancha. Albacete Spain.

{Enrique.GRodrigo, JuanAngel.Aledo, Jose.Gamez}@uclm.es

Abstract

The main goal of this article is to improve the results obtained by the GLAD algorithm in cases with large data. This algorithm is able to learn from instances labeled by multiple annotators taking into account both the quality of the annotators and the difficulty of the instances. Despite its many advantages, this study shows that GLAD does not scale well when dealing with large number of instances, as it estimates one parameter per instance of the dataset. Clustering is an alternative to reduce the number of parameters to be estimated, making the learning process more efficient. However, as the features of crowdsourced datasets are not usually available, classical clustering procedures can not be applied directly. To solve this issue, we propose using clustering from vectors created by matrix factorization. Our analysis shows that this clustering process improves the results obtained by GLAD both regarding accuracy and execution time, especially in large data scenarios. We also compare this approach against other algorithms with a similar goal.

Keywords: non-standard classification · crowdsourcing · multiple annotators · weakly supervised

This is the accepted version of:

Enrique González Rodrigo, Juan A. Aledo, Jose A. Gámez.
Scaling up the learning-from-crowds GLAD algorithm using
instance-difficulty clustering. Progress in Artificial Intelligence 8,
389399 (2019). <https://doi.org/10.1007/s13748-019-00189-9>

Please, visit the provided url to obtain the published version.

Table 1: Dataset of annotations. Not every annotator labels all the instances.

Annotator ₁	Annotator ₂	...	Annotator _N
0	0	...	1
1	1	...	-
0	-	...	0
1	0	...	1
1	-	...	1
...	...	...	...

1 Introduction

Among the non-standard classification problems [8, 2], learning from instances labeled by multiple annotators (learning from crowds) is a recent topic of increasing interest. It differs from the typical classification scenario in that the instances for training the models lack a reliable ground truth. Instead, a set of annotators of unknown expertise label each instance, producing a set of unreliable labels. Not every annotator labels all the instances so this set is usually relatively sparse. An example of a dataset with such characteristics can be found in Table 1. Besides the annotations, it is sometimes possible to obtain features for each instance. However, there are few algorithms that take them into account [16] in comparison with those that tackle the label aggregation process separately [24].

Treating this kind of data is interesting in several areas such as Bioinformatics, where the methods for obtaining a reliable ground truth are sometimes very expensive. In these cases, a group of people can be asked to give their opinion about a specific problem (for example, to categorize cells in a medical image) and then these opinions must be aggregated in a way that takes into account the expertise of the labelers in the particular task [16]. Datasets labeled by multiple annotators also appear in the area of Natural Language Processing, where a large number of instances are required and platforms such as Amazon Mechanical Turk are used [6]. This kind of platforms make the data acquisition process relatively inexpensive and, even more importantly, very efficient, as the user can hire a large number of people to label the data. In scenarios with large amounts of unlabeled data, these platforms offer a clear advantage. For this reason, developing scalable methods that are able to tackle this kind of data is essential.

One of the main problems in this area is the estimation of a reliable ground truth from the available annotations [23]. The most straightforward approach to estimate the true class from annotations is to use the most frequent class, usually known as MajorityVoting. However, as the quality and experience of the annotators is normally unknown, aggregating labels under the assumption that every annotator is equally reliable may not be a good option, especially in cases

where the number of annotations per instance is limited. Different methods exist to estimate not only the labels but also the annotator quality and even the difficulties of the instances given only the set of annotations [4, 20].

The GLAD (Generative model of Labels, Abilities and Difficulties) model [20] can be used to estimate the ground truth from multiple binary annotations taking into account not only the reliability of the annotators, but also the difficulty of the instances, which can be interesting in problems where a certain expertise is important. However, GLAD has several shortcomings, especially when dealing with large datasets. As this kind of techniques are especially important when dealing with large datasets, it is important to take into account both the accuracy of the estimation and the scalability of the methods, so that they can be used in a wide range of problems.

This article extends a previous work [7], where an approximation for using the GLAD algorithm with large datasets was introduced. In the current article, we build upon the previous results, both improving the proposed algorithm as well as extending the experimental study. Regarding the method, the algorithm presented in this paper, CGLAD-2, improves the results obtained by the previous version of CGLAD (from now on, CGLAD-1) [7], in terms of accuracy and execution time for both small and large datasets (the reader may refer to the main differences between CGLAD-1 and CGLAD-2 in Section 4.6). In the case of the experimentation, we use a new method to simulate datasets and also introduce a new real dataset. Moreover, we extend the previous analysis with a study of the scalability of the methods as the number of computational resources increases.

The current paper is organised in five sections. First, Section 2 exposes recent work related to this paper. After that, Section 3 introduces the GLAD algorithm and analyzes its scalability problems. Next, Section 4 presents the new version of the CGLAD algorithm, showing the improvements in terms of scalability in comparison with the GLAD algorithm. Section 5 compares both algorithms in terms of accuracy, execution time and scalability properties. CGLAD-2 algorithm is also compared to other crowdsourcing algorithms. Lastly, Section 6 includes the conclusions of our work.

2 Related work

Ground truth inference from multiple annotators plays a key role for a great number of crowdsourcing applications [6]. The most direct approach for solving this problem is using a majority vote (the mode for discrete target variables and the mean for continuous problems). Although this simple approach seems to work well when enough labels are provided [18], several works try to improve upon this approach [24] by estimating parameters related to the labeling process, especially the reliability of the annotators.

Some works tackle the problem using optimization, defining a function to be optimized and an iterative procedure to improve upon the solution. This is the case of the PM algorithm [1, 13], which uses a parameter for each annotator

to represent their quality, or the CATD algorithm [12], which also takes into account the confidence on the estimation of the annotator reliability parameter (similar to PM).

Other methods deal with the problem using probabilistic graphical models. Depending on how they model annotators, two main approaches can be identified: one reliability parameter for each annotator or a confusion matrix for each annotator. In the first category, the works [5, 20, 14, 10, 7] tackle the problem by modelling each annotator with the probability of classifying an instance correctly. Thus, a high parameter value implies a more reliable annotator. This is the approach followed in [5] and also in [10, 14] as a variational inference problem. The GLAD algorithm [20] and our improved proposal of it tackle the worker reliability in a similar way but extend it to model the difficulty of the instances.

In the second category, the works [4, 16, 11, 19] approach the representation of reliability with a confusion matrix. The first model using this approximation was proposed by Dawid and Skene [4]. This method uses the EM algorithm to estimate both the ground truth and the confusion matrices for the annotators only using their annotations. This approach is extended in [16], where the authors incorporate the features of the dataset in the learning process and also propose a version for continuous target variables. On the other hand, in [11] the authors extend these ideas, proposing a different optimization goal and the Gibbs Sampling algorithm. This same model is extended in [19] to support annotator communities.

Although studies mainly focus on the single label problem, there are also some studies that try to perform ground truth inference in different problem settings [22, 3]. In [22], the authors propose a model for dealing with multiple label problems in the context of crowdsourcing. Regarding ranking aggregation, the authors in [3] proposed a method to inferring the consensus ranking taking into account the annotator capabilities.

3 GLAD algorithm

The GLAD algorithm [20] estimates the true labels for instances labeled by multiple annotators. In the process, it also estimates both the quality of these annotators and the difficulty of the instances, unlike many other algorithms such as the Dawid-Skene algorithm [4], the Raykar’s algorithms [16] or the PM algorithm [13]. In this section, we introduce GLAD and its scalability problems.

3.1 Model

Apart from the true label, the annotation model for the GLAD algorithm depends on two elements:

- **Difficulty of each example**, represented as $1/\beta_i \in [0, \infty)$, for each example i , where β_i is a positive real number. A value of $1/\beta_i$ for an

instance near to 0 means that the instance is easy to label. By contrast, a high value of $1/\beta_i$ means that the example is so difficult that even experienced annotators have only a 50% to label the instance correctly.

- **Annotator quality**, represented as $\alpha_j \in (-\infty, \infty)$, for each annotator j . If an annotator has a value of α_j that tends to ∞ , the annotator will tend to always label the instances correctly. If the value of α_j tends to $-\infty$ the annotator will tend to label all the instances incorrectly. This means that the annotator is as good as the previous one distinguishing between classes, but he is flipping the annotations (he may be adversarial or could have misunderstood the instructions). If the value of α_j is close to 0, the annotator cannot distinguish between the classes (random annotator or spammer).

The generative annotation model uses the last two parameters and obtains the probability that an annotator labels an instance correctly, using the following expression

$$c_{ji} = p(y_i^j = y_i | \alpha_j, \beta_i) = \frac{1}{1 + e^{-\alpha_j \beta_i}}$$

where y_i^j is the label given by the annotator j to the instance i , and y_i is the true label for the instance. Therefore, more experienced annotators (with high α_j value) have a higher probability of labeling correctly. Likewise, as the difficulty of an instance, $1/\beta_i$, increases, the probability of labeling correctly an instance approaches 0.5. This is also the case if the value of α_j for an annotator is close to 0.

3.2 Inference

To be able to estimate the ground truth for an instance, the algorithm needs to estimate the annotator quality as well as the difficulty for each instance. However, to estimate these variables, it needs an estimation of the ground truth labels. This leads to the use of the Expectation-Maximization (EM) algorithm, an algorithm able to estimate hidden values iteratively. For the GLAD algorithm, we have that the observable variables are the labels provided by the annotators, while the hidden variables are the true labels, the parameters for the annotator (α) and the difficulty (β) of the instances. The EM algorithm can obtain a maximum likelihood estimate of the values of the hidden variables. First, the algorithm obtains the true label estimations by MajorityVoting. Then, these estimations (M step) are used to learn the parameters of the model (α and β). Lastly the new parameters are used to estimate the true labels of the examples (E Step).

- **E-step:** We name $\mathcal{Y}_i = \{y_i^j\}$ the set of annotations for the instance i (not every annotator labels all the instances). If we know α , β and $\mathcal{Y}_i$, we can determine the probability of the true label y_i using the following

expression:

$$p_i^k = (y_i = k | \mathcal{Y}_i, \boldsymbol{\alpha}, \boldsymbol{\beta}) \propto p(y_i = k) \prod_j s_{ji}^k$$

where k can be 0 or 1 (as the algorithm is defined for binary problems) and

$$\begin{aligned} s_{ji}^k &= p(y_i^j = k | y_i, \alpha_j, \beta_i) \\ &= \begin{cases} (c_{ji})^k \cdot (1 - c_{ji})^{1-k} & \text{if } y_i = 1 \\ (c_{ji})^{1-k} \cdot (1 - c_{ji})^k & \text{if } y_i = 0 \end{cases} \end{aligned}$$

- **M-step:** The expression below is maximized using gradient descent with respect to the values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

$$Q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_i E_k[\ln(p(y_i = k))] + \sum_{ij} E_k[\ln(s_{ji}^k)],$$

where E_k is the expected value over k with respect to the estimations from the previous step.

The following gradients can be obtained from the expression above

$$\begin{aligned} \frac{\partial Q}{\partial \alpha_j} &= \sum_i (p_i^1 y_i^j + p_i^0 (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \beta_i, \\ \frac{\partial Q}{\partial \beta_i} &= \sum_j (p_i^1 y_i^j + p_i^0 (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \alpha_j, \end{aligned}$$

where $\sigma(x) = 1/(1 + e^{-x})$.

The weights for each class are obtained using

$$p(y_i = k) = \sum_i \frac{p_i^k}{N}$$

where N is the number of instances of the dataset.

3.3 GLAD pseudocode

We can summarize the steps needed for training the GLAD model by the pseudocode in Algorithm 1.

First, the algorithm uses the most frequent class as the initial value for the true label of each instance. Next, it iterates through the EM steps, estimating the parameters of the model in the M-step and obtaining the true label again for each instance in the E-step. This algorithm stops when it reaches a fixed number of EM iterations or when the likelihood function converges.

Algorithm 1 GLAD algorithm

- 1: **Initialization (initial E-step)**: Majority vote aggregation $\rightarrow y_i$
- 2: **for** $i = 0$ until convergence **do**
- 3: **M-step**
- 4: Optimize through gradient descent using:

$$\frac{\partial Q}{\partial \alpha_j} = \sum_i (p_{ji}^1 y_i^j + p_{ji}^0 (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \beta_i$$

$$\frac{\partial Q}{\partial \beta_i} = \sum_j (p_{ji}^1 y_i^j + p_{ji}^0 (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \alpha_j$$

- 5: Obtain new weights using:

$$p(y_i = k) = \sum_i \frac{p_i^k}{N}$$

- 6: **E-step**: Estimate class probabilities for each instance:

$$p(y_i = k | \mathcal{Y}_i, \alpha, \beta) \propto p(y_i = k) \prod_j p(y_i^j = k | y_i, \alpha_j, \beta_i)$$

- 7: **end for**
-

3.4 Scalability problems

In this section, we provide results of the execution of the algorithm as the number of instances of synthetic datasets increases. We also test different values for the learning rate of the gradient descent algorithm. This parameter seems to be closely related to the ability of the algorithm to converge, especially if we keep unaltered the gradient descent convergence threshold and the maximum number of gradient descent iterations. For the experimentation, we used synthetic datasets with the following features:

- *Dataset sizes.* We generated datasets of 6 sizes (1-6): 10K, 50K, 100K, 500K, 1M and 5M instances. For each of these sizes, we generated the real ground truth label (0 or 1) as well as a difficulty parameter (0, 1, 2), that represents how difficult to classify an instance is, with 0 being the easiest case and 2 the hardest case. For the ground truth label, we used a uniform probability, so that both classes are equally likely. For the difficulty parameter we used a 40% probability for the easiest case and a 30% for the other two cases.
- *Annotations.* For each of the instances generated, we simulated 8 annotations depending on the ground truth and the difficulty of the examples. We used four kinds of annotators: *verygood*, *good*, *regular*, *random*. For each of these types of annotator, we generated 2 values following a probability table that is different for each type. The probability table for each type of annotator can be found in Figure 1.

Class	Difficulty	Negative	Positive
Negative	0	0.8	0.2
Negative	1	0.75	0.25
Negative	2	0.7	0.3
Positive	0	0.2	0.8
Positive	1	0.25	0.75
Positive	2	0.3	0.7

(a) *verygood*

Class	Difficulty	Negative	Positive
Negative	0	0.8	0.2
Negative	1	0.7	0.3
Negative	2	0.6	0.4
Positive	0	0.2	0.8
Positive	1	0.3	0.7
Positive	2	0.4	0.6

(b) *good*

Class	Difficulty	Negative	Positive
Negative	0	0.8	0.2
Negative	1	0.6	0.4
Negative	2	0.5	0.5
Positive	0	0.2	0.8
Positive	1	0.4	0.6
Positive	2	0.5	0.5

(c) *regular*

Class	Difficulty	Negative	Positive
Negative	0	0.5	0.5
Negative	1	0.5	0.5
Negative	2	0.5	0.5
Positive	0	0.5	0.5
Positive	1	0.5	0.5
Positive	2	0.5	0.5

(d) *random*

Figure 1: Annotator models

- *Seed.* For each of the sizes, we generate 3 datasets (A, B and C) with the same annotator configuration but different simulation seed. As a result, we obtained 15 binary datasets. We named them using the pattern **binary[Letter][Number]**, so that the dataset with name **binaryA3** is a simulated binary dataset of 100k instances and seed A.

For each of these datasets, we tried four different learning rates: 0.1, 0.01, 0.001 and 0.0001. We obtained the accuracy score by the model, comparing with the true label data (since the datasets are synthetic, the true labels for the examples are available), and the execution time. The results are summarized in Figure 2. We can observe that as the number of instances increases, the method fails to provide an accurate solution. As the learning rate decreases, the algorithm is able to work with bigger datasets. However, the execution time needed to deal with the instances of highest sizes significantly increases, and the performance obtained by the algorithm in datasets of high size is still not competitive.

(a) *Accuracy. It is obtained comparing the estimated ground truth against the true label for the instance*

(b) *Execution time obtained by the algorithm measured in seconds*

Figure 2: GLAD scalability in terms of accuracy and execution time

The main reason for this problem is the increase in the number of parameters that have to be estimated. As all the datasets keep the same number of annotators, the number of parameters to be estimated are the same for all the datasets. However, as the number of instances of a dataset increases, the number of difficulty parameters increases linearly, since GLAD needs one parameter for each instance. As derived from Figure 2, the gradient descent algorithm manages to converge for datasets with a small number of instances but fails when dealing with the largest datasets. This can be seen both in the accuracy obtained by the algorithm, as well as in the execution time, where failing to optimize such a large number of parameters provokes an early stop of the learning procedure, which leads to a poor result. Therefore, it seems necessary to reduce the number of difficulty parameters to obtain a more stable algorithm and, mainly, to be able to use this method with massive datasets.

4 CGLAD-2 Algorithm

In this section we describe the CGLAD-2 algorithm¹, an improvement over the GLAD algorithm to reduce the number of parameters of the model. For this, we add an initial step previous to the EM algorithm that obtains clusters of similar instances, and then the difficulty of those clusters is used in the EM process. We explain the details of the full algorithm next.

4.1 Model

The model uses the same idea as GLAD, but it estimates difficulties for instance clusters instead of for every instance.

- **Cluster difficulty**, modeled using a parameter $1/\beta_t \in [0, \infty)$ for each cluster t , where β_t is a positive real number. As was the case with GLAD, if the value of $1/\beta_t$ for a cluster is close to 0, the instances in *cluster* t are easier. If the value for $1/\beta_t$ tends to ∞ , the instances in cluster t are more complex.
- **Annotator quality**. It is represented as $\alpha_j \in (-\infty, \infty)$ for each annotator j . As it was the case in the GLAD algorithm, if an annotator has a value of α_j that tends to ∞ , the annotator tends to always label the instances correctly while if it tends to $-\infty$ the annotator always labels the instances incorrectly (but is able to distinguish between the classes reliably). If the value of α_j is close to 0, the annotator cannot distinguish between the classes (random annotator or spammer).

The annotation model is similar to GLAD, but it uses difficulty parameters for each cluster. If we denote as $\phi(i)$ the function that maps the instance i to

¹The reader can find an implementation of this algorithm and several others in the Apache Spark package *spark-crowd* [17]

one of the clusters, the annotation model is

$$c_{ji} = p(y_i^j = y_i | \alpha_j, \beta_{\phi(i)}) = \frac{1}{1 + e^{-\alpha_j \beta_{\phi(i)}}}$$

where y_i^j is the label given by annotator j to instance i , and y_i is the true label of the instance. Except for the use of difficulties for each cluster, the interpretation of the expression is the same as in GLAD.

4.2 Inference

We use the EM approach to estimate the parameters of the model (before using EM, the cluster mapping is known).

- **E-step:** We call $\mathcal{Y}_i = \{y_i^j\}$ the set of annotations for the instance i (not every annotator labels all the instances). If we know α , β and $\mathcal{Y}_i$, we can determine the probability of the true label y_i using the following expression:

$$s_i^k = p(y_i = k | \mathcal{Y}_i, \alpha_j, \beta_{\phi(i)}) \propto p(y_i = k) \prod_j p_{ji}^k \quad (1)$$

where

$$\begin{aligned} s_{ji}^k &= p(y_i^j = k | y_i, \alpha_j, \beta_{\phi(i)}) \\ &= \begin{cases} (c_{ji})^k \cdot (1 - c_{ji})^{1-k} & \text{if } y_i = 1 \\ (c_{ji})^{1-k} \cdot (1 - c_{ji})^k & \text{if } y_i = 0 \end{cases} \end{aligned}$$

- **M-step:** The expression below is maximized using gradient descent with respect to the values of α and β .

$$Q(\alpha, \beta) = \sum_i E_k[\ln(p(y_i = k))] + \sum_{ij} E_k[\ln(p_{ji}^k)]. \quad (2)$$

where E_k is the expected value over k with respect to the estimations of the previous step. For this, we use the stochastic gradient descent implementation provided with Spark MLlib (in CGLAD-1 [7], we use a standard gradient descent procedure). This is impossible in the case of GLAD algorithm as the number of parameters to estimate is too high and needs to send these parameters over the network. The following gradients can be obtained from the expression above

$$\begin{aligned} \frac{\partial Q}{\partial \alpha_j} &= \sum_i (p_i^1 y_i^j + p_i^0 (1 - y_i^j) - \sigma(\alpha_j \beta_{\phi(i)})) \beta_{\phi(i)}, \\ \frac{\partial Q}{\partial \beta_t} &= \sum_i \delta(\phi(i), t) \sum_j (p_i^1 y_i^j + p_i^0 (1 - y_i^j) - \sigma(\alpha_j \beta_t)) \alpha_j, \end{aligned}$$

where $\delta(\phi(i), t) = 1$ if $\phi(i) = t$ and 0 otherwise. The weights for each class must also be obtained using

$$w_k = p(y_i = k) = \sum_i \frac{p_i^k}{N} \quad (3)$$

where N is the number of instances of the dataset. In comparison with CGLAD-1 [7] we use a multiple restart mechanism. If the optimization algorithm is not able to converge, the learning rate is reduced by an increasing reduction factor (2,3,4...) until it reaches a minimum threshold, when the algorithm will finish.

4.3 Initialization

CGLAD-2 proposes a special initialization that not only obtains a first estimation for the true labels (using majority voting) but also divides the instances into clusters, using matrix factorization and K-Means. This combination of methods allows the algorithm to obtain clusters of similar instances even when the features for the dataset are not available, only using the annotations.

- **Matrix factorization of the annotations dataset.** As occurs in collaborative filtering, we can visualize the dataset of annotations as a matrix where annotators are represented as rows and instances are represented as columns. Matrix factorization provides us with two matrices that allow us to estimate the matrix of annotations. This method gives us vectors of size R (a parameter that must be set by the user) for each instance (and also for the annotators) that capture differences between instances. For this step we used ALS [25] that can be used at scale. After the factorization, we obtain matrices A and D , that represent annotators and instances respectively.
- **Clustering.** Once the vectors are estimated, we can apply whichever clustering algorithm over D to obtain instance groups. In this proposal, we use K-Means (with Euclidian distances), although other methods may also be applicable [21]. The selection of the number of clusters depends of the problem to solve, but as they do not need to be interpreted, a higher number can be used. As K-Means depends on the initial position of centroids, we perform several restarts of the clustering process (we used five restarts for the experimentation). From all the restarts, the algorithm chooses the best number of clusters using the Silhouette measure, although other alternative measures can be considered [15]. This process of multiple restarts of the clustering method is another difference from the method presented in [7].

Once the previous steps are completed, we obtain the function ϕ , which is the last component needed to use the algorithm.

4.4 CGLAD-2 pseudocode

The pseudocode of the CGLAD-2 algorithm is divided in three functions: Initialization, which initializes the algorithm obtaining the clusters and the first ground truth estimation; Step, which performs a EM optimization iteration; and CGLAD, the main function. First, the initialization function for the algorithm is shown in Algorithm 2. It receives 2 parameters, the annotation function and the number of restarts of the K-Means algorithm that need to be executed (RM). First, the method estimates the vectors needed for the clustering procedure from the annotations $\mathcal{Y}_i$. Then, it obtains the clusters from those vectors, after performing a number of restarts. Lastly, it estimates the ground truth using the most frequent class.

Algorithm 2 Initialization function

```

1: function INITIALIZATION( $\mathcal{Y}_i$ , RM)
2:    $D \leftarrow \text{ALS}(\mathcal{Y}_i)$ 
3:   BestSilh  $\leftarrow -1$ 
4:   BestPhi  $\leftarrow$ 
5:   for  $j = 0$  until RM do
6:      $\phi' \leftarrow \text{KMeans}(D, \text{seed} = j)$ 
7:     if SilhouetteM( $\phi'$ ) > BestSilh then
8:       BestSilh  $\leftarrow \text{SilhouetteM}(\phi')$ 
9:       BestPhi  $\leftarrow \phi'$ 
10:    end if
11:  end for
12:   $\phi \leftarrow \text{BestPhi}$ 
13:   $y_i \leftarrow \text{MajorityVote}(\mathcal{Y}_i)$ 
14:  return  $\phi, y_i$ 
15: end function

```

The Step function is shown in Algorithm 3. It accepts three parameters, the annotations set, the ground truth estimation (given by a previous EM step or by the initialization process), the stochastic gradient descent learning rate (RM) and the cluster mapping (ϕ). It starts optimizing the parameters for the annotator and difficulty models, as well as the class priors. Using these parameters, a new estimation of the ground truth is obtained. Then, all these parameters and the negative log-likelihood are returned.

Algorithm 3 EM Step function

```

1: function STEP( $\mathcal{Y}_i, y_i, \text{LR}, \phi$ )
2:    $\alpha, \beta \leftarrow \text{StochasticGradientDescent}(\mathcal{Y}_i, y_i, \text{LR}, \phi)$  2
3:    $w_k \leftarrow \text{WeightEstimation}(y_i)$  3
4:    $y_i \leftarrow \text{GroundTruthEstimation}(\mathcal{Y}_i, \alpha, \beta, \phi)$  1
5:   NegLogLike  $\leftarrow \text{NegativeLogLikelihood}(\mathcal{Y}_i, y_i, \alpha, \beta, \phi)$ 
6:   return  $y_k, \alpha, \beta, w_k, \text{NegLogLike}$ 
7: end function

```

Lastly, the main function of the algorithm is shown in 4. It accepts as pa-

rameters the annotations set, $\mathcal{Y}_i$, the initial learning rate (LR), the number of EM iterations to perform (EMIter), the restart threshold (RT) and the number of restarts for K-Means (RM). The restart threshold limits the minimum number that the learning rate can be reduced to. As this threshold decreases, the number of restarts for the optimization increases. At the beginning, the algorithm is initialized, using the Initialization function. Next, the initial values for the reduction term and the previous negative log-likelihood (to compare if the EM step improved the solution) are set. Then, the algorithm performs several iterations of the EM method, using the Step function. After each step, the algorithm checks if the solution has improved and, if not, reduces the learning rate using the reduction term. The method finishes if it reaches the desired EM iterations or if the learning rate reaches a minimum value.

Algorithm 4 CGLAD algorithm

```

1: function CGLAD( $\mathcal{Y}_i$ , LR, EMIter, RT, RM)
2:   reduction  $\leftarrow$  2
3:    $\phi, y_i \leftarrow$  INITIALIZATION( $\mathcal{Y}_i$ , RM)
4:   NLL  $\leftarrow$   $\infty$ 
5:   for  $i = 0$  until EMIter do
6:      $r \leftarrow 0$ 
7:     improvement  $\leftarrow$  true
8:     while (not improvement) do
9:        $y'_k, \alpha', \beta', w'_k, \text{NLL}' \leftarrow$  STEP( $\mathcal{Y}_i, y_i, \text{LR}, \phi$ )
10:      if  $\text{NLL}' > \text{NLL}$  then
11:         $\text{LR} \leftarrow \frac{\text{LR}}{\text{reduction}}$ 
12:        reduction  $\leftarrow$  reduction + 1
13:        improvement  $\leftarrow$  false
14:      else
15:        improvement  $\leftarrow$  true
16:         $y_k, \alpha, \beta, w_k, \text{NLL} \leftarrow y'_k, \alpha', \beta', w'_k, \text{NLL}'$ 
17:      end if
18:      if  $\text{LR} < \text{RT}$  then
19:        return  $y_k, \alpha, \beta, w_k, \text{NLL}$ 
20:      end if
21:    end while
22:  end for
23:  return  $y_k, \alpha, \beta, w_k, \text{NLL}$ 
24: end function

```

4.5 Scalability comparison

We applied CGLAD-2 to the same datasets that we used when analyzing the scalability problems of GLAD. For the comparison we used higher learning rates than in the GLAD cases, as they seem to perform better, given our analysis of the parameter optimization in Section 5.2. The results in terms of accuracy and execution time for CGLAD-2 can be found in Figure 3. In comparison with the results obtained by GLAD (Figure 2), CGLAD-2 shows a more stable performance across the datasets and learning rate values regarding accuracy. CGLAD-2 seems to scale better to bigger datasets, whereas the difference in accuracy in smaller datasets seems negligible (we show further comparisons

between the models in the next section). Regarding execution time, the one of CGLAD-2 grows, as may be expected, as the dataset increases in size. However, even for relatively small data, CGLAD-2 obtains better results in comparison to GLAD. For larger sizes, GLAD obtains a lower execution time at the expense of a far worse solution.

(a) GLAD

(b) CGLAD-2

Figure 3: GLAD against CGLAD-2 in terms of accuracy

4.6 Comparison with CGLAD-1

CGLAD-2 adds several improvements over CGLAD-1 [7]. The main ones are:

- CGLAD-2 uses the implementation of stochastic gradient descent provided by Spark MLlib. The use of this algorithm makes possible to obtain much better results in terms of execution time.
- CGLAD-2 uses multiple restarts in the clustering method. As K-Means could suffer from a bad initial clustering location, in this version we included a multiple restarts procedure. In this case, the clustering is per-

formed five times and each result is evaluated using the Silhouette measure. The best solution measured by the metric from the five restarts is the one used for the rest of the algorithm.

- Multiple restarts on the EM optimization step. As the EM progresses through the optimization, the gradient descent procedure needs a smaller learning rate to converge into a solution. In this version, the algorithm reduces the optimization step by an increasing factor when the algorithm detects that the gradient descent has diverged.

In Table 2, CGLAD-2 is compared to CGLAD-1 using the same datasets as in [7]. The results show that CGLAD-2 improves upon CGLAD-1, especially in the bigger dataset sizes. CGLAD-2 obtained similar accuracy to CGLAD-1 in the small datasets but for the the biggest cases the results obtained by CGLAD-2 far exceed the ones obtained by CGLAD-1.

Table 2: CGLAD-1 vs CGLAD-2 in terms of accuracy

Instances	Accuracy	
	CGLAD-1	CGLAD-2
5000	0.9940	0.9940
10000	0.9904	0.9948
20000	0.9941	0.9948
40000	0.9941	0.9948
80000	0.8731	0.9951
160000	0.9132	0.9951
320000	0.8879	0.9950
640000	0.8873	0.9950
1280000	0.7750	0.9931
2560000	0.8056	0.9947

5 Experimentation

In this section, we compare CGLAD-2 against GLAD and also to other algorithms in the literature with a similar goal.

5.1 Datasets

For the comparisons done in this section we use the synthetic datasets described in 3.4. and we also include a real dataset: SpamCF. It contains 11040 examples annotated by 153 annotators producing a total of 42762 annotations. For evaluation, we have the ground truth of 100 of those examples. The data is a processed version of the HITspam-UsingCrowdfower [9]. All the datasets used for the experimentation are available through: <http://bit.ly/prai2019cglad>

5.2 Parameter optimization

As CGLAD-2 has some extra parameters that need to be optimized, it is necessary to set them for the comparison in the next sections. As there is no ground truth available, we used 6 simulated datasets from the ones introduced in Section 5.1, all of them with the same simulation seed (C). The other 12 simulated datasets (seeds A and B) and the real dataset (SpamCF) are used for the experimentation in the next section. We performed a grid search over the extra parameters used by this method:

- Stochastic gradient descent data fraction: 1, 0.9, 0.8
- Stochastic gradient descent initial learning rate: 30, 10, 3, 1, 0.3
- Restart threshold in the optimization process: 1, 0.3, 0.1, 0.03
- Vector Size for the ALS algorithm: 8, 16
- Cluster size: 32, 64, 128

To decide about which configuration to choose, for each dataset we obtained the best one and then we chose the mode of the winning parametrizations. The chosen configuration was:

- Stochastic gradient descent data fraction: 1
- Stochastic gradient descent initial learning rate: 30
- Restart threshold in the optimization process: 1
- Vector Size for the ALS algorithm: 8
- Cluster size: 32

5.3 CGLAD-2 vs GLAD

First we compared CGLAD-2 against GLAD regarding accuracy and execution time in Table 3. For the comparison of execution time we used the same environment for both algorithms². Although CGLAD-2 achieved worse results in the smallest datasets (5000 instances), it obtained better results than GLAD algorithm in the bigger cases. The degradation on the results of GLAD was especially notable from the second size on. Analyzing the execution time, CGLAD-2 spent more time in bigger datasets than in smaller datasets, as should be expected. However, in the case of GLAD, the inability to converge in the optimization step was notable even in the cases with 10k instances. Due to the convergence and early stopping problem, GLAD showed an execution time in large datasets which is sometimes lower than in the smaller ones. In total, CGLAD-2 won against GLAD in 8 cases and lost in 2 cases. Other 2 cases ended in a tie.

²An Apache Spark cluster with 3 worker nodes with 10 cores and 30Gb of memory each

Table 3: GLAD vs CGLAD-2 in accuracy and execution time

Dataset	Accuracy		Execution time (s)	
	CGLAD-2	GLAD	CGLAD-2	GLAD
binaryA1	0.823	0.841	83	1604
binaryA2	0.827	0.827	154	101
binaryA3	0.823	0.647	168	136
binaryA4	0.824	0.430	320	114
binaryA5	0.824	0.463	528	128
binaryA6	0.821	0.211	3233	791
binaryB1	0.824	0.838	105	1596
binaryB2	0.822	0.822	155	138
binaryB3	0.824	0.732	159	128
binaryB4	0.824	0.812	314	82
binaryB5	0.824	0.542	621	140
binaryB6	0.810	0.459	3485	816
SpamCF	0.668	0.660	102	1604

It is also interesting to analyze the execution time regarding the scalability as the number of nodes in the Apache Spark cluster increases, using the speed-up measure: the reduction in time obtained while increasing the number of nodes in comparison with the execution time using only one node. This measure is especially important in the highest dataset sizes, as smaller datasets do not usually benefit of distributed execution (because of the parallelization costs). In Figure 4 the results for the three biggest sizes are shown ³. In spite of the iterative nature of CGLAD-2, our implementation obtained a high time reduction as the number of nodes increased, especially in the biggest case. In all the problems, CGLAD-2 obtained better results than GLAD in terms of speed up.

Figure 4: Speed-up of GLAD and CGLAD-2

³ All the simulated datasets were used for these comparison. The (aggregated) results for each data size are shown in Figure 4.

5.4 CGLAD-2 against other algorithms

There exist other algorithms capable of estimating the ground truth from annotations. However, most of them are not able to take into account the difficulties of each example, although it is common to estimate the quality of each annotator. In this section, CGLAD-2 is compared against two of these algorithms: majority voting, which could be considered as the baseline, and Dawid-Skene algorithm. The majority voting algorithm supposes that all the annotators have the same quality, so the ground truth estimation is, essentially, the mode, for discrete target variables and the mean, for continuous target variables. If we suppose that most of the annotators are providing correct information, enough annotations can lead to a correct ground truth [18]. On the other hand, Dawid-Skene algorithm tries to estimate the quality of each of the annotators so that their labels are aggregated in an optimum manner, depending on the probability of an annotator to label correctly each class. This algorithm only works for discrete target variables and it obtains great results in practice. In Table 4 CGLAD-2 is compared to these two algorithms. As can be observed, CGLAD-2 obtains a better accuracy than both Majority Voting and Dawid-Skene in all of the tested datasets. However, it is also the algorithm with the higher execution time, as CGLAD-2 involves an extra step of gradient descent, in comparison to MajorityVoting, which does not perform any optimization, or Dawid-Skene, which does not need to use gradient descent in the M step because the optimization has a closed-form expression that can be easily parallelized.

Table 4: Comparative with other methods (Accuracy and Execution time)

Dataset	Methods			Execution time (s)		
	Majority	Dawid-Skene	CGLAD-2	Majority	Dawid-Skene	CGLAD-2
binaryA1	0.777	0.806	0.823	11.0	42.0	83.0
binaryA2	0.782	0.809	0.827	15.0	54.0	154.0
binaryA3	0.781	0.807	0.823	14.5	46.0	168.0
binaryA4	0.781	0.806	0.824	16.5	97.0	320.0
binaryA5	0.781	0.806	0.824	16.5	128.0	528.0
binaryA6	0.781	0.806	0.821	34.0	354.0	3233.0
binaryB1	0.778	0.803	0.824	11.5	43.0	105.0
binaryB2	0.778	0.805	0.822	10.0	55.0	155.0
binaryB3	0.780	0.805	0.824	9.5	46.0	159.0
binaryB4	0.781	0.807	0.824	12.0	94.0	314.0
binaryB5	0.781	0.807	0.824	13.5	121.0	621.0
binaryB6	0.782	0.807	0.810	34.0	385.0	3485.0
spamCF	0.660	0.660	0.668	7.5	23.0	102.0

6 Conclusions

This article presents CGLAD-2, an algorithm which provides an improvement over both GLAD and CGLAD-1 for the problem of learning from big crowd-sourced data. The experiments show that the new proposal achieves better scalability and stability when dealing with large datasets, improving upon not only GLAD execution time but also its accuracy in both large and medium size datasets. We also compare CGLAD-2 to other state of the art algorithms: MajorityVoting and Dawid-Skene. In this sense, CGLAD-2 obtains better results in terms of accuracy for the datasets used in the experimentation. However, if a small execution time is needed, algorithms such as Dawid-Skene can be interesting, as they propose a good solution in a reduced execution time. When the execution time is not essential, CGLAD-2 obtains a better solution in a relatively small ammount of time and it is especially suitable for contexts where the difficulty of the instances is significant.

This improvement can lead to future lines of work. Both GLAD and CGLAD-2 are algorithms that can only tackle binary class problems. Extending the capabilities of the algorithms to problems with multiple classes can be an interesting line of future work. Other algorithms in the field of learning from multiple annotations are able to estimate multiple classes, such as Dawid-Skene, but they do not take into account the difficulties of the example. In our opinion, taking ideas from both algorithm types can be useful to develop a solution able to both estimate the difficulties and learn from multiple classes.

Although CGLAD-2 does not use the features of the examples, these could be helpful in the learning process and, especially, in the initial clustering process. Regarding this last point, both the information obtained from the annotations and the features of each example could be easily integrated in the initialization process of CGLAD-2.

Acknowledgements

This work has been partially funded by FEDER funds, the Spanish Research Agency (MINECO) and the Junta de Comunidades de Castilla-La Mancha (JCCM) through projects TIN2016-77902-C3-1-P and SBPLY/17/18050/000493. Enrique G. Rodrigo has also been funded by the FPU scholarship FPU15/02281 by MECD.

References

- [1] Aydin, B.I., Yilmaz, Y.S., Li, Y., Li, Q., Gao, J., Demirbas, M.: Crowd-sourcing for multiple-choice question answering. In: Twenty-Sixth IAAI Conference (2014)
- [2] Charte, D., Charte, F., García, S., Herrera, F.: A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem trans-

- formations and algorithm adaptations. *Progress in Artificial Intelligence* (2018). DOI 10.1007/s13748-018-00167-7
- [3] Chen, X., Bennett, P.N., Collins-Thompson, K., Horvitz, E.: Pairwise ranking aggregation in a crowdsourced setting. In: *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 193–202. ACM (2013)
 - [4] Dawid, A.P., Skene, A.M.: Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics* **2**, 20–28 (1979)
 - [5] Demartini, G., Difallah, D.E., Cudré-Mauroux, P.: Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In: *Proceedings of the 21st international conference on World Wide Web*, pp. 469–478. ACM (2012)
 - [6] G. Rodrigo, E., Aledo, J.A., Gámez, J.A.: Machine learning from crowds: A systematic review of its applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **0**(0), e1288. DOI 10.1002/widm.1288. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1288>
 - [7] G. Rodrigo, E., Aledo, J.A., Gámez, J.A.: CGLAD: Using GLAD in crowd-sourced large datasets. In *Lecture Notes in Computer Science*, vol 11314 (IDEAL 2018) pp. 783–791 (2018)
 - [8] Hernández-González, J., Inza, I., Lozano, J.A.: Weak supervision and other non-standard classification problems: a taxonomy. *Pattern Recognition Letters* **69**, 49–55 (2016)
 - [9] Ipeirotis, P.G., Provost, F., Wang, J.: Quality management on amazon mechanical turk. In: *Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP '10*, pp. 64–67. ACM, New York, NY, USA (2010). DOI 10.1145/1837885.1837906. URL <http://doi.acm.org/10.1145/1837885.1837906>
 - [10] Karger, D.R., Oh, S., Shah, D.: Iterative learning for reliable crowdsourcing systems. In: *Advances in neural information processing systems*, pp. 1953–1961 (2011)
 - [11] Kim, H.C., Ghahramani, Z.: Bayesian classifier combination. In: *Artificial Intelligence and Statistics*, pp. 619–627 (2012)
 - [12] Li, Q., Li, Y., Gao, J., Su, L., Zhao, B., Demirbas, M., Fan, W., Han, J.: A confidence-aware approach for truth discovery on long-tail data. *Proceedings of the VLDB Endowment* **8**(4), 425–436 (2014)
 - [13] Li, Q., Li, Y., Gao, J., Zhao, B., Fan, W., Han, J.: Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1187–1198. ACM (2014)

- [14] Liu, Q., Peng, J., Ihler, A.T.: Variational inference for crowdsourcing. In: Advances in neural information processing systems, pp. 692–700 (2012)
- [15] Luna-Romera, J.M., García-Gutiérrez, J., Martínez-Ballesteros, M., Riquelme Santos, J.C.: An approach to validity indices for clustering techniques in big data. *Progress in Artificial Intelligence* **7**(2), 81–94 (2018)
- [16] Raykar, V.C., Yu, S., Zhao, L.H., Valadez, G.H., Florin, C., Bogoni, L., Moy, L.: Learning from crowds. *Journal of Machine Learning Research* **11**(Apr), 1297–1322 (2010)
- [17] Rodrigo, E.G., Aledo, J.A., Gamez, J.A.: spark-crowd: A spark package for learning from crowdsourced big data. *Journal of Machine Learning Research* **20**(19), 1–5 (2019). URL <http://jmlr.org/papers/v20/17-743.html>
- [18] Snow, R., O’Connor, B., Jurafsky, D., Ng, A.Y.: Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In: Proceedings of the conference on empirical methods in natural language processing, pp. 254–263. Association for Computational Linguistics, Honolulu, USA (2008)
- [19] Venanzi, M., Guiver, J., Kazai, G., Kohli, P., Shokouhi, M.: Community-based bayesian aggregation models for crowdsourcing. In: Proceedings of the 23rd international conference on World wide web, pp. 155–164. ACM (2014)
- [20] Whitehill, J., Wu, T.f., Bergsma, J., Movellan, J.R., Ruvolo, P.L.: Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In: Advances in neural information processing systems, pp. 2035–2043 (2009)
- [21] Xu, R., Wunsch, D.: Survey of clustering algorithms. *IEEE Transactions on neural networks* **16**(3), 645–678 (2005)
- [22] Zhang, J., Wu, X.: Multi-label inference for crowdsourcing. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD ’18, pp. 2738–2747. ACM, New York, NY, USA (2018). DOI 10.1145/3219819.3219958. URL <http://doi.acm.org/10.1145/3219819.3219958>
- [23] Zhang, J., Wu, X., Sheng, V.S.: Learning from crowdsourced labeled data: a survey. *Artificial Intelligence Review* **46**(4), 543–576 (2016)
- [24] Zheng, Y., Li, G., Li, Y., Shan, C., Cheng, R.: Truth inference in crowd-sourcing: is the problem solved? *Proceedings of the VLDB Endowment* **10**(5), 541–552 (2017)

- [25] Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: International Conference on Algorithmic Applications in Management, pp. 337–348. Springer (2008)