

Automatic Inconsistency Treatment in Case-Based Reasoning Systems

Vera Lúcia M. Falquete¹, Júlio C. Nievola¹ and Celso A.A. Kaestner¹

¹Pontifical Catholic University of Paraná
Graduate Program in Computer Science
Rua Imaculada Conceição 1155, 80215-901 - Curitiba, Brazil

veralucia@jsol.com.br, {nievola, kaestner}@ppgia.pucpr.br

Abstract. *Inconsistency normally arises when using Case-Based Reasoning (CBR) systems. As these systems aim to solve problems using previously recorded situations, real applications frequently present cases that, even starting from similar premisses, lead to different conclusions. In this work we propose an automatic way of dealing with the inconsistency problem in CBRs. We employ the Evidential Logic (EL) formalism, a special case of the Paraconsistent Logics, which admit inconsistent but non-trivial theories. In EL belief and disbelief evidential factors are associated to formulas or, in CBRs context, to each case. We automatically calculate these factors by applying the well-known Machine Learning Naïve-Bayes algorithm to the case base. The overall proposal is tested in a CBR platform that employs the k -Nearest Neighbors (k -NN) algorithm: the solution of an unseen case is obtained as the “closest” one in the case base given by k -NN. We test our proposal using six different rules for combining belief and disbelief factors and the distances given by k -NN, in nine databases from the UCI Machine Learning repository. Obtained results indicate the utility of our proposal in practical applications.*

Keywords: Case-base reasoning, Inconsistency treatment, Paraconsistent Logic.

1. Introduction

Case-Based Reasoning (CBR) is a problem solving technique that attracts a lot of attention, specially due to its great potential use in real applications. CBR means reasoning based on previous examples, where the system expertise is embodied in a database of past cases, instead of being encoded in expert-system classical rules style [1], [2], [17]. Case-based reasoning tends to please users because they feel happier with examples rather than with long reasoning lines and obscure conclusions. Also, a case library can be a powerful corporate resource, allowing everyone to access a corporate case library when handling a new problem.

In general terms, in a CBR the past cases are stored in a database, where each case typically consists of a description of the problem and its solution. To solve a new problem we search the case base for similar cases. The retrieved cases are used to suggest a solution, obtained by combining the solutions of these cases; the suggested solution is reused and tested for success. If necessary, the final solution is revised, and finally the current problem and the final solution can be included in the case base.

Inconsistency is a phenomenon that can be defined in several ways. Roughly speaking a set of assertions is inconsistent if the assertions cannot all be true, or if they are contradictories or contraries [4]. Inconsistency can be frequently found in CBRs, because in real applications, and specially for large bases, we can find cases for the same problem with contradictory solutions.

Racine and Yang [16] analyze the CBR inconsistency problem in detail, identifying two main types of inconsistencies: (a) an *intra-case* inconsistency occurs when a case contradicts itself internally, causing a data integrity violation, such as a domain exception; (b) an *inter-case* inconsistency occurs when cases present a contradiction between themselves. Our proposal deals specifically with this second type of inconsistencies, which were recognized by Racine and Yang as more difficult to deal with.

A problem that is directly related to the inconsistency problem is the continuous to discrete transformation process. In several situations it is necessary to convert continuous attribute values to discrete ones. This can be necessary because of the employed algorithm or due to semantic nature of the real application [14]. We argue that a contradiction depends strongly on the *granularity* of the continuous to discrete transformation process.

In formal logic a theory is inconsistent if it includes a formula and its negation. The main problem with inconsistency is that such theories, from the point of view of the classical logic, are trivial, that is, any formula is also a theorem, causing a collapse in the deductive process [13]. As the cases can be considered as logical formulas, an inter-case inconsistency corresponds to an inconsistent theory. Paraconsistent logics [8], [5] propose a formal solution for inconsistency: they are logic formalisms that admit contradictory but non-trivial theories. Evidential Logical Programming (EL) [19] is a special paraconsistent logic where belief and disbelief factors are associated to each formula.

In this paper we propose an automatic formalism to treat inconsistency on CBRs, based on the EL formalism. Belief and disbelief factors are automatically calculated by applying the Naïve-Bayes well-known machine learning algorithm to the case base. In order to evaluate our proposal we use a CBR platform that employs the k -Nearest Neighbors (k -NN) algorithm: the solution of an unseen case is obtained as the “closest” one in the case base given by k -NN. We test our proposal using six different rules for combining belief and disbelief factors and the distances given by k -NN, in nine databases from the UCI Machine Learning repository [6].

Our paper is organized as follows: section 2 presents the inconsistency problem and the continuous to discrete transformation process in more detail; the EL formalism and the procedure employed to compute belief and disbelief factors are explained in section 3; section 4 describes the CBR platform, the employed methodology and executed tests; finally section 5 presents our conclusions and future research directions.

2. The inconsistency problem in CBRs

In general terms, we can define a CBR procedure as follows. The case base repository (CB) is formed by a set of cases. Each case typically consists of a pair (p, s) where p is the description of the problem and s is the description of a solution for p . Both p and s can be expressed by logical formulas, the same way employed in relational algebra [18]. To solve an unseen problem u , u is matched against the problems of the cases in CB using

a similarity metric d ; similar cases, the ones in the set $S = \{(p, s) \in \text{CB} : d(u, p) < \epsilon\}$ for a proximity parameter ϵ , are retrieved. The retrieved cases – the ones in the set S – are used to suggest a solution. This solution is usually obtained applying a combination function Φ on S , which produces the final solution s_f . Usually the solutions found in S are reused and tested for success. If necessary, the final solution s_f is revised, and finally the current problem and the final solution can be included in CB for future application.

Inconsistency, or equivalently, contradictory or contrary assertions, can be frequently found in CBRs. They form inter-case inconsistencies in the terminology of Racine and Yang [16]. Let's suppose that two cases (p_1, s) and $(p_2, \sim s)$ are in CB, that is, two cases with different solutions. If the similar case set is $S = \{(p_1, s), (p_2, \sim s)\}$ for a new case u , an inconsistency appears: what solution the system must present to the user? Inconsistency, as can be easily noted, depends strongly on the proximity parameter ϵ .

A problem that is directly related to the inconsistency problem is the continuous to discrete transformation process. In several situations it is necessary to convert continuous attribute values to discrete ones. This can be necessary because of the employed algorithm – for example, the Naïve-Bayes algorithm and some decision trees need discrete values – or due to semantic nature of the real application [14].

Let us analyze the situation with two cases (p_1, s) and $(p_2, \sim s)$, p_1 and p_2 being described by attribute-value pairs $p_1 = \langle a, v_1 \rangle$ and $p_2 = \langle a, v_2 \rangle$. We can consider that the two values v_1 and v_2 are distinct: in this case we do not have an inter-case inconsistency. But if we apply a continuous to discrete transformation process v_1 and v_2 can be transformed in the same discrete value v_d , leading to the cases $(\langle a, v_d \rangle, s)$ and $(\langle a, v_d \rangle, \sim s)$ and therefore, to a contradiction !

So, a contradiction can appear depending on the *granularity* of the continuous to discrete transformation process. Finer granularity transformations tend to produce less inconsistent cases than coarser ones. To illustrate this important point, consider the cases $(\langle \text{temperature}, 3.9 \rangle, \sim \text{liquid})$ and $(\langle \text{temperature}, 3.1 \rangle, \text{liquid})$ in a CB: originally they do not lead to a contradiction, but if 3.9 and 3.1 temperature values are transformed in a discrete value like *low*, a contradiction arises.

We note that continuous to discrete transformation and the use of the similarity parameter ϵ in the retrieving process are two faces of the same key element that can lead a CBR system to inconsistency. For a new unseen case both can produce similar cases with contradictory solutions.

3. Automatic Inconsistency Treatment and Evidential Logic Programming

As previously explained, in formal logic a theory is inconsistent if it includes a formula and its negation. If we employ the classical approach to such theories the deductive process collapse: every well-formed formula is also a theorem of the theory [13].

The same situation takes place in the CBR case. For example, the cases $(\langle a, v_d \rangle, s)$ and $(\langle a, v_d \rangle, \sim s)$ previously presented can be translated – with some liberality in the notation – into the logic formulas $(\langle a = v_d \rangle \rightarrow s)$ and $(\langle a = v_d \rangle \rightarrow \sim s)$, which in the classical context lead to the clauses $c_1 : \sim (\langle a = v_d \rangle) \vee s$ and $c_2 : \sim (\langle a = v_d \rangle) \vee \sim s$. If a problem $u: \langle a = v_d \rangle$ must be solved, resolution principle [12] applied to $\{c_1, c_2, u\}$ causes a contradiction.

Paraconsistent logics are a family of formalisms constructed to deal with inconsistency. They are logic formalisms that admit contradictory but non-trivial theories [8]. This means that assertions p and $\sim p$ may be present in a theory, but inconsistency is in some way “restricted” to them, without causing the complete collapse of the deduction process. That is, in a paraconsistent logic we can have a theory which is inconsistent but still possess formulas which are theorems and formulas which are not.

Several paraconsistent logics are *annotated*, that is, they associate annotation constants to each atomic formula and possess a propagation mechanism that associates valuations to each formula [5]. Evidential Logical Programming (EL) [19] is a special annotated paraconsistent logic where belief and disbelief factors are associated to each formula.

An atomic formula in EL is a triple $\langle p, bf, df \rangle$ where p is the atomic proposition, bf is the belief factor and df is the disbelief factor.

The semantics of the triple is the following: bf indicates how much the truth of a proposition p is “believed”, and df indicates how much the negation of p is believed. bf and df usually are real numbers between 0 and 1. Therefore we can have the following extreme cases:

- a triple $\langle p, 1, 0 \rangle$ represents a certainly truth proposition p ;
- $\langle p, 0, 1 \rangle$ represents that p is false (equivalently to the triple $\langle \sim p, 1, 0 \rangle$);
- $\langle p, 0, 0 \rangle$ indicates complete ignorance about p ; and
- $\langle p, 1, 1 \rangle$ indicates a inconsistency about p , or that p and $\sim p$ are equally believed.

In general there are infinite possible annotations for one premise, allowing to quantify the belief of the involved knowledge. In the case of complex formulas EL possess behavior rules that allow to calculate bf and df associated to the formula, depending on the involved connectives and on the factors of the atomic components. The reader can refer to the work of Subrahmanian [19] to a more complete description of the procedures involved.

But how to compute belief and disbelief factors? In expert systems, where EL was employed, this work was done directly by the human expert [19]. In our case we search for an automatic procedure to compute bf and df . We propose to use a machine learning approach, where the desired factors are calculated directly from the cases on CB. In fact we propose to use the Naïve-Bayes algorithm and formulas from the well-known MYCIN system [7]. This methodology was also employed by Enembreck [9] in a pattern recognition application.

We consider that the problem description of a case is given by a set of attribute-value pairs. For Naïve-Bayes to be applied all attribute values are discretized, following the algorithm described in Mitchell [14], with a minor modification to incorporate a minimal width for the continuous domain that correspond to each discrete value.

Naïve-Bayes [10] is a supervised machine learning algorithm primarily used in classification tasks. This classifier supposes that the probability of the conjunctive evidence $e = (\langle a_1, v_1 \rangle, \dots, \langle a_n, v_n \rangle)$ is given by the product of the probability of the occurrence of each one of the values of the attributes, meaning that the individual evidences are independent.

That is, the probability of e belonging to \hat{c} is given by:

$$p(e|\hat{c}) = p(c) * \prod_{i=1}^n p(\langle a_i, v_i \rangle | \hat{c}) \quad (1)$$

The independency hypothesis allows the calculation of the joint evidence probability to be computed as the product of the individual probabilities of each attribute [14]. Using the Bayes Rule we have:

$$p(c|e) = \frac{p(c) * \prod_{i=1}^n p(\langle a_i, v_i \rangle | \hat{c})}{\sum_{j=1}^k p(c_j * \prod_{i=1}^n p(\langle a_i, v_i \rangle | c_j))} \quad (2)$$

Therefore, the model learnt by the classifier is simply the set of probabilities $p(c_i)$ and $p(c_i|e)$. These probabilities can be directly calculated from the data, estimating probabilities by data frequencies. For example, in order to calculate the probability $p(\langle a_i, v_i \rangle | \hat{c})$ we simply divide the number of occurrences of the v_i value for the attribute a_i by the number of cases in the class \hat{c} . The final classification decision is that e belongs to the class with maximum a posteriori probability, that is $\hat{c}_{final} = argmax_i \{c_i\}$.

In our proposal the Naïve-Bayes is employed to calculate the belief and disbelief factors. In the CBR context the *classes* correspond to the *solutions* in the CB. We employ two formulas inspired in the early MYCIN system [7, 15].

We use $bf[c, e]$ to indicate the increase in the belief of choosing the class c given the example e , and $df[c, e]$ indicates the increase in the disbelief of choosing c given e . In the MYCIN terminology these factors are called certainty and uncertainty factors. As in MYCIN, $p(c_i)$ indicates the probability of the expert believing in the class (hypothesis) c_i , and $(1 - p(c_i))$ indicates his disbelief in c_i , with no evidences [7]. If $p(c|e)$ is bigger than $p(c)$ then the evidence e increases the expert's belief in c , and decreases his disbelief. In this case the increase in bf can be computed by:

$$\frac{max[p(c|e), p(c)] - p(c)}{(1 - p(c))} \quad (3)$$

Reciprocally if $p(c|e)$ is lower than $p(c)$ then e increases the expert's disbelief on c , and decreases his belief. The disbelief factor can be computed by the equation:

$$\frac{min[p(c|e), p(c)] - p(c)}{-p(c)} \quad (4)$$

It is important to observe that an evidence e cannot at the same time to be for and against the hypothesis c . So when $bf[c, e] > 0$ then $df[c, e] = 0$ and when $df[c, e] > 0$ then $bf[c, e] = 0$. To summarize, for a class c and a case e we have:

$$bf[c, e] \begin{cases} 1 & \text{if } p(c) \\ \frac{\max[p(c|e), p(c)] - p(c)}{1 - p(c)} & \text{otherwise} \end{cases} \quad (5)$$

$$df[c, e] \begin{cases} 1 & \text{if } p(c) \\ \frac{\min[p(c|e), p(c)] - p(c)}{-p(c)} & \text{otherwise} \end{cases} \quad (6)$$

These formulas allow us to compute the belief and disbelief factors for each case e in CB, for a given solution c . This will be employed to weight the retrieved cases by the CBR system for an unseen case u , as explained in the next section.

4. The CBR test-bed platform and experimental results

In order to test and evaluate our proposal we construct a basic CBR platform based on the k -Nearest Neighbors (k -NN) algorithm. The k -NN is an instance-based machine learning algorithm also used in classification [14]. The traditional algorithm associates an unseen case u to the most frequent class of the ones associated to the k “closest” cases to u in the database.

The algorithm can be employed straightforward in CBRs. As previously described in a CBR we must select previous cases that are “similar” to the new case. Therefore the final solution to be presented by the system corresponds to the final class assigned by the classification procedure. If we employ k -NN we use a distance as similarity measure. That is, given two cases $e_1 = (p_1, s)$ e $e_2 = (p_2, s')$ and considering that the problem descriptions are in the form $p_j = \langle (a_1, v_{1j}), (a_2, v_{2j}) \dots (a_n, v_{nj}) \rangle$ for $j = 1, 2$, the distance $d(e_1, e_2)$ can be computed by the classical Hamming distance:

$$d(e_1, e_2) = \sum_{i=1}^n \delta(v_{i1}, v_{i2}) \quad (7)$$

where $\delta(v_{i1}, v_{i2})$ is 0 if $v_{i1} = v_{i2}$ for an attribute a_i and 1 otherwise. This distance is employed because in our test platform all attributes values are discretized.

Hence, for a new case u , the k -NN algorithm inspect the whole CB, computing $d(u, e)$ for each case e in the base. The obtained distances are sorted in ascending order. The k first examples – the k NN – are selected and the most frequent of their solutions are assigned as the suggested solution for u . Usually k is odd in order to avoid a draw. The complexity of the k -NN algorithm is the $O(mn)$, where m is the number of attributes and n are the number of cases of the base [20]. This complexity can be reduced if some techniques are used by introducing more information in the CB [11].

The traditional k -NN algorithm considers that each neighbor contributes equally to the final class assignment decision. In order to treat inconsistency, in our test platform we propose to weight the distance between the new case u and the cases e_j in the CB using the belief and disbelief factors of the case e_j . We can interpret that each voting neighbor (case) e_j contributes to the final solution of a given problem u with a strength proportional to its bf and df .

We have employed six options of voting formulas for the k -NN algorithm, as shown in Table 1. Option 1 represents the traditional k -NN classifier, and works as a baseline for comparisons.

Table 1. k -NN voting formulas for distance d , belief factor bf , disbelief factor df

Option	Weighted distance
<i>Baseline</i>	1
1	bf
2	$1/d$
3	$bf * (- df)$
4	$1/(- bf) * df * d$
5	$bf * (- df) * (/d)$

The final class (solution) is obtained by the following procedure, for a given new case u and $e_j \in CB$:

1. Find the k nearest neighbors set $NN = \{e_1, e_2 \dots e_k\}$ of u using the standard distance d ;
2. Weight the corresponding classes $\{s_1, s_2 \dots s_k\}$ of the cases in NN according to the formula in use (note that d, bf and df of $s_i, i = 1 \dots k$ can be employed);
3. For each voted class, sum its weights;
4. Take the class with larger sum as the final classification verdict.

In order to put our CBR system to work it is necessary to execute a preprocessing phase. This phase embodies unknown values treatment, the discretization procedure and the computation of the belief and disbelief factors of each case p_j in CB by the Naïve-Bayes algorithm. After training our CBR takes an unseen case u and employing the k -NN algorithm plus the combination given by the current voting formula obtains the most adequate solution to be presented to the user.

Our proposal was tested in nine databases available in the UCI machine learning repository [6]. These databases were chosen because they contain a series of standardized attributes, and one separate attribute – the one that correspond to the class of the case – which can be considered as the solution of the corresponding case. They also contain several inconsistent examples, which simulate inconsistency cases in the CBR context. Table 2 introduces a summary of the main features of the bases used in the tests. For the Naïve-Bayes algorithm (to calculate belief and disbelief factors) we have employed a holdout procedure with 70 % of the CB for training and 30 % for testing.

In order to establish suitable evaluation metrics, we have employed the well-known precision and recovery information retrieval measures [3]. Precision (P), which represents the quality of an algorithm in hitting the correct classification regarding the classified cases, is calculated by:

$$P(c) = \frac{\# \text{ of correctly classified cases for class 'c'}}{\text{total \# of cases classified by the CBR as class 'c'}} \quad (8)$$

Recall (R), which represents the quality of the algorithm in hitting the correct classification regarding the total of cases of the class in the case base, is calculated by:

$$R(c) = \frac{\# \text{ of correctly classified cases for class 'c'}}{\text{total \# of cases of the class 'c' in CB}} \quad (9)$$

Table 2. Main features of the used bases

Base	# of Classes	# of Instances	# of Attributes	# of continuous attributes	# of discrete attributes	missing values
Anneal	6	798	38	6	32	yes
Breast-Cancer	2	699	10	0	10	yes
Chess	2	3196	36	0	36	no
Dermatology	6	366	34	1	33	yes
Haberman	1	306	3	3	0	no
Iris	3	150	4	4	0	no
Tic-Tac-Toe	2	958	9	0	9	no
Wine	3	178	13	13	0	no
Zoo	7	101	17	0	17	no

As shown, in these formulas precision and recall are calculated for each class. In order to evaluate overall classification we compute the average values for all classes, that is \bar{P} and \bar{R} for all classes. Table 3 presents the results of precision and recall for each relevant voting formula. Best results are shown in bold. Table 4 presents the precision and recall gain in relation to the baseline for each voting formula.

An initial analysis reveals that the results of precision and recall are worse in the base Haberman. When observing the attributes of the base Haberman, it becomes clear the poor performance in this case. There are just three attributes, and the attribute “year of operation” are not strongly correlated to the corresponding class. It is probable that the attribute “quantity of auxiliary nodules” introduce a correlation with the patient’s time of survival, and that the attribute “age of the patient” also has a correlation with the classification in smaller degree. The poor performance, in this case, indicates more the little correlation between attributes of the base and the final class than the efficiency of the algorithm.

The bases Zoo and Tic-Tac-Toe have intermediary indices (from 80% to 90%). The base Tic-Tac-Toe introduces the final position of this game, having as class a boolean variable that represents the victory or not of the first player. In case of a draw, to both players is assigned the same class, that is, the victory. As the attributes are positions, it is not surprising that the best result is obtained with option 2, which only uses the distance. Option 4, which can assign a large weight for the belief factor, has the worst precision and recall indices.

The base Zoo contains a correlation of input symbolic attributes characteristic of animals species and their taxonomy. Taking in account that there are twenty possible classes and seventeen predictor attributes, one can consider that the results are satisfactory. In this base surprisingly the option 4 has produced the best results. In fact, the classification taxonomy of the animals tends to use few attributes to determine a class. For example, the presence of feathers is mandatory for classifying an animal as a bird; analogously the milk secretion is mandatory for the class mammal. Because of this particularity the option 4, that gives priority to cases with elevated belief factors, produces good results.

Table 3. Average precision (\bar{P}) and average recall (\bar{R}) for the voting options

Base	Option 1	Option 2	Option 3	Option 4	Option 5	Baseline
Anneal	0.9934	0.9957	0.9934	0.9900	0.9932	0.9927
	0.9313	0.9299	0.9313	0.9059	0.9386	0.9003
Breast-Cancer	0.9646	0.9646	0.9646	0.9607	0.9646	0.9633
	0.9599	0.9599	0.9599	0.9626	0.9599	0.9600
Chess	0.9156	0.9590	0.9156	0.9071	0.9209	0.9562
	0.9145	0.9585	0.9145	0.9075	0.9198	0.9656
Dermatology	0.9654	0.9654	0.9654	0.9573	0.9654	0.9441
	0.9651	0.9651	0.9651	0.9578	0.9651	0.9632
Haberman	0.6138	0.7884	0.6138	0.6424	0.7166	0.6952
	0.6650	0.6905	0.6650	0.6465	0.6784	0.6786
Iris	0.9280	0.9253	0.9280	0.9165	0.9331	0.9241
	0.9290	0.9253	0.9290	0.9182	0.9327	0.9217
Tic-Tac-Toe	0.8632	0.9277	0.8632	0.8105	0.8632	0.9277
	0.8550	0.9177	0.8550	0.7998	0.8550	0.9178
Wine	0.9587	0.9587	0.9587	0.9583	0.9587	0.9587
	0.9494	0.9494	0.9494	0.9392	0.9494	0.9494
Zoo	0.7946	0.7946	0.7946	0.8988	0.7946	0.7946
	0.8458	0.8458	0.8458	0.9083	0.8458	0.8458

The remaining bases (Chess, Iris, Wine, Dermatology, Breast-Cancer and Anneal) produce high indices of precision and recall (above of 90%). In spite of the great number of missing values in the base Anneal, the obtained precision (0.9931) is very good. In the Chess database, which deals with positional portraying at the end of chess game, the results can also be considered good. In this case, as in the Tic-Tac-Toe CB, the best results are given by the option 2 formula, which considers only the distance; the second best result is given by the option 5 formula, which considers the distance and the belief and disbelief factors. If we observe the characteristics of the bases Anneal, Breast-Cancer, Dermatology and Wine, we can argue that the proposed solution has better performance in bases with elevated number of attributes and few classes, and in bases where the set of attributes is a good predictor of the final class, as it would be, according to the general knowledge in the area of machine learning. For example, in the Wine base, the source of a wine is based on complex combinations of attributes related to physical and chemical properties. In the base Dermatology, which is very complex, several attributes must agree for the diagnosis, because the symptomatology of the involved diseases has many common points.

5. Conclusions

Dealing with inconsistency in CBR systems is a problem that has not been frequently treated. In this work we propose an automatic way of dealing with inconsistencies, based on the use of evidential logic programming, a special case of paraconsistent logics which admit inconsistent but non-trivial theories.

We associate belief and disbelief factors to each case in a case base, which are

Table 4. Upgrade in \bar{P} and \bar{R} in relation to the conventional k-NN

Base	Option 1	Option 2	Option 3	Option 4	Option 5
Anneal	0.07%	0.30%	0.07%	- 0.27%	0.05%
	3.40%	3.32%	3.40%	0.65%	4.28%
Breast-Cancer	0.00%	0.00%	0.00%	- 0.40%	0.00%
	-0.01%	-0.01%	-0.01%	0.27%	-0.01%
Chess	- 4.24%	0.29%	- 4.24%	- 5.13%	- 3.69%
	- 5.30%	-0.74%	- 5.30%	- 6.02%	- 4.75%
Dermatology	0.13%	0.13%	0.13%	- 0.70%	0.13%
	0.20%	0.20%	0.20%	0.55%	0.20%
Haberman	- 11.65%	13.4%	- 11.65%	- 7.54%	3.13%
	1.98%	1.76%	1.98%	- 4.71%	-0.01%
Iris	0.27%	- 0.02%	0.27%	0.97%	0.82%
	0.79%	0.39%	0.79%	0.37%	1.19%
Tic-Tac-Toe	- 6.95%	0.00%	- 6.95%	- 12.63%	- 6.95%
	- 6.84%	-0.01%	- 6.84%	- 12.85%	- 6.84%
Wine	0.00%	0.00%	0.00%	0.04%	0.00%
	0.00%	0.00%	0.00%	-1.07%	0.00%
Zoo	0.00%	0.00%	0.00%	13.1%	0.00%
	0.00%	0.00%	0.00%	7.38%	0.00%

obtained directly from the base itself. We apply the well-known Naïve-Bayes algorithm to calculate these factors, and their computation formulas follow the same methodology used in the old MYCIN system.

As a CBR platform we have employed a basic system that use the k -NN algorithm to recover the similar cases for an unseen new case. The k -NN employs a distance metric among the involved cases in order to produce the final solution. In our proposal the original distances computed by k -NN are weighted by functions of the belief and disbelief factors.

We tested our proposal in nine machine learning databases of the UCI, employing six different options for the voting formula. The overall performance is calculated using precision and recall measures. The best results were obtained for complex bases, with elevated number of cases and attributes, using the formulas $1/d$ and $bf * (- df) * (/d)$ for a distance d , belief factor bf and disbelief factor df .

The proposal has shown to be robust and and reliable, allowing an automatic detection and treatment of inconsistencies. Therefore, this work empirically shows that EL can be used to treat and deal with inconsistencies in classification-like tasks, and improve CBR performance concerning this point.

This work is limited to our CBR platform based on the k -NN algorithm. It is possible to employ the same ideas and the computed belief and disbelief factors of each case in other retrieval algorithms, such as decision trees, genetic algorithms or neural nets. In addition, more complex formulas involved bf , df and d can be used, and tested in real case base environments in order to empirically evaluate the real potential of our proposal.

References

- [1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations and systems approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.
- [2] B. P. Allen. Case-based reasoning: Business applications. *Communications of the ACM*, 37(3):40–42, 1994.
- [3] R. Baeza-Yates and Ribeiro-Neto B. *Modern Information Retrieval*. Addison-Wesley, 1999.
- [4] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Lecture Notes in Computer Science*, 287(1):340–360, 1987.
- [5] H. A. Blair and V. S. Subrahmanian. Paraconsistent foundations for logic programming. *Non-Classical Logic*, 5(2):46–73, 1988.
- [6] C. L. Blake and C. J. Merz. The UCI repository of machine learning databases, 1998.
- [7] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, USA, 1984.
- [8] W.A. Carnielli, M.E. Coniglio, and I.M.L. D’Ottaviano (Eds.). *Paraconsistency: The Logical Way to the Inconsistent*. Marcel Dekker, 2002.
- [9] F. Enembreck. A paraconsistent system for automatic signature verification (*in portuguese*). Master’s thesis, Pontifical Catholic University of Paraná, Brazil, 1999.
- [10] E. Frank, H. Mark, and Bernhard P. Locally weighted naïve-bayes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 249–256. Morgan Kaufmann, 2003.
- [11] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 13 th annual ACM symposium on Theory of Computing*, pages 604–613. ACM Press, 1998.
- [12] J.W. Lloyd. *Logic Programming*. Springer-Verlag, Berlin, 1987.
- [13] E. Mendelsson. *Introduction to Mathematical Logic*. Chapman-Hall, 3rd. Edition, 1987.
- [14] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [15] R. E. Neapolitan. *Probabilistic Reasoning In Expert Systems: Theory and Algorithms*. Wiley-Interscience, USA, 1990.
- [16] K. Racine and Q. Yang. On the consistency management of large case bases: the case for validation. In *AAAI Technical Report - Verification and Validation Workshop*, Burnaby, Canada, 1996.
- [17] R. C. Schank. *Dynamic Memory*. Lawrence Erlbaum Associates, New Jersey, 1989.
- [18] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill International Editions, 1997.
- [19] V. S. Subrahmanian. Towards a theory of evidential reasoning in logic programming. In *The European Summer Meeting of the Assoc. for Symbolic Logic*, Spain, July 1987.
- [20] P. Tsaparas. Nearest neighbor search in multidimensional spaces. Technical Report 319-02, Dept. of Computer Science, University of Toronto, 1999.