Learning to classify software defects from crowds: a novel approach

Jerónimo Hernández-González[1], Daniel Rodriguez[2], Iñaki Inza[1], Rachel Harrison[3], Jose A. Lozano[1,4]

[1]Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, Donostia, Spain

[2]Department of Computer Science, University of Alcala, Madrid, Spain

[3]Department of Computering, Oxford Brookes University, Oxford, UK

[4]Basque Center for Applied Mathematics BCAM, Bilbao, Spain

Abstract

In software engineering, associating each reported defect with a category allows, among many other things, for the appropriate allocation of resources. Although this classification task can be automated using standard machine learning techniques, the categorization of defects for model training requires expert knowledge, which is not always available. To circumvent this dependency, we propose to apply the learning from crowds paradigm, where training categories are obtained from *multiple non-expert* annotators (and so may be incomplete, noisy or erroneous) and, dealing with this subjective class information, classifiers are efficiently learnt. To illustrate our proposal, we present two real applications of the IBM's orthogonal defect classification working on the issue tracking systems from two different real domains. Bayesian network classifiers learnt using two state-of-the-art methodologies from data labeled by a crowd of annotators are used to predict the category (impact) of reported software defects. The considered methodologies show enhanced performance regarding the straightforward solution (majority voting) according to different metrics. This shows the possibilities of using non-expert knowledge aggregation techniques when expert knowledge is unavailable.

Keywords: Learning from crowds, Orthogonal defect classification, Missing ground truth, Bayesian network classifiers

1 Introduction

Defect classification is an important task during software maintenance [1] which can be used to facilitate defect prioritization, faster and cheaper defect resolution, and analysis of module and component quality [2, 3, 4, 5]. It is a time-consuming task which has been traditionally performed manually by developer team members with expert knowledge of the task. Recently, machine

learning (ML) techniques, such as supervised classification, have been applied to the classification of defects [6].

Given a classification task of interest, standard supervised classification techniques infer, from a set of previously labeled examples (certainly categorized defects in our case), the mapping between examples and categories. Classifiers can thus be trained to anticipate the category of new unlabeled examples (i.e., new defects). In this paradigm, each training example describes a specific case (defect) by means of a set of features and is provided together with its real category. In software engineering, obtaining the real category of a large set of previous defects, as required by standard ML techniques, is a difficult task which needs to be carried out by a domain expert. This fact usually prevents managers from advocating ML techniques to automate the classification of defects in their projects. In the first place, every developer team does not necessarily include an expert on defect classification. And, even when an expert is available, a careful categorization (one by one) of defects may be impractical.

In this paper, we address the following research question: *can we learn to classify defects without the labeling of a domain expert?* Although no expert knowledge is available in our scenario, we do have access to a pool of computer scientists who may have partial knowledge about the task. In contrast to the concept of "expert", the term *novice* is used throughout the rest of the paper to refer to annotators without expert knowledge who provide partially reliable categorizations (possibly inaccurate and/or biased annotations). Certainly, the use of a single novice for labeling a whole dataset of defects is risky. The output of a learning process where the class information is only provided by such an annotator will likely be a classifier which reproduces their unreliable labeling behavior; that is, an inaccurate classifier. A key study of sources of error [7] showed that a straightforward solution for dealing with the problem of learning from a single novice annotator is to take into account the opinion (labeling) of *a set of* novices. This is the fundamental idea behind the learning from crowds paradigm [8, 9], which inspires the solution proposed in this paper.

For this work, two different groups of 5 novice annotators have labeled the defects reported in two real domains, the Compendium and Mozilla open-source projects. Although our approach could be applied to any classification problem in software engineering, we have selected, without loss of generality, to categorize defects based on their impact as defined by the Orthogonal Defect Classification (ODC) taxonomy [10]. The 13-category ODC taxonomy allows developers to separate defects depending on their impact on the customer. It is particularly suitable for open-source projects, where users are also commonly developers, as the impact classification will, in theory, find the defects that impact user experience the most. Thus, novices are asked to associate each defect (training example) with an impact (category). Apart from the problem of the reliability of the annotators, this application faces the issue of processing the text in which the defect has been reported, mainly written in natural language. However, this work has been carried out to illustrate the applicability of the learning from crowds paradigm to real defect classification domains in the absence of expert supervision. Thus, the natural language processing (NLP) challenge is beyond

the scope of this paper; standard techniques have been applied to deal with it.

Keeping in line with this scenario, the research question could be rephrased as: *can we learn a classification model of software defects using the impact categories provided by a set of novice annotators?* Addressing the research question, the main contribution of this paper is an in-depth analysis of two real applications (the Compendium and Mozilla projects). In order to carry out the analysis, two learning from crowds state-of-the-art techniques have been applied. On the one hand, a K-means based approach [11], which assumes the existence of common tendencies of category-confusion among the annotators, tries to find out which distributions of labels are usually associated with every category. On the other hand, an adaptation to defect classification of an Expectation-Maximization (EM) based technique [9], which follows the popular Dawid-Skene [12] strategy to infer a classification model, has been also used. This technique (i) models the subjective point of view of the different novices (reflected in significant rates of disagreement as observed in Section 3.1), (ii) estimates their reliability individually and (iii) takes it into account to learn the resulting classifiers.

The rest of the paper is organized as follows. In the next section, background definitions and related work are described. Next, the real domains on which the empirical studied is carried out and the applied methodology are presented. The experimental work is explained and discussed in Section 4. The paper concludes with conclusions and future work.

2 Background

According to the IEEE Standard 1044-1993 [13], a *defect* is "*an imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced*". In practice for each defect a report is usually generated through an issue tracking system. A defect report is a description of the issue which can be used to replicate and fix the problem. An issue tracking or bug reporting system is typically used by software project managers for reporting and tracking defects as well as proposing new functionalities, other project management tasks and infrastructure decisions and code reviews. Open source issue tracking systems include Bugzilla, Launchpad, GitHub and RedMine. Tickets are used to organize the information. Each ticket maintains data such as an identifier, summary, description, opening/closing/modification dates, reporter, priority, severity, environment, current status, etc.

The classification of software defects aims to capture the semantics of the reports of each type of defect. Software defect classification provides extra information about defects and so is valuable for many tasks such as prioritizing software defects, improvement of defect prediction, assignment of defects to developers, defect resolution, identifying the quality of components, etc. One of the most popular defect classification taxonomies is IBM's ODC, although it has been criticized due to a variety of drawbacks such as being neither fully

orthogonal nor consistent in the terminology [14]. It is said to be difficult to apply in practice [15], and complicated to customise to specific contexts [16, 17]. In a controlled experiment with students, Falessi et al. [18] also reported that there is affinity between some ODC defect types and previous training is needed to apply it. Nevertheless, IBM and other organizations have applied ODC to improve software development processes [19, 20, 21, 22].

ODC consists of four steps: (i) *classify* the data; (ii) *validate*; (iii) *assess* the ODC attributes and defect trend analysis; and (iv) *act* to implement the actions. When a defect is reported following the ODC process, three attributes have to be added: (i) *ODC activity*, such as design review, unit test, etc.; (ii) *ODC trigger*, which is the environment or condition that led to the failure; and (iii) *ODC defect impact*, which relates the impact of the software defect to customer satisfaction. As opposed to the goal of reducing the total number of defects, ODC impact can be used with severity to focus quality improvement effort on reducing the defects that most significantly impact customer satisfaction.

2.1 Related Work

There is a large amount of literature related to defect classification starting with the seminal work by Endres [2], and followed by other studies such as those by [23], [24] or [25]. Multiple models, variations and customisation of the initial taxonomies have been proposed (e.g., [26], [3], etc.). The IEEE Standard Classification for Software Anomalies [13] defines both the terminology and the process to deal with defects. Thus, reports can refer to *errors* (human mistakes), *defects* (deficiencies in a product), *faults* (issues in software) or *failures* (issue preventing normal use). It also defines the classification process and the attributes to report. In addition to ODC [10], another popular approach was developed by HP [15], where sources of defects are classified according to three axes: *origin*, *type* and *modes*. Defect classification approaches and challenges have been discussed previously [4]. Recently, a comprehensive taxonomy was proposed [27].

Typically, developers manually classify defects into the ODC categories based on the reported descriptions using, for example, root-cause defect analysis (RCA) [28, 29]. The automation of software engineering problems by means of machine learning techniques is increasingly being explored. The differentiation between defects and requirements, the importance of which has been noted [30], has already been solved making use of the reported data. Additionally, the problem of duplicate report recovery has been addressed by means of unsupervised learning techniques [31, 32]. The classification of reports during enhancement work or other activities, reaching 77% and 82% of accuracy, has been reported [33]. Recently, Zhou et al. [34] combined text mining on the defect descriptions with structured data (e.g., priority and severity) to identify *corrective* bugs.

Related to our work, Thung et al. [6] classify defects into three *super*-categories (control and data flow, structural, and non-functional) which cover all the ODC defect types. As opposed to our approach, they rely on ex-

pert knowledge to obtain the ground truth. Also, Huang et al. propose AutoODC [35], an automatic defect classification approach based on ODC to automatically categorize reports taking advantage of extra expert knowledge. Relevant words/phrases of the reports are identified and selected by experts to be used as predictive variables. In this way, accuracy gains of up to 10 percentage points are obtained. Whereas our methodology aims to obtain defect categories when experts are unavailable, AutoODC uses extra expert knowledge to enhance the set of descriptive variables.

3 Materials and Methods

3.1 Datasets

The first dataset used in this paper is composed of reports collected from the Compendium project[1], a software tool for mapping information, ideas and arguments. The issue tracking system, implemented in Bugzilla, collects support issues, feature requests and bug reports from the Compendium community.

The collected dataset comprises all the entries available in August 2014. For each of the 846 obtained defects, only the informative fields have been considered: *severity*, *summary* and *description*. *Severity* is a 3-value variable (Bug, Support or Feature), and both *summary* and *description* are text fields. Five novice annotators were asked to annotate the impact of each example, according to the descriptions of the corresponding 13-category ODC standard [10]. We found that only 9 out of the original 13 categories were used by the annotators to label the defects of the dataset. Moreover, we found high variability among annotators: some categories were assigned to less than 10 reports whereas the *usability* impact, for instance, was consistently assigned to about a third of the collected defects.

Rather than solve the classification task at hand, this paper aims to point out the viability of a learning from crowds approach when no expert supervision is available in software engineering classification problems. Dealing with the original annotations would require supplementary machine learning techniques, which are not necessarily related to the crowd learning paradigm, in order to learn from such a highly unbalanced multi-class dataset. Including these techniques would make it difficult to interpret the results and assess the contribution of the learning from crowds approach. Therefore, for the sake of simplicity, the dataset has been pre-processed to reduce the number of categories: the three majority categories (*Installability*, *Requirements* and *Usability*) have been maintained while the other annotations have been grouped in a new label, *Other*. The result is a dataset with four categories moderately balanced which aligns with the essence of crowd labeling: a large number of disagreements among annotators can be resolved by our techniques to learn trustworthy classifiers.

The second dataset has been collected from the Mozilla project, a popular open-source application which started back in the late 90s with the Netscape

[1] `http://compendium.open.ac.uk/bugzilla/`

Table 1: No. of examples assigned by each annotator to the different labels (defect impacts).

Impact	Annotator					Impact	Annotator				
	L_1	L_2	L_3	L_4	L_5		L_1	L_2	L_3	L_4	L_5
Installability	92	82	86	87	87	*Installability*	158	108	73	115	158
Requirements	192	236	139	239	242	*Maintenance*	2	140	184	36	62
Usability	392	267	473	279	353	*Reliability*	130	159	201	375	130
Other	170	261	148	241	164	*Other*	308	191	140	72	248
Compendium dataset						Mozilla dataset					

Table 2: Agreement on the assigned categories. Each cell shows the number of examples assigned to a certain category (defect impact) —row— by a subset of annotators of certain size —column. The last column shows the number of examples where a majority of annotators (3 or more) agree on the assignment of a certain category.

Impact	Annotator					Impact	Annotator				
	2	3	4	5	≥ 3		2	3	4	5	≥ 3
Installability	6	6	20	59	85	*Installability*	64	32	16	52	100
Requirements	65	73	100	37	210	*Maintenance*	45	18	4	0	22
Usability	50	71	129	96	296	*Reliability*	66	64	40	52	156
Other	0	13	121	0	134	*Other*	4	36	105	0	141
Compendium dataset						Mozilla dataset					

browser. Nowadays, it is a suite of tools that includes the Firefox browser and the Thunderbird e-mail client. This second dataset, which contains 598 defects, has undergone a similar pre-processing step to reduce the number of labels. In this case, only the *Installability*, *Maintenance* and *Reliability* defect impacts are kept from the 10 defect impacts originally labeled by the annotators. In the same way as the first dataset, the other defect impacts are replaced by the category *Other*.

Both the original and processed datasets of both domains are publicly available [36]. Table 1 shows the number of examples that each annotator assigned to each class label for both datasets. Although for *Installability* reports of the Compendium project the number of examples assigned by the different annotators is almost the same, the variability is considerable in the other categories. In some cases, such as the *Maintenance* reports of the Mozilla project, it is extreme. Indeed, a similar number of annotations does not imply consensus. Table 2 shows the assignment of examples to labels based on the consensus among annotators: each cell shows the number of examples assigned to a class label by a certain number of annotators. The last column shows the number of examples in which the consensus label is supported by a majority of annotators (three or more in our case). This can be seen as an estimation of the class distribution of the respective systems. It can be observed that the annotations for the Compendium dataset are more stable, resulting in the agreement of a larger number of annotators than in the case of the Mozilla dataset.

In Table 3, examples of real defect reports and labelings are shown for both studied systems. In some cases, the description of the defect is clear and the agreement among annotators is high. This behavior is mainly observed with *installability* defects, which are usually identified easily by annotators, as also reflected in Table 2. However, annotators do not usually show agreement in other categories or their vote is not unanimous. As shown in Table 2, both systems contain examples where annotators have reported two, three and even four different categories for the same defect report. Maybe due to lack of expertise or incomplete report description, the information provided by this type of defect for the learning process is certainly limited.

3.2 Learning from crowds

In software engineering, crowdsourcing usually refers to outsourcing software development to an undefined network of developers through web platforms [37, 38]. Crowdsourcing is a way of addressing a problem collaboratively and has become an important technique for dealing with software requirements, design, development, testing and decision making. In machine learning, learning from crowds [8, 9] is a weakly supervised classification problem [39] where the examples provided for model training are unreliably categorized by a set of annotators of questionable trustfulness and the ground truth is unavailable. Although such labeling usually shows disagreements among annotators (see Figure 1 for a graphical representation), competitive classifiers can be learnt from their combination. Snow et al. [40] measured the contribution of the *non-expert* annotators: they suggest that the combination of *four* non-expert annotations matches the knowledge of a domain expert. Global behaviors, those owed to the whole crowd, have been explored by Zhang et al. [11]. Other approaches try to model instance difficulty [41] or bias [42]. However, estimating the reliability of the individual annotators is the most common practice [8, 9, 41, 42]. Hence, the contribution of each annotator is balanced based on their reliability in order to carry out an informed aggregation of information. In this work, we use a learning from crowds approach to learn from the labelings of a set of novices and, thus, overcome the lack of the real (expert) categorization of the training set of defects.

Formally, the objective in standard supervised classification is to learn from a set of previous examples a classification model that anticipates the class label (category) of new unclassified examples. A problem is described by a set of n predictive variables $(X_1, \ldots, X_n)$ and a class variable C. Each variable has a set of possible values, with $\mathcal{C}$ specifically representing the set of values (class labels or categories) that the class variable can take. Thus, the dataset provided for learning the classifier $D = \{(\boldsymbol{x}^1, c^1), (\boldsymbol{x}^2, c^2), \ldots, (\boldsymbol{x}^N, c^N)\}$ is composed of N examples, where each example is a $(n+1)$-tuple (independent and identically distributed sampled from some unknown underlying probability distribution) that assigns a value x_j^i to each predictive variable X_j and a label c^i to the class variable C. In this context, the provided class labels c^i are considered completely reliable (ground truth). A classifier which maps examples $(\boldsymbol{x})$ to

Table 3: Examples of defects and the corresponding labelings provided by the annotators.

Summary	Description	L_1	L_2	L_3	L_4	L_5
Error Launching Compendium LD after install	Hi team, Error message launching Compendium LD after initial install: Java Virtual Machine Launcher Could not find the main class: com.compendium.ProjectCompendium. Program will exit. I have run through the suggestion on the forums of adding the path to javaw in the .bat, and verified the path through a command prompt is successful, same error. Any other tips? Regards, Eric	Installability	Other	Installability	Installability	Installability
Spell Checker	Add a spelling checker to Compendium with the ability to switch on and off auto-spell checking.	Requirements	Requirements	Requirements	Requirements	Requirements
Can small icons also work with images? Make small images?	Right now when you choose small icons, it shrinks the normal Compendium icons but not any reference node images, so they stay really big. Can we add an option to shrink those proportionally as well?	Requirements	Requirements	Usability	Requirements	Other
Text find/replace	Global search/find/replace functionality. Maybe coupled with existing search parameters. Ability to change text in found nodes without having to open the nodes, edit the label/detail, etc.	Requirements	Usability	Usability	Requirements	Requirements

Compendium dataset

NSS autoconf does not include IRIX	–enable-crypto does not work on IRIX as security/nss/configure.in does not define XP_UNIX and friends on IRIX.	Requirements	Reliability	Maintenance	Requirements	Maintenance
Mozilla automatically checks the “Reassign bug to” radio button	Mozilla automatically checks the “Reassing bug to” radio button in Bugzilla causing unintentional changes to bugs. Tested with win32 051404 mozilla win32 build on NT. More to come.	Other	Other	Reliability	Reliability	Reliability
installation failed with error -214 due to empty flash.xpi	seen on mac commercial build 2001-05-09-04-trunk. The installers, both full and stub, failed with a -214 error. Though the installation “appears” complete, when launched, it crashes at the end of the splash screen. I was able to use the non-installer bits for smoketesting	Installability	Installability	Installability	Installability	Installability
CreateChildList in nsPref.cpp needs to be fixed.	CreateChildList does not create the childList. buf is not getting assigned properly. PL_strncpy(buf, (char*)&childArray[i], PR_MIN (512, PL_strlen((char*)&childArray[i]) + 1)); and PL_strcat is failing PL_strcat(childList, buf); I have a patch for this.	Other	Maintenance	Maintenance	Reliability	Maintenance

Mozilla dataset

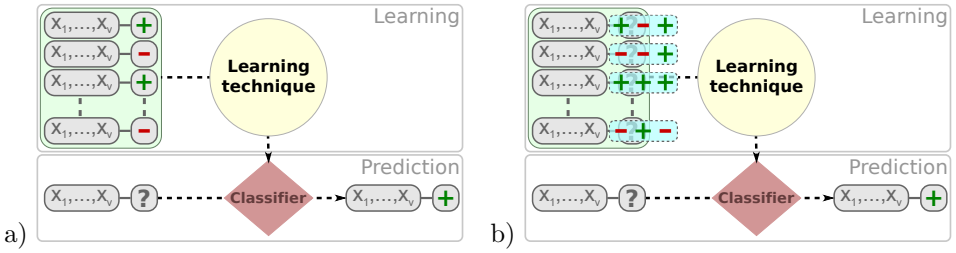
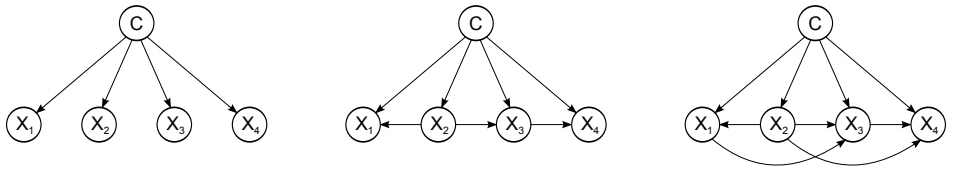


Figure 1: Using a hypothetical binary domain ($\{-, +\}$), graphical comparison of (a) standard supervised classification —each training example is provided with its real label— and (b) the learning from crowds paradigm —real labels are unknown; the opinions of 3 annotators are available for each example.



(a) Naive Bayes (NB)

(b) Tree-augmented naive Bayes (TAN)

(c) 2-dependence Bayesian network (KDB)

Figure 2: Examples of the structures of the Bayesian network classifiers used in this study.

categories (c) is learnt such that, given a new example $\boldsymbol{x}^*$, the classifier will anticipate the corresponding category c^*.

In the learning from crowds paradigm, the real class labels of the examples are unknown and only the subjective opinions of a set of t novice annotators are available. The information of supervision of each example $\boldsymbol{x}^i$ is codified by a t-tuple $\boldsymbol{l}^i$, where $l^i_a \in \mathcal{C}$ indicates the class label assessed by annotator L_a for $\boldsymbol{x}^i$. Thus, the training dataset is $D = \{(\boldsymbol{x}^1, \boldsymbol{l}^1), (\boldsymbol{x}^2, \boldsymbol{l}^2), \ldots, (\boldsymbol{x}^N, \boldsymbol{l}^N)\}$. Although the annotations are known to be noisy (the provided label l^i_a is not always the real *unknown* label c^i), assuming better-than-random annotators is a common practice in the related literature [8, 9]. The learning from crowds paradigm overcomes the unavailability of the real labels by combining the provided multiple annotations, $\boldsymbol{l}^i$, in an informed way. Only the way in which the information of supervision is provided differs from standard supervised classification; the objective and other assumptions remain the same.

3.3 Classification models

In this analysis, our ML technique learns Bayesian network models, which are used as probabilistic classifiers, i.e., *Bayesian Network Classifiers* (BNC) [43]. This choice is motivated by the interpretability of these models: influences and

Algorithm 1 Pseudocode of the structural learning procedure for TAN models.

procedure StructuralTAN(D) ▷ D: training dataset
  $MI_{ij} \leftarrow I(X_i, X_j|C), \forall i, j : i \neq j$ ▷ Conditional Mutual Information, using D
  $G \leftarrow$ complete undirected graph with all the variables $\{X_i\}_{i=1}^{n}$.
  Weight every edge $X_i \longleftrightarrow X_j$ with MI_{ij}
  $T \leftarrow$ Maximum weight spanning tree over G [44]
  $T \leftarrow$ Transform undirected edges to directed: Randomly select a variable X_i as the root and direct all edges outward from it
  $T \leftarrow T+$ Variable C + $\forall i$: edge $C \longrightarrow X_i$ ▷ Add naive Bayes structure
  return T
end procedure

Algorithm 2 Pseudocode of the structural learning procedure for KDB models.

procedure StructuralKDB(D, K) ▷ D: training dataset; K: no. of parents
  $MI_{ic} \leftarrow I(X_i, C), \forall i$ ▷ Mutual Information, using D
  $MI^c_{ij} \leftarrow I(X_i, X_j|C), \forall i, j : i \neq j$ ▷ Conditional Mutual Information, using D
  $T \leftarrow \emptyset$; $sNodes \leftarrow \{\arg\max_{X_i} MI_{ic}\}$ ▷ Initialize graph
  repeat
    Select $X_m = \arg\max_{X_i \notin sNodes} MI_{ic}$
    Select $min(|sNodes|, K)$ variables $\{X_j\}$ with the highest MI^c_{mj}
    $T \leftarrow T+$ edges from the selected variables $\{X_j\}$ to X_m
    $sNodes \leftarrow sNodes + X_m$
  until All the variables X_i are included in $sNodes$
  $T \leftarrow T+$ Variable C + an edge $C \longrightarrow X_i, \forall i$ ▷ Add naive Bayes structure
  return T
end procedure

dependencies among variables can be deduced from the explicit probability relationships. They can be graphically represented, enhancing model comprehensibility and facilitating the interaction with domain experts. Moreover, BNC have been successfully used to model many classification problems of different domains. A Bayesian network, represented by a pair $(\mathcal{G}, \boldsymbol{\theta})$, is a probabilistic graphical model that encodes the conditional dependencies between a set of random variables $\boldsymbol{V}$ using a directed acyclic graph (DAG). The graph structure, $\mathcal{G} = (\boldsymbol{V}, \boldsymbol{R})$, codifies the arcs $\boldsymbol{R}$ (conditional dependencies) between nodes $\boldsymbol{V} = (X_1, \ldots, X_n, C)$ (random variables), and $\boldsymbol{\theta}$ is the set of parameters of the conditional probability functions of each variable given its parents in the graph.

Specifically, three kinds of BNC where all the predictive variables are conditioned to the class variable have been considered in this study: naive Bayes classifier (NB) [45], tree augmented naive Bayes classifier (TAN) [46] and K-dependence Bayesian network classifier (KDB) [47]. Based on the assumption of conditional independence between the predictive variables given the class variable, the naive Bayes classifier presents the simplest network structure (see Figure 2). The TAN and KDB classifiers are more complex in terms of network structure and allow models to capture some conditional dependencies between predictive variables. Both the model parameters and the graph of conditional

(in)dependencies of a BNC can be estimated from a set of examples. In the specific case of learning from certainly labeled examples, maximum likelihood estimates of the model parameters can be obtained by means of frequency counts [48]. Regarding the graph structure, in this paper the standard methods for learning TAN [46] and KDB [47] structures have been implemented. Their pseudocodes are given in Algorithms 1 and 2, respectively. NB does not require structural learning as its structure is fixed. The general classification rule of this type of BNC is defined as,

$$\operatorname*{argmax}_c p(C=c)\prod_{j=1}^{n} p(X_j=x_j|\mathbf{PA}_j=\boldsymbol{pa}_j,C=c) \tag{1}$$

where $\operatorname{argmax}_c f(c)$ is an operator that finds the value c which maximizes the expression $f(c)$, $\boldsymbol{pa}_j$ is the vector of values assigned in the example $\boldsymbol{x}$ to the predictive variables, $\mathbf{PA}_j$, which are parents of X_j in the structure $\mathcal{G}$.

The lack of the ground truth labels prevents us from directly applying the standard BNC learning techniques for complete data. Precisely, the use of learning from crowds techniques allows us to deal with this issue. In this paper, two state-of-the-art techniques of different nature are used to show the performance of the crowd learning paradigm on two defect classification domains. On the one hand, a pre-processing technique that, using the K-means clustering algorithm, models labeling behaviors of the whole crowd is considered. Its result is a completely labeled dataset in which standard techniques can be applied for learning a classification model. On the other hand, an EM-based technique that models the individual behavior of each labeler is also applied. In this case, model learning and ground truth estimation are iteratively alternated.

3.4 K-means based method

The method proposed by Zhang et al. [11], which only considers annotations $\{\boldsymbol{l}^i\}_{i=1}^N$ (the corresponding examples $\{\boldsymbol{x}^i\}_{i=1}^N$ are disregarded), has been implemented (see Algorithm 3 for its pseudocode). First of all, the annotated labels are transformed into label counts disregarding the information about who provided each label: the *number of* annotators who provided class label c for example $\boldsymbol{x}$ is calculated for every example and label. These vectors of label counts are the examples provided to the K-means clustering algorithm, which is set up with k equal to the number of categories, $|\mathcal{C}|$. The vectors with the highest label count for each label c are used as initial centroids. As usual, the K-means algorithm assigns each example (vector of label counts) to a centroid. As each centroid was generated for representing a class label, the ground truth estimation of this technique assumes that each example belongs to the class label that is represented by its closest centroid.

This method outputs an estimation of the ground truth labels. That is, a vector $\boldsymbol{gs}$ in which each element $gs^i \in \mathcal{C}$ (with $i = \{1,\ldots,N\}$) is a class label. In this way, using this labeling together with the corresponding original predictive data, a complete dataset $\hat{D} = \{(\boldsymbol{x}^1, gs^1), (\boldsymbol{x}^2, gs^2), \ldots, (\boldsymbol{x}^N, gs^N)\}$ can be built

Algorithm 3 Pseudocode of the implemented K-means based approach.

procedure KMEANSAPPROACH(D) ▷ $D = \{(\boldsymbol{x}^1, \boldsymbol{l}^1), (\boldsymbol{x}^2, \boldsymbol{l}^2), \ldots, (\boldsymbol{x}^N, \boldsymbol{l}^N)\}$
  $\boldsymbol{R} \leftarrow new\ matrix($ `nRow:`N `, nCol:`$|\mathcal{C}| + 1$ $)$
  for $i \in \{1, \ldots, N\}$ **do**
    for $c \in \{1, \ldots, |\mathcal{C}|\}$ **do**
      $R_{ic} \leftarrow countsOfLabel(\boldsymbol{l}^i, c)$ ▷ No. annotators providing label c in $\boldsymbol{l}^i$
    end for
    $R_{ic+1} \leftarrow \sum_{i=2}^{|\mathcal{C}|} R_{ic} - R_{ic-1}$
  end for
  $iCentroids \leftarrow \{\arg\max_{i \in 1,\ldots,N} R_{ic}\}_{c=1}^{|\mathcal{C}|}$
  $\boldsymbol{gs} \leftarrow Kmeans(R, iCentroids, k = |\mathcal{C}|)$ ▷ Assign each example to a centroid
    ▷ Examples assigned to the centroid representing label c *belong to* label c
  return $\hat{D} = \{(\boldsymbol{x}^1, gs^1), (\boldsymbol{x}^2, gs^2), \ldots, (\boldsymbol{x}^N, gs^N)\}$
end procedure

and used to learn classification models by means of the techniques presented in the previous section. Thus, ground truth inference and model learning are, in this approach, two sequential but separate steps.

As aforementioned, the individual information about the annotators is disregarded. The individual labels and, therefore, the information about which annotator provided each label, are not considered. This makes any attempt to individually model the behavior (reliability) of the annotators impossible. On the contrary, this approach looks for profiles of label counts. That is, for each category, it approximates the mean counts of labels assigned by the annotators to examples of that category. The assumption that underlies this approach is that the tendency to confuse categories, a.k.a. bias in this context, is somehow global and can be modeled at crowd-level.

3.5 EM-based method

In contrast to the previously presented approach, the second technique, following a Dawid-Skene scheme [12], models individual annotators by means of a set of reliability parameters that are subsequently used to calibrate the contribution of the labels that they provide to ground truth estimation. An Expectation-Maximization (EM) based method previously proposed for the multi-dimensional learning from crowds problem [9] has been adapted to this unidimensional but multi-class classification task. The EM strategy [49] allows us to combine the estimation of a set of weights that model the reliability of each annotator and the learning of the model using the labels provided by the set of novices. In our method, the *Expectation* step estimates the reliability weights of the annotators and, in the *Maximization* step, the model parameters are re-estimated such that the likelihood is maximized given the data and the weights estimated in the *Expectation* step. Iteratively, both steps are repeated. Under general conditions, the iterative increase of the likelihood has been proved to converge to a stationary value (local maximum) [50].

When TAN or KDB classifiers are learnt, an outer loop to the traditional

Algorithm 4 Pseudocode of the implemented Structural EM method.

procedure StructuralEM($D, maxIt, \epsilon$) ▷ $D = \{(\boldsymbol{x}^1, \boldsymbol{l}^1), (\boldsymbol{x}^2, \boldsymbol{l}^2), \ldots, (\boldsymbol{x}^N, \boldsymbol{l}^N)\}$
    ▷ Stop conditions: max. no. iterations, $maxIt$ / threshold, ϵ
    $W \leftarrow$ initialReliabilityWeights(D)
    $G_0 \leftarrow$ initialStructure(D, W)
    repeat ▷ Increasing $i = 1, 2, ...$
        $\boldsymbol{\theta}_0 \leftarrow$ estimateParameters(D, W, G_{i-1})
        repeat ▷ Increasing $j = 1, 2, ...$
            $W \leftarrow$ reestimateReliabilityWeights($D, \mathbb{M} \equiv (G_{i-1}, \boldsymbol{\theta}_{j-1})$)
            $\boldsymbol{\theta}_j \leftarrow$ estimateParameters(D, W, G_{i-1})
        until (diff($\boldsymbol{\theta}_j, \boldsymbol{\theta}_{j-1}$) $< \epsilon$) Or ($j = maxIt$) ▷ Model parameter optim. loop
        $G_i \leftarrow$ improveStructure(D, W, G_{i-1})
    until ($G_i = G_{i-1}$) Or ($i = maxIt$) ▷ Model structure optimization loop
    return $\mathbb{M} \equiv (G_i, \boldsymbol{\theta}_j)$
end procedure

EM procedure allows us to combine model parameter estimation and structural learning (see Algorithm 4). This extension of EM, known as Structural EM [51], iteratively improves an initially-proposed structure (see Algorithms 1 and 2). At each iteration, the structural improvement is carried out by means of a one-step local search which looks for the structure that maximizes the complete-data minimal description length (MDL) score. The neighborhood is composed of all the structures (in the same space as the original one) that can be obtained by removing one conditional dependency between two predictive variables and adding another dependency between a different pair of predictive variables. When no structure overcomes the current one in terms of MDL, the algorithm stops.

For this study, two types of reliability weights, which codify the trustworthiness of each annotator, have been considered. On the one hand, a reliability weight per class label and annotator is used. These *per-label* weights (w_c^a, for all $a \in \{1, \ldots, t\}$) codify the reliability of each annotator L_a when they provide examples of a specific class label c. On the other hand, the *confusion-matrix* weights ($W_{cc'}^a$, for all $a \in \{1, \ldots, t\}$ and $c, c' \in \{1, \ldots, |\mathcal{C}|\}$) codify, for each annotator, both the reliability of an annotator when they predict a class label and the probability of label c' being the real label when the annotator provides c. Firstly, the initial set of reliability weights is estimated by comparing the annotations of each labeler with those of the rest of the annotators. Next, a model is learnt using a counting procedure for model parameter estimation which has been adapted to consider the multiple (weighted) labelings. A detailed description of the adapted procedure is presented in the next subsection. Once a model is available, in the *Expectation* step of the EM strategy, the annotator reliability weights can be re-estimated assuming that the ground truth is the output of the predictive model. Reliability weight estimation procedures, both initial and model-based assessments, are explained in detail in Section 3.5.2. A numerical example of the calculation involved in this process is available in the additional

material of this paper.

3.5.1 Estimation of model parameters

The standard parameter estimation procedure has been adapted to collect frequency counts from multiple noisy annotations per example, using the annotator reliability weights in order to carry out an informed aggregation of the different contributions. Similar to Hernández-González et al. [9], the parameter estimation procedure to collect frequency counts integrating the multiple and weighted labels can be expressed as follows:

$$N(\boldsymbol{u}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{l}^i) \in D} \sum_{c=1}^{|\mathcal{C}|} \mathbb{I}[x^i_{J_1} = u_1, \ldots, x^i_{J_k} = u_k] \cdot F^{\boldsymbol{l}^i}_{u_{k+1}} \tag{2}$$

where $\mathbb{I}[condition]$ is a function that returns 1 if *condition* is true and 0 otherwise, $\boldsymbol{u} = (u_1, \ldots, u_k, u_{k+1})$ is an instantiation of the random vector $\boldsymbol{U} = (X_{J_1}, \ldots, X_{J_k}, C)$, a sub-vector of the original $\boldsymbol{V} = (\boldsymbol{X}, C)$ with $\{J_1, \ldots, J_k\} \subseteq \{1, \ldots, n\}$. Finally, $F^{\boldsymbol{l}}_c$ is the reliability of assigning label c jointly taking into account the opinion of the annotators $\boldsymbol{l}$ and their reliability weights. With $\sum_{c=1}^{|\mathcal{C}|} F^{\boldsymbol{l}}_c = 1$, it is calculated differently depending on the type of annotator reliability weights. On the one hand, using the *per-label* weights (w^a_c), it is calculated as,

$$F^{\boldsymbol{l}}_c = \frac{\sum_{a=1}^{t} \mathbb{I}[l_a = c] \cdot w^a_c}{\sum_{c'=1}^{|\mathcal{C}|} \sum_{a=1}^{t} \mathbb{I}[l_a = c'] \cdot w^a_{c'}} \tag{3}$$

On the other hand, $F^{\boldsymbol{l}}_c$ is calculated using the *confusion-matrix* reliability weights ($W^a_{cc'}$) as follows,

$$F^{\boldsymbol{l}}_c = \frac{\sum_{a=1}^{t} \mathbb{I}[c \in \boldsymbol{l}] \cdot W^a_{l_a c}}{\sum_{c'=1}^{|\mathcal{C}|} \sum_{a=1}^{t} \mathbb{I}[c' \in \boldsymbol{l}] \cdot W^a_{l_a c'}} \tag{4}$$

3.5.2 Estimation of reliability weights for the annotators

A simple estimation of the reliability weights of the annotators (line 2 in Alg. 4), which only uses the available multiple labelings, is obtained by means of the consensus criterion [9]. In the case of *per-label* weights, the consensus weight of an annotator L_a in class label c is,

$$w^a_c = iRelWei_{label}(D) = \frac{1}{\Phi} \sum_{i=1}^{N} \mathbb{I}[l^i_a = c] \frac{1}{t-1} \sum_{a' \neq a} \mathbb{I}[l^i_{a'} = c] \tag{5}$$

with normalization factor $\Phi = \sum_{i=1}^{N} \mathbb{I}[l^i_a = c]$. In the case of *confusion-matrix* weights, the consensus weight of an annotator L_a for confusing label c' with c is,

$$W^a_{cc'} = iRelWei_{matrix}(D) = \frac{1}{\Phi} \sum_{i=1}^{N} \mathbb{I}[l^i_a = c] \frac{1}{t-1} \sum_{a' \neq a} \mathbb{I}[l^i_{a'} = c'] \tag{6}$$

Table 4: Relation of the different configurations used in the experiments and their equations.

	Acc	*Prob*	*Acc+Cons*	*Prob+Cons*
Per-label	Eq. 7	Eq. 8	(Eq. 7+Eq. 5)/2	(Eq. 8+Eq. 5)/2
Confusion-matrix	Eq. 9	Eq. 10	(Eq. 9+Eq. 6)/2	(Eq. 10+Eq. 6)/2

Once a model fit $\mathbb{M}$ is available, the re-estimation of the reliability weights of the annotators [9] (line 7 in Alg. 4) can be carried out using two different strategies: (1) an accuracy-based strategy (*Acc*), where the class label $\hat{c}$ predicted (according to Eq. 1) by the model $\mathbb{M}$ for each example is used as ground truth, and (2) a probability-based strategy (*Prob*), which uses the probability given by the model $\mathbb{M}$ to the labels assigned by each annotator to calculate their reliability weights. In the case of using *per-label* weights (w_c^a), both estimation techniques can be formulated as,

$$w_c^a = reRelWei_{label}^{acc}(D, \mathbb{M}) = \frac{1}{\Phi} \sum_{i=1}^{N} \mathbb{I}[l_a^i = c] \cdot \mathbb{I}[\hat{c}^i = c] \tag{7}$$

$$w_c^a = reRelWei_{label}^{prob}(D, \mathbb{M}) = \frac{1}{\Phi} \sum_{i=1}^{N} \mathbb{I}[l_a^i = c] \cdot p_{\mathbb{M}}(c|\boldsymbol{x}^i) \tag{8}$$

with normalization factor $\Phi = \sum_{i=1}^{N} \mathbb{I}[l_a^i = c]$. And, in the case of using the *confusion-matrix* reliability weights ($W_{cc'}^a$), both estimation procedures are,

$$W_{cc'}^a = reRelWei_{matrix}^{acc}(D, \mathbb{M}) = \frac{1}{\Phi} \sum_{i=1}^{N} \mathbb{I}[l_a^i = c] \cdot \mathbb{I}[\hat{c}^i = c'] \tag{9}$$

$$W_{cc'}^a = reRelWei_{matrix}^{prob}(D, \mathbb{M}) = \frac{1}{\Phi} \sum_{i=1}^{N} \mathbb{I}[l_a^i = c] \cdot p_{\mathbb{M}}(c'|\boldsymbol{x}^i) \tag{10}$$

where Φ is in both cases a normalization constant such that $\sum_{c'=1}^{|\mathcal{C}|} W_{cc'}^a = 1$.

As the EM strategy proposes a hill climbing approach for the problem of model parameter estimation, a procedure that updates the annotator reliability weights relying exclusively on the learnt model could be detrimental. If our EM procedure iteratively converges to a harmful classifier that only predicts a subset of labels, the estimated reliability weights can differ considerably from the real reliability values. In order to avoid this undesirable deviation, our method allows us to use the consensus weights (Eq. 5 or Eq. 6, as appropriate) throughout the iterations of the EM process as a correction term (*cons*). Thus, in this case the annotator reliability weights are re-estimated using the average value of the consensus weights and the model-estimated weights.

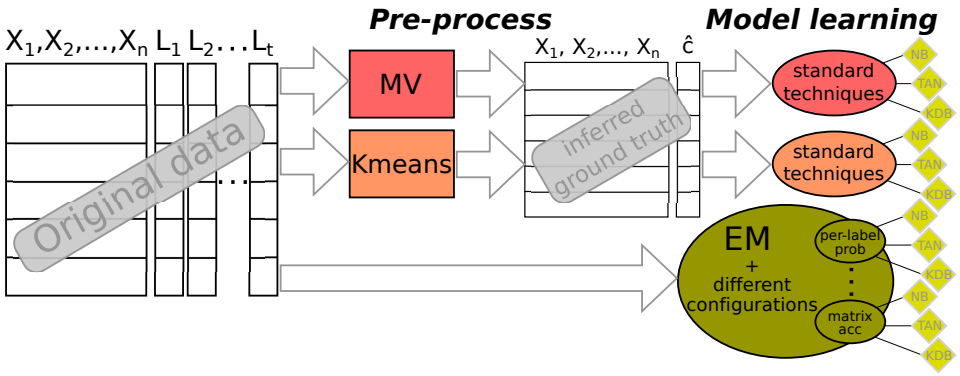


Figure 3: Graphical description of the learning process. The different techniques make use of the training dataset labeled by a crowd in a different way: the majority voting and k-means based techniques transform it before a classifier is built; and different configurations of the EM-based technique deal with the raw training data in different ways to build a classifier.

4 Experimental work

4.1 Experimental settings

Different experiments have been carried out using both the K-means based and the EM-based learning techniques to learn three types of Bayesian network classifiers (NB, TAN and 2DB) from both datasets [36]. In the case of the EM-based technique, all the possible configurations have been tested for its three adjustable features: the type of reliability weights (*per-label* and *confusion-matrix*) of the annotators, the weight estimation procedure (*Prob* and *Acc*) and the use, or not, of consensus weight correction (*cons*). In order to assess the size of the improvement achieved with the implementation of the crowd learning paradigm, the majority voting (MV) strategy is used as a baseline. This simple strategy completes the dataset by labeling each example with the label most voted among the set of novices and, in this way, learns as in a standard supervised classification problem. A graphical description of the training process is provided in Figure 3. The use of the training crowd-labeled data by the different learning techniques is compared.

All these experiments were carried out using our own implementation of the different learning algorithms and evaluation strategies. As our developments are written in Java, we can make easy use of several data management features currently implemented in the popular software Weka [52]. In this way, the text reported by users to describe defects (in two text fields, *summary* and *description*) was processed. In a pre-processing stage, standard NLP techniques have been used to extract a relevant set of variables from the text fields and transform the original database into a dataset which can be handled by ML techniques. Specifically, the popular *StringToWordVector* filter implemented in Weka [52]

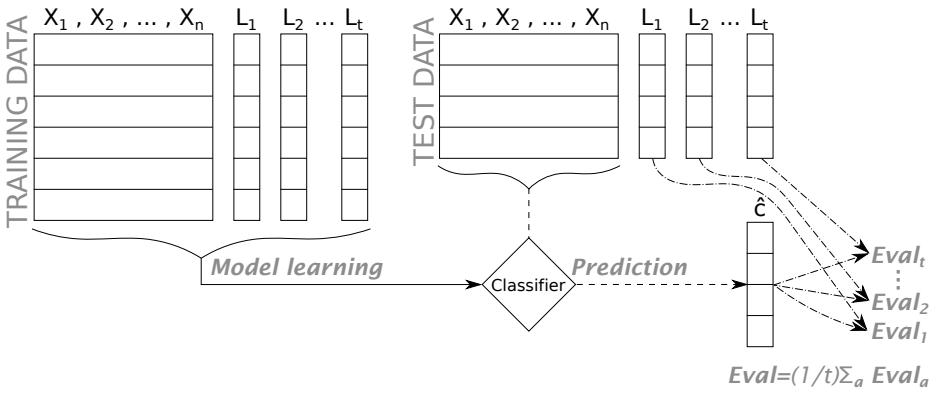


Figure 4: Graphical description of the evaluation strategy. The final performance is the mean value after comparing each annotation with the predicted labels.

was used. Stop-words were removed based on Rainbow [53], text was converted to lowercase; the iterated version of the Lovins stemmer [54] was applied as well as an alphabetic tokenizer where tokens are formed only using contiguous alphabetic sequences. For each word a numeric variable is created which, for each defect, takes as value the *Term Frequency-Inverse Document Frequency* (TF-IDF) ratio. Without a ground truth to compare with, the number of selected attributes was set to 100. Finally, each numeric variable was transformed into a binary variable using a step function which takes a positive value only if the original numeric value is larger than zero. Similarly, both parameters of the EM-based technique have been set to their default values [9]: a threshold indicating parametric convergence (set to 0.1%) and the maximum number of iterations (fixed to 200). The use of default parameters allows us to focus this work on the benefits of the learning from crowds approach. Note that there is room for performance improvement via this pre-processing step.

Model evaluation is not straightforward in the learning from crowds paradigm. The lack of a ground truth (certain labeling) makes the use of standard evaluation techniques impossible. Given the relatively recent emergence of the learning from crowds paradigm, the model evaluation in this scenario is still a field to be explored. In this paper, the evaluation strategy followed is based on the same idea that confers its characteristic robustness on the majority voting: the combination of multiple independent assessments [9, 40]. That is, the *mean* value of the performance metric calculated using the labels of one annotator at a time as ground truth is considered. In practice, all the experiments in this section are evaluated as follows (see Fig. 4): (1) after model learning, the performance of the model is estimated using the annotations of each labeler, one at a time, as ground truth, and (2) the mean value of all the estimates is the final performance value. All the experimental results are obtained with a 10×5-fold cross validation procedure [55].

Table 5: Definition of the evaluation metrics based on a confusion matrix $\boldsymbol{V}$, where $V_{c\hat{c}}$ is the number of examples predicted by the learnt classifier as class $\hat{c}$ when their real class is c.

Recall_c	$V_{cc}/\sum_{c'=1}^{\lvert\mathcal{C}\rvert} V_{c'c}$	Min-Recall	$\min_{c\in\{1,\dots,\lvert\mathcal{C}\rvert\}} Recall_c$
Precision_c	$V_{cc}/\sum_{c'=1}^{\lvert\mathcal{C}\rvert} V_{cc'}$	Max-Recall	$\max_{c\in\{1,\dots,\lvert\mathcal{C}\rvert\}} Recall_c$
F1-mean	$\frac{1}{\lvert\mathcal{C}\rvert}\sum_{c=1}^{\lvert\mathcal{C}\rvert} \frac{2\cdot Recall_c\cdot Precision_c}{(Recall_c+Precision_c)}$	A-mean	$\frac{1}{\lvert\mathcal{C}\rvert}\sum_{c=1}^{\lvert\mathcal{C}\rvert} Recall_c$
Accuracy	$\frac{1}{\lvert\mathcal{C}\rvert}\sum_{c=1}^{\lvert\mathcal{C}\rvert} V_{cc}$		

4.2 Results

In order to provide a complete overview of the performance of the learnt classifiers, results in terms of A-mean (Table 6), F1-mean (Table 7) and accuracy (Table 8) are presented. See Table 5 for a description of the evaluation metrics used in this paper. In tables 6 to 8, the MV strategy, the K-means based technique and the eight different configurations (all the possible combinations of the three features, see Table 4) of the EM-based technique are displayed in columns; each row shows the experiments using a specific BNC in one of the datasets. The best configuration for each BNC and dataset (by row) is highlighted in bold. In (multi-class) classification, analyzing the performance of a classifier depends on the preferences of the final user. Accuracy is a global measure that evaluates the performance of a classifier independently of the number of class labels. Classifiers which completely disregard one or more class labels can show competitive accuracy values if their performance in examples of the rest of the categories is outstanding. It is, therefore, a good option for users interested in classifiers which show high global performance. However, if the final user is interested in classifiers which perform well in all the class labels, A-mean [56], the mean of the recall values, or F1 [57] are more suitable metrics. To illustrate this trade-off between local and global performance, Figure 5 shows the minimum and maximum recall values obtained by the different classifiers in any of the class labels. These values provide an insight into the performance of the classifiers across class labels: large differences among minimum and maximum values usually correspond to large accuracy values and low differences to large A-mean and F1-mean values.

The simplest solution, a standard supervised classification approach that uses the most-voted labels (MV) as ground truth, gives a baseline whose robust behavior has already been analyzed [9]. In these experiments, MV is a solid strategy which gives a competitive baseline; it is able to occasionally beat the performance of some configurations of the applied techniques. However, both applied crowd learning techniques consistently outperform the basic MV strategy. In the *Compendium* domain, the results of the k-means based technique overcome those of MV in terms of all the metrics. However, mainly with the TAN and KDB classifiers, the MV strategy beats the k-means approach in the *Mozilla* domain. Regarding the EM-based approach, configurations with *confusion-matrix* weights always outperform MV in terms of A-mean, where

Table 6: Results in terms of A-mean of the BNC classifiers learnt from both datasets —rows— using a K-means based technique [11] and a EM-based technique with different configurations (Table 4) —columns. Majority Voting (MV) is used as a baseline strategy.

	BNC	MV	Kmeans	EM							
				Per-label				*Confusion-matrix*			
				Prob	*Prob+Cons*	*Acc*	*Acc+Cons*	*Prob*	*Prob+Cons*	*Acc*	*Acc+Cons*
Compendium	NB	0.455	**0.593**	0.480	0.474	0.479	0.475	0.491	0.492	0.488	0.492
	TAN	0.433	**0.603**	0.436	0.440	0.436	0.442	0.456	0.458	0.454	0.460
	2DB	0.404	**0.572**	0.400	0.408	0.402	0.413	0.424	0.425	0.417	0.415
Mozilla	NB	0.454	0.475	0.459	0.462	0.453	0.463	0.479	0.477	**0.480**	0.479
	TAN	0.502	0.490	0.491	0.523	0.496	0.519	**0.529**	0.528	0.521	**0.529**
	2DB	0.480	0.475	0.496	0.496	0.498	0.494	0.508	0.508	**0.513**	0.498

the differences are up to 4 percentage points. Nevertheless, configurations using *per-label* weights consistently beat MV accuracy values, with differences which are close to 3 percentage points.

The results reveal a clear behavior: the K-means based technique outperforms the rest of the approaches in the *Compendium* dataset, whereas in the *Mozilla* dataset the best performing approach is the EM-based technique. It is observed in terms of all the measured metrics. In the bar graphs corresponding to the *Compendium* dataset in Figure 5, the difference between the K-means approach and the rest of techniques is especially noticeable: it shows the best results in terms of both maximum and minimum recall. Apart from the iterative nature of the EM strategy (the K-means approach works as a pre-process that produces an estimate of the ground truth), the main difference between both approaches is the behavior that they aim to model. Whereas the K-means based technique can only model biases shown by the whole crowd (annotators usually confuse labels c and c'), the EM-based approach can model individual biases (annotator a tends to confuse labels c and c'). This is the most feasible explanation for the different behaviors of both methods in both domains. In Table 1 it can be observed that annotations for the *Compendium* domain are similar for all the labelers. The main divergence relates categories *Usability* and *Other* (when the former is observed more frequently by an annotator, the latter is not annotated as often, and vice versa). However, in the case of the *Mozilla* domain, different behaviors can be observed among annotators; from annotator L_2, who provides balanced annotations, to annotators L_1 and L_5, who seem to label similarly both overpopulating the *Other* category. In this last dataset and according to the experimental results, modeling annotators individually is probably an adequate decision. With the *Compendium* dataset, the global modeling carried out by the K-means approach seems to be more appropriate.

Regarding the EM-based approach and its different configurations, although the differences are slight, the *Prob* procedure mostly outperforms *Acc* according to A-mean and F1 metrics (tables 6 and 7, respectively). Note that the performance of a classifier in all the class labels contributes to the computation of these metrics. Similarly, the use of *confusion-matrix* reliability weights seems more suitable if one of these metrics has to be optimized. The trend is clearly

Table 7: Results in terms of mean F1 of the BNC classifiers learnt from both datasets —rows— using a K-means based technique [11] and a EM-based technique with different configurations (Table 4) —columns. Majority Voting (MV) is used as a baseline strategy.

	BNC	MV	Kmeans	EM							
				Per-label				*Confusion-matrix*			
				Prob	*Prob+Cons*	*Acc*	*Acc+Cons*	*Prob*	*Prob+Cons*	*Acc*	*Acc+Cons*
Mozilla Compendium	NB	0.407	**0.572**	0.393	0.398	0.387	0.393	0.408	0.409	0.408	0.407
	TAN	0.393	**0.589**	0.374	0.387	0.372	0.382	0.399	0.400	0.400	0.402
	2DB	0.381	**0.562**	0.345	0.364	0.352	0.367	0.389	0.386	0.387	0.381
Mozilla Defects	NB	0.394	0.376	0.405	0.402	0.404	0.403	**0.407**	0.405	0.401	0.405
	TAN	0.449	0.408	0.432	0.454	0.437	0.452	0.462	**0.463**	0.452	0.462
	2DB	0.439	0.395	0.429	0.442	0.430	0.440	**0.447**	0.446	0.444	0.439

Table 8: Results in terms of accuracy of the BNC classifiers learnt from both datasets —rows— using a K-means based technique [11] and a EM-based technique with different configurations (Table 4) —columns. Majority Voting (MV) is used as a baseline strategy.

	BNC	MV	Kmeans	EM							
				Per-label				*Confusion-matrix*			
				Prob	*Prob+Cons*	*Acc*	*Acc+Cons*	*Prob*	*Prob+Cons*	*Acc*	*Acc+Cons*
Mozilla Compendium	NB	0.474	**0.554**	0.482	0.478	0.480	0.480	0.454	0.456	0.453	0.452
	TAN	0.465	**0.565**	0.471	0.474	0.468	0.475	0.441	0.441	0.442	0.443
	2DB	0.459	**0.535**	0.461	0.465	0.467	0.468	0.437	0.435	0.437	0.428
Mozilla Defects	NB	0.456	0.423	**0.463**	0.462	0.462	**0.463**	0.448	0.446	0.425	0.439
	TAN	0.526	0.449	0.535	0.532	**0.539**	0.535	0.505	0.501	0.479	0.495
	2DB	0.518	0.431	0.541	0.537	**0.543**	0.536	0.485	0.483	0.477	0.472

noticeable in the experimental results: configurations using *per-label* reliability weights always outperform configurations using *confusion-matrix* weights in terms of global accuracy (Table 8), and configurations using *confusion-matrix* weights always stand out in terms of A-mean or F1 metrics. On the one hand, it can be observed in Figure 5 that configurations with *per-label* weights often show minimum recall values near to 0. This behavior is associated with classifiers which concentrate their performance in a subset of class labels; usually, in the most populated categories. Performing robustly in highly populated class labels can lead to competitive global performance (e.g., in terms of accuracy) even when results in sparsely populated categories are poor. On the other hand, high A-mean or F1 values are associated with high minimum recall values. As these metrics balance the performance on all the class labels, high values can only be obtained when the performance is competitive on each label. Moreover, the use of consensus correction affects the results mainly when *per-label* weights are used. To sum up, *per-label* weights promote classifiers with competitive global performance, whereas *confusion-matrix* weights are appropriate whenever the objective is an averaged competitiveness across class labels (defect types, in our case).

Another interesting trend is the different performance of the three types of BNCs when they are learnt with the different techniques. Inarguably, the best performance is shown by TAN classifiers, always associated to the best learning

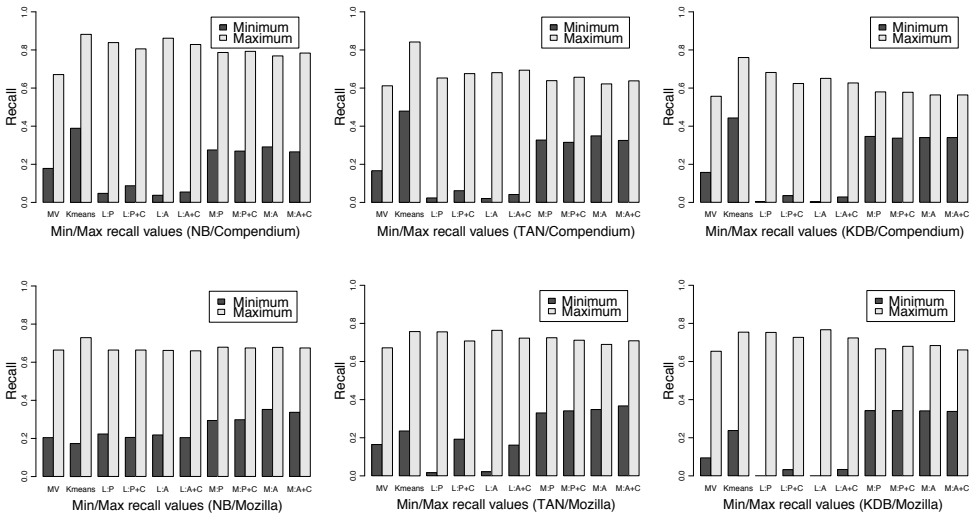


Figure 5: Graphical representation of minimum and maximum recall values obtained by the BNC classifiers —columns— learnt from both datasets —rows. Similar to tables 6 to 8, results are displayed in each subfigure for majority voting (first pair of bars), K-means based technique [11] (second pair), and different configurations of the EM-based technique (Table 4).

technique, the K-means approach in the *Compendium* dataset and the EM-based technique for *Mozilla*. This means that probabilistic relationships among predictive variables have been correctly modeled. When the MV strategy and the EM-based approach learn from the Compendium dataset, the best results are associated with the experiments that infer NB classifiers. That is, when the circumstances are not favorable, NB still performs reasonably well, showing its robustness. In the same way, the high maximum-recall values of the NB classifiers in the *Compendium* dataset (Figure 5) are noteworthy. Learnt 2DB classifiers show a competitive behavior in the *Mozilla* dataset, but not to the same extent as in the *Compendium* domain. This also emphasizes the different nature of both domains. Allowing the learning techniques to introduce extra relationships among predictive variables does not boost the performance of the classifiers. Thus, either no such relationships are present on the *Compendium* data or learnt models overfit the training data.

4.3 Discussion

Two crowd learning techniques have been applied to the task of defect classification. The EM-based technique, an adaptation to this unidimensional multi-class problem of our proposal for multi-dimensional problems [9], stands out in the *Mozilla* domain. The second approach, a K-means based technique [11], out-

performs the rest of methods in the *Compendium* domain. In spite of the high variability observed in the annotations (see Table 3), the experimental results show that learning to classify defects without the ground truth, only using the labelings provided by novices, is possible. Therefore, the proposed crowd learning paradigm is a robust choice for solving the defect classification problem. This opens an interesting path for reducing the reliance on expert knowledge for future software engineering classification tasks. Indeed, the performance improvement regarding the majority voting strategy shows the value of modeling the behavior of the novices, either globally —the contribution of the whole crowd— or individually —the annotations of each labeler.

The use of an ML approach in practice will result in a classification model that, given a new defect, predicts its category (in this case, ODC Defect Impact). Following the predictions of any classifier involves a partial risk, as an irreducible error may exist even when the best possible model is learnt. The existence of this irreducible error, known as Bayes error rate [58], is inherent to the problem and should always be taken into account. In our case, an estimation of the probability of error can be obtained for the learnt models: the summation over the probabilities of all the label assignments which do not maximize Equation 1 for any possible defect description. The cost of this estimation increases dramatically with the size of the descriptor vector (n). Although an implementation of the proposed approach is liable to classification error, the amount of mistakes can be minimized with training data and rates of confidence on the classification can be obtained.

Theoretical studies [59, 9] and other previous real applications [40, 60] suggest that the classifiers learnt by means of the crowd learning approach are competitive with the standard supervised classification whenever enough training data is provided. In our specific task one can expect that defects will continue being reported; i.e., more data will be available. However, in order to avoid overloading annotators, their effort can be focused on labeling the most useful examples. As not all the types of defects are equally difficult to classify (see Fig. 5), further developments should ideally select the defects which need to be annotated by the novices to boost the learning process: those which are not accurately classified by the classifier. Similarly, Table 3 shows reports where the agreement is unanimous. The number of annotators who are asked to annotate each defect report could be optimized to further reduce the cost of the labeling process. The estimated reliability weights could be taken into account to select, individually for each defect, the annotators to be questioned. Although it is a common practice to assume that annotators are novices, nothing prevents a domain expert from participating. The presented EM-based technique is able to identify experts and promote their annotations. Once identified, an effective procedure would firstly ask experts for their opinion. This selective learning process can be achieved by means of active learning [61], a strategy that allows the classifier to be used in production and improved in parallel. Its application to the learning from crowds paradigm has already been studied [60]. The active learning extension for the proposed paradigm would cover all the needs of an automatic defect classification procedure implemented in a real system and

would allow the classifier to be continuously improved at the same time.

Finally, an issue tracking system will have to be ultimately adapted to include the developments required for the participation of the community, which is necessary to put the proposed approach into production. The study, development and implementation of the ideas discussed in this paper is a step forward towards the use of the presented approach in real world systems.

4.4 Threats to Validity

Concerning external validity, an obvious threat is the representativeness of the studied systems, Compendium and Mozilla. Software systems usually have specific features such as the application domain, development environment and number of people. Different systems usually differ in the distribution of types of defects and, therefore, machine learning techniques need to adjust to the specific environment of each problem. Moreover, in the presented applications five annotators participated in the labeling processes. The results show that, in spite of the high levels of noise reported in Table 1, their contributions are informative and can be used to learn classification models. However, a larger number of annotators is expected to enhance the performance of the different methods, particularly that of the MV strategy [9, 40]. Although a more extensive study would certainly be more conclusive, two systems have been analyzed in the present study to foster representativeness. According to the results presented in the previous section, both domains are different enough to observe particular behaviors and diversity of performance among the used techniques.

Concerning construct validity, the quality in the issue tracking system makes it hard to easily classify defect data manually. We do not address other problems faced in the defect repositories such as defect duplicates. Apart from the summary and the description of defects, more data which could be extracted from Bugzilla repositories might be helpful. Other preprocessing decisions could have been chosen or optimized: e.g., removal of outliers or text field (natural language) processing. In order to focus the present study on the enhancement associated to the application of the learning from crowds paradigm, standard NLP procedures and default values have been used. The optimization of these procedures for the defect classification task would likely report improved performance. Moreover, and following the same reason, i.e., to focus the discussion on the usefulness of the class information provided by the multiple annotators, the original databases were transformed to 4-class classification problems. This decision could have had an impact on the results. However, dealing with the original databases would have required specific techniques to deal with the multi-class imbalance classification problem and their inclusion might obscure the interpretability of the results. Moreover, the techniques that we would have required for such an approach are not available in the state-of-the-art as the problem has not been addressed yet in the machine learning community. Both the original and processed databases are publicly available [36] to guarantee replicability.

Internal validity is concerned with whether the automated classifications

have arisen as a result of chance or not. In the case of 4 balanced class labels, the probability of randomly assigning the right label to an example is $1/4 = 0.25$. Assuming a random assignment of labels according to the distribution of labels estimated for the studied domains —based on the last column of Table 2—, the probability of being right is approximately $0.298 = (0.12^2 + 0.29^2 + 0.41^2 + 0.18^2)$ for the Compendium system and $0.312 = (0.24^2 + 0.05^2 + 0.37^2 + 0.34^2)$ for Mozilla. Both domains have similar probability of randomly selecting the real label. Taking this and the results of the previous section into account, it can be concluded that the automated classifications are not a product of chance. However, the performance of the learnt classifier is different in both datasets. That may be a product of the discriminant ability of the texts describing the defects and the NLP procedures applied to them. That is, the predictive variables have to be informative for this task to succeed.

5 Conclusions and future work

In this paper, the proposal of automation of the defect classification problem without the supervision of an expert, only relying on multiple partially reliable annotators, has been presented and tested in two real systems, *Compendium* and *Mozilla*. Two state-of-the-art methodologies, one based on the EM strategy and another one based on the K-means clustering algorithm, have been applied to learn Bayesian network classifiers from reported defects.

Both techniques and the different tested configurations show their competitive behavior in both domains. Whereas the K-means based technique models the crowd of annotators as a whole, the EM-based technique tries to individually model the different annotators of the crowd. Their performance is different through both studied domains. However, both crowd learning techniques systematically outperform a basic approach based on standard classification which uses the most-voted labels, encouraging the study of advanced techniques to combine the multiple contributions. Although further research is required, this study supports the use of a *learning from crowds* approach to defect classification when expert knowledge is not available.

For future work, dealing with the original 13-category problem would require us to model the studied systems as multi-class imbalance problems. Specific machine learning techniques, such as SMOTEBoost [62], have already been proposed to deal with this type of classification problem. However, their adaptation to the learning from crowds paradigm is not straightforward and would require further research. Specifically, we would like to study the effect of a set of skewed annotators on the learning process of a domain where the types of the reported defects are also unbalanced [63]. Regarding the evaluation of models learned from crowds without ground truth, it would be interesting to explore how the reliability weights estimated by, for example, the EM-based technique during the learning phase can be used to constrain the contribution of the different partial scores in the calculation of the final metric score (see Figure 4).

Acknowledgments

This work has been partially supported by the Basque Government (IT609-13, Elkartek BID3A), the Spanish Ministry of Economy and Competitiveness (TIN2016-78365-R) and the University-Society Project 15/19 (Basque Government and University of the Basque Country UPV/EHU). Jose A. Lozano is also supported by BERC program 2014-2017 (Basque Government) and Severo Ochoa Program SEV-2013-0323 (Spanish Ministry of Economy and Competitiveness). Daniel Rodriguez carried out this work while visiting Oxford Brookes University. He is partly supported by projects BadgePeople TIN2016-76956-C3-3-R. We would like to thank Varsha Veerappa for her help with data collection.

References

[1] B. Boehm, V. R. Basili, Software defect reduction top 10 list, in: B. Boehm, H. D. Rombach, M. V. Zelkowitz (Eds.), Foundations of empirical software engineering: the legacy of Victor R. Basili, Springer, 2005, pp. 426–431.

[2] A. Endres, An analysis of errors and their causes in system programs, in: Proc. Int. Conf. Reliable Software, ACM, New York, NY, USA, 1975, pp. 327–336.

[3] R. B. Grady, Practical Software Metrics for Project Management and Process Improvement, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[4] S. Wagner, Defect classification and defect types revisited, in: Proc. 2008 Workshop on Defects in Large Software Systems, DEFECTS'08, ACM, New York, NY, USA, 2008, pp. 39–40.

[5] D. N. Card, Learning from our mistakes with defect causal analysis, IEEE Software 15 (1) (1998) 56–63.

[6] F. Thung, D. Lo, L. Jiang, Automatic defect categorization, in: Proc. 19th Working Conf. Reverse Engineering, 2012, pp. 205–214.

[7] G. Lugosi, Learning with an unreliable teacher, Pattern Recognit. 25 (1) (1992) 79–87.

[8] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, L. Moy, Learning from crowds, J. Mach. Learn. Res. 11 (2010) 1297–1322.

[9] J. Hernández-González, I. Inza, J. A. Lozano, Multidimensional learning from crowds: Usefulness and application of expertise detection, Int. J. Intell. Syst. 30 (3) (2015) 326–354.

[10] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, M.-Y. Wong, Orthogonal defect classification-a concept for in-process measurements, IEEE Trans. Softw. Eng. 18 (11) (1992) 943–956.

[11] J. Zhang, V. S. Sheng, J. Wu, X. Wu, Multi-class ground truth inference in crowdsourcing with clustering, IEEE Trans. Knowl. Data Eng. 28 (4) (2016) 1080–1085.

[12] A. P. Dawid, A. M. Skene, Maximum likelihood estimation of observer error-rates using the EM algorithm, J. R. Stat. Soc. Ser. C-Appl. Stat. 28 (1) (1979) 20–28.

[13] IEEE, IEEE Std 1044-1993. IEEE Standard Classification for Software Anomalies (1993).

[14] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo, S. Godfrey, Defect categorization: Making use of a decade of widely varying historical data, in: Proc. 2nd ACM-IEEE Int. Symp. Empirical Software Engineering and Measurement, ESEM'08, ACM, New York, NY, USA, 2008, pp. 149–157.

[15] B. Freimut, C. Denger, M. Ketterer, An industrial case study of implementing and validating defect classification for process improvement and quality management, in: Proc. 11th IEEE Int. Symp. on Software Metrics (METRICS'05), 2005, pp. 10–19.

[16] T. Nakamura, L. Hochstein, V. R. Basili, Identifying domain-specific defect classes using inspections and change history, in: Proc. 2006 ACM/IEEE Int. Symp. Empirical Software Engineering (ISESE'06), ACM, New York, NY, USA, 2006, pp. 346–355.

[17] J. Duraes, H. Madeira, Definition of software fault emulation operators: a field data study, in: Proc. Int. Conf. Dependable Systems and Networks, 2003, pp. 105–114.

[18] D. Falessi, G. Cantone, Exploring feasibility of software defects orthogonal classification, in: Proc Int. Conf. Software and Data Technologies, Springer Berlin Heidelberg, 2006, pp. 136–152.

[19] M. Butcher, H. Munro, T. Kratshmer, Improving software testing via ODC: Three case studies, IBM Syst. J. 41 (1) (2002) 31–44.

[20] M. Soylemez, A. Tarhan, Using process enactment data analysis to support orthogonal defect classification for software process improvement, in: Proc. Joint Conf. 23rd Int. Workshop on Software Measurement and 8th Int. Conf. Software Process and Product Measurement (IWSM-MENSURA), 2013, pp. 120–125.

[21] N. Bridge, C. Miller, Orthogonal defect classification using defect data to improve software development, Softw. Qual. J. 3 (1998) 1997–8.

[22] R. Mays, C. Jones, G. Holloway, D. Studinski, Experiences with defect prevention, IBM Syst. J. 29 (1) (1990) 4–32.

[23] N. Schneidewind, H.-M. Hoffmann, An experiment in software error data collection and analysis, IEEE Trans. Softw. Eng. 5 (3) (1979) 276–286.

[24] T. J. Ostrand, E. J. Weyuker, Collecting and categorizing software error data in an industrial environment, J. Syst. Softw. 4 (4) (1984) 289–300.

[25] V. R. Basili, B. T. Perricone, Software errors and complexity: An empirical investigation, Commun. ACM 27 (1) (1984) 42–52.

[26] R. A. Demillo, A. P. Mathur, A grammar based fault classification scheme and its application to the classification of the errors of TEX, Tech. rep., Software Engineering Research Center and Department of Computer Sciences, Purdue University, W. Lafayette, IN 47907 (November 1995).

[27] T. Hall, D. Bowes, S. Counsell, L. Moonen, A. Yamashita, Software fault characteristics: A synthesis of the literature, `http://bura.brunel.ac.uk/handle/2438/11013` (2015).

[28] M. Leszak, D. E. Perry, D. Stoll, Classification and evaluation of defects in a project retrospective, J. Syst. Softw. 61 (3) (2002) 173–187.

[29] L. Buglione, A. Abran, Introducing root-cause analysis and orthogonal defect classification at lower CMMI maturity levels, in: Proc. Int. Conf. Software Process and Product Measurement (Mensura'06), 2006, pp. 29–40.

[30] K. Herzig, S. Just, A. Zeller, It's not a bug, it's a feature: How misclassification impacts bug prediction, in: Proc. 2013 Int. Conf. Software Engineering, 2013, pp. 392–401.

[31] P. Runeson, M. Alexandersson, O. Nyholm, Detection of duplicate defect reports using natural language processing, in: 29th Int. Conf. Software Engineering (ICSE), 2007, pp. 499–510.

[32] N. Jalbert, W. Weimer, Automated duplicate detection for bug tracking systems, in: Proc. 2008 IEEE Int. Conf. Dependable Systems and Networks With FTCS and DCC (DSN), 2008, pp. 52–61.

[33] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, Y.-G. Guéhéneuc, Is it a bug or an enhancement?: A text-based approach to classify change requests, in: Proc. 2008 Conf. Center for Advanced Studies on Collaborative Research: Meeting of Minds, ACM, New York, NY, USA, 2008, pp. 23:304–23:318.

[34] Y. Zhou, Y. Tong, R. Gu, H. Gall, Combining text mining and data mining for bug report classification, J. Softw.: Evol. Process 28 (3) (2016) 150–176.

[35] L. Huang, V. Ng, I. Persing, M. Chen, Z. Li, R. Geng, J. Tian, AutoODC: Automated generation of orthogonal defect classifications, Automat. Softw. Eng. 22 (1) (2015) 3–46.

[36] J. Hernández-González, D. Rodriguez, I. Inza, R. Harrison, J. A. Lozano, Two datasets of defect reports labeled by a crowd of annotators of unknown reliability, Data in Brief (2017) 1–7, in press.

[37] J. Howe, The rise of crowdsourcing, Wired Mag. 15 (6) (2006) 1–4.

[38] K. Mao, L. Capra, M. Harman, Y. Jia, A survey of the use of crowdsourcing in software engineering, Tech. Rep. RN/15/01, Department of Computer Science, University College London (2015).

[39] J. Hernández-González, I. Inza, J. A. Lozano, Weak supervision and other non-standard classification problems: A taxonomy, Pattern Recognit. Lett. 69 (2016) 49–55.

[40] R. Snow, B. O. Connor, D. Jurafsky, A. Y. Ng, D. Labs, C. St, Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks, in: Proc. Conf. Empirical Methods in Natural Language Processing, Vol. 254, 2008, pp. 254–263.

[41] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, J. R. Movellan, Whose vote should count more: Optimal integration of labels from labelers of unknown expertise, in: Proc. Advances in Neural Information Processing Systems 22 (NIPS), 2009, pp. 2035–2043.

[42] P. Welinder, S. Branson, S. Belongie, P. Perona, The multidimensional wisdom of crowds, in: Proc. Advances in Neural Information Processing Systems 23 (NIPS), 2010, pp. 2424–2432.

[43] C. Bielza, P. Larrañaga, Discrete Bayesian network classifiers: a survey, ACM Comput. Surv. 47 (1) (2014) 5.

[44] C. K. Chow, C. N. Liu, Approximating discrete probability distributions with dependence trees, IEEE Trans. Inf. Theory 14 (3) (1968) 462–467.

[45] D. J. Hand, K. Yu, Idiot's bayes—not so stupid after all?, Int. Stat. Rev. 69 (3) (2001) 385–398.

[46] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (2–3) (1997) 131–163.

[47] M. Sahami, Learning limited dependence Bayesian classifiers, in: Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining, 1996, pp. 335–338.

[48] D. Heckerman, A tutorial on learning with bayesian networks, Tech. Rep. MSR-TR-95-06, Learning in Graphical Models (1995).

[49] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc. Ser. B-Stat. Methodol. 39 (1) (1977) 1–38.

[50] G. J. McLachlan, T. Krishnan, The EM Algorithm and Extensions (Wiley Series in Probability and Statistics), Wiley-Interscience, 1997.

[51] N. Friedman, Learning belief networks in the presence of missing values and hidden variables, in: Proc. 14th Int. Conf. Machine Learning, 1997, pp. 125–133.

[52] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The WEKA data mining software: an update, SIGKDD Explor. 11 (1) (2009) 10–18.

[53] A. K. McCallum, Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, http://www.cs.cmu.edu/ mccallum/bow (1996).

[54] J. B. Lovins, Development of a stemming algorithm, Mech. Trans. Comput. Ling. 11 (1968) 22–31.

[55] J. D. Rodríguez, A. Perez, J. A. Lozano, Sensitivity analysis of k-fold cross validation in prediction error estimation, IEEE Trans. Pattern Anal. Mach. Intell. 32 (3) (2010) 569–575.

[56] A. Menon, H. Narasimhan, S. Agarwal, S. Chawla, On the statistical consistency of algorithms for binary classification under class imbalance, in: Proc. 30th Int. Conf. Machine Learning, 2013, pp. 603–611.

[57] H. He, E. Garcia, et al., Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.

[58] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, 2nd Edition, Springer Series in Statistics, 2009.

[59] A. Papoulis, S. U. Pillai, Probability, Random Variables, and Stochastic Processes, 4th Edition, McGraw-Hill Education, 2002.

[60] V. S. Sheng, F. Provost, P. G. Ipeirotis, Get another label? improving data quality and data mining using multiple, noisy labelers, in: Proc. 14th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, ACM, 2008, pp. 614–622.

[61] D. Cohn, L. Atlas, R. Ladner, Improving generalization with active learning, Mach. Learn. 15 (2) (1994) 201–221.

[62] N. V. Chawla, A. Lazarevic, L. O. Hall, K. W. Bowyer, Smoteboost: Improving prediction of the minority class in boosting, in: Proc. 7th European Conf. Principles and Practice of Knowledge Discovery in Databases, Springer, 2003, pp. 107–119.

[63] J. Zhang, X. Wu, V. S. Sheng, Imbalanced multiple noisy labeling, IEEE Trans. Knowl. Data Eng. 27 (2) (2015) 489–503.