Probabilistic Decision Trees using SVM for Multi-class Classification

Juan Sebastian Uribe, Nazih Mechbal, Marc Rébillat, Karima Bouamama, Marco Pengov

Probabilistic Decision Trees using SVM for Multi-class Classification

Juan Sebastian Uribe[1], Nazih Mechbal[1], Marc Rébillat[1], Karima Bouamama[2], Marco Pengov[2]

Abstract— **In the automotive repairing backdrop, retrieving from previously solved incident the database features that could support and speed up the diagnostic is of great usefulness. This decision helping process should give a fixed number of the more relevant diagnostic classified in a likelihood sense. It is a probabilistic multi-class classification problem. This paper describes an original classification technique, the Probabilistic Decision Tree (PDT) producing *a posteriori* probabilities in a multi-class context. It is based on a Binary Decision Tree (BDT) with Probabilistic Support Vector Machine classifier (PSVM). At each node of the tree, a bi-class SVM along with a sigmoid function are trained to give a probabilistic classification output. For each branch, the outputs of all the nodes composing the branch are combined to lead to a complete evaluation of the probability when reaching the final leaf (representing the class associated to the branch). To illustrate the effectiveness of PDTs, they are tested on benchmark datasets and results are compared with other existing approaches.**

I. INTRODUCTION

With more and more complex vehicle architectures, failure possibilities have grown increasingly and fixing solutions are becoming harder to find quickly. During a car repairing session, a useful diagnosis method consists in applying to the damaged vehicle a technical solution corresponding to a previously solved incident. Hence, collecting known incidents and storing them is a key issue to improve the after-sales services. However, the strategy to use in order to deal effectively with an unknown (and possibly unreferenced) failure is still an open problem in the diagnosis step. As some data regarding past failures are available, we can expect that an automatic classifier could give, knowing some information about the encountered failure and with a level of confidence, the more relevant procedures that could be followed to repair the vehicle. Formally, we are interested in solving a multi-class data classification problem in a manner that produces confidence probabilities associated to each possible class.

Actually, there exist two main types of classifiers: hard and soft [1, 2]. Hard classifiers, such as support vector machine (SVM) and all the associated multi-class techniques, build a frontier between classes. They only label new unknown points with the class associated to the side of the frontier in which they fall, without giving any idea of the certitude of the decision or the degree of membership to that class. These classifiers are very appealing, because in general they tend to give very accurate predictions. On the opposite, soft classifiers like Logistic Regression (LR) [3] are able to build probability estimations for the belonging to all the classes, and then with this information they choose the most likely class. We are thus interested, in a multi-class context, by the probabilities estimation offered by soft classifiers while keeping the hard classifiers proved performances [4].

To reach this objective, hard classifiers need to be first adapted to a multi-class context. Support Vector Machines (SVM) are a powerful tool for data classification [5]. Unfortunately, they were originally designed for bi-class decision problems and their extension to multi-class problems is not straightforward and is still an on-going research issue [6]. Classic SVM multi-class approaches such as *"one-against-one"* (OvO) [7], *"one-against-all"* (OvA) [8] or Diagram Acyclic Graph (DAG) [9] have shown adequate results but don't take into account the structure and the distribution of the data when separating the classes. To overcome this drawback, Madzarov [10] came up with a simple and intuitive approach based on building a binary decision tree. By selecting specific features, such as the distance between gravity centers of the different classes, an automatic graph is generated where at each node a bi-class SVM is trained. However, such multi-class hard classifiers only provide one predicted class without any associated score indicating the confidence of the classification.

In order to compute confidence probabilities in a bi-class context, Logistic Regression can be an adequate choice. This soft classifier is widely used in economics [11] and biology [12]. This method is simpler to use than SVM because it does not have parameters to adjust and it also has the advantage of providing direct probability outputs. But in general, Logistic Regression based classifiers are less accurate than well-tuned SVM classifiers. Nevertheless, it is also possible to introduce confidence probabilities into hard classifiers. In a bi-class context, Platt [4] proposed a method for extracting probabilities $p(class|input)$ from SVM outputs to be used for classification post-processing. The approach consists in training the parameters of a sigmoid function to map the SVM outputs into probabilities. The underlying idea of Probabilistic SVM classifier (PSVM) is that as the distance from an example to the frontier is larger, the example is closer to that class, which implies that the example will very likely belong to that class. Adapting

[1] J. S. Uribe, N. Mechbal and M. Rebillat are with the PIMM UMR CNRS, Arts et Métiers ParisTech, 151 boulevard de l'Hôpital, 75013 Paris France. (juan-sebastian.uribe-echeverri@ensam.eu ; nazih.mechbal@ensam.eu ; marc.rebillat@ensam.eu)

[2] K. Bouamama and M. Pengov are with PSA Peugeot Citroën, Route de Gisy, CC VV1404, 78940 Vélizy-Villacoublay. (karima.bouamama1@mpsa.com ; marco.pengov@mpsa.com)

Platt's method to a multi-class context, it is thus in principle possible to build the confidence index that we need while keeping the demonstrated performances of hard classifiers.

On the basis of Madzarov [10] and Platt [4] algorithms, we present Probabilistic Decision Trees (PDT) as an original approach to the multi-class probabilistic classification problem. The proposed PDT algorithm takes advantage of the decision tree architecture and of the classification posterior probability provided by PSVM. The PDT will provide fast classification (logarithmic complexity) along with associated posterior probabilities $p(class|input)$. At each node of the PDT, SVM classification associated with a sigmoid function is performed to estimate the probability of membership to each sub-group. A probability function is then built for each leaf, by following the path that the PDT has generated for it. In the context of automobile diagnosis, the PDT is used to generate a decision diagnostic tree where from known incidents database the most probable failure will be provided. The performances of the PDT will be measured on samples from benchmark databases and on an artificial database emulating actual car breakdown reports database. These tests show that the proposed technique is suitable for the automotive problem.

The article is organized as follows. In section II we present the methodology of automotive diagnosis. In section III we review probabilistic classification methods, including the related work and the techniques that inspired our approach. In section IV we present the PDT algorithm and in section V some applications conducted with artificial and actual databases are described. Finally, the conclusions and perspectives of our method are outlined in section VI.

II. Information Retrieval in Automotive Diagnosis

A useful diagnosis method commonly used in car workshops is to apply to the damaged vehicle a technical solution corresponding to a previously solved incident. For that purpose, troubleshooting and repair information are stored in specific databases that are frequently updated. The objective of the present work is to elaborate an efficient algorithm that extract from these databases features that could support and speed up the diagnostic. This decision helping process should give a fixed number of the more relevant diagnostics classified in a maximum likelihood sense. The strategy to use in order to deal efficiently with an unknown (and possibly unreferenced) failure is an open problem in the diagnosis step that needs to be addressed.

PSA Peugeot Citroën databases are very rich and a lot of information may be extracted from them. But at the same time they are very large and thus difficult to process. For this reason an automatic classifier is the most convenient method to exploit these databases. Feature selection is a very important pre-processing step in data mining. Clustering the problem according to the damage location (electronical system, mechanical system, etc) is very helpful and may potentially simplify the classification task. An automatic classifier that is able to do so should thus be very efficient.

III. Probabilistic Classification

In this section we introduce the classification methods that give a notion of probability like Logistic regression and Probability-SVM. Then we present some methods for solving multiclass problems, and specially the binary decision tree.

1) Logistic Regression

This regression method models the probability of membership to a class $y \in \{-1,1\}$ with a sigmoid function whose inputs are linear combinations of the example x attributes [3].

$$p(y=-1|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}}} \quad (1)$$

$$p(y=1|\boldsymbol{x}) = 1 - p(y=0|\boldsymbol{x}) \quad (2)$$

The model selection is conducted by searching for the coefficients vector w which maximizes the likelihood function, defined as:

$$l(\boldsymbol{w}) = \prod_{i=1}^{all\ data} p(\boldsymbol{x_i})^{y_i}(1-p(\boldsymbol{x_i}))^{1-y_i} \quad (3)$$

where $p(\boldsymbol{x_i})$ is defined as in Equation 1.

In general, Logistic Regression (LR) is an easy-to-use method as it doesn't have parameters to adjust. We will use it as a benchmark method, to validate the results provided by our technique. Given that LR is a bi-class classifier, we will associate it to a binary decision tree (BDT) technique [10] to solve the multi-class problem. We have called this technique LR-BDT. We will present the binary decision tree afterwards in this same section.

2) Probabilistic Support Vector Machines

Classic support vector machines (SVM) have proved to be a very effective classification method [13]. They are binary linear classification techniques [14] which search for the hyper plane (in the hyperspace of attributes) that separates two classes in a training set. This hyper plane is found by maximizing the so-called margin, which is the distance from the hyper plane to the closest points, denoted support vectors. A common variant of classic SVM, is called *soft margin* and it consists of admitting some misclassified points in the training set for preventing the over fitting problem. Although, we want to avoid too many points being misclassified, thus we impose a penalty C that will penalize every misclassified example. C can take values in the range $0 < C \leq \infty$. A high value of C means a strict classifier that doesn't admit many misclassified points. On the opposite, a small C means a very flexible classifier. Formally, we have a training set $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \dots, (\boldsymbol{x_n}, \boldsymbol{y_n})\}$, where every point $\boldsymbol{x_i} = (\boldsymbol{x_{i1}}, \dots, \boldsymbol{x_{im}})$ has m attributes and one of the two possible labels $\boldsymbol{y_i} = \{-1, 1\}$. A soft margin SVM classifier will label a new unknown point $\boldsymbol{x_t}$ according to the decision function:

$$y(\mathbf{x}_t) = sign((\boldsymbol{w}.\boldsymbol{x}_t) + w_0) \quad (4)$$

where w and b are the hyper plane parameters obtained from the minimization of the cost function in Equation 7 on the training set.

In SVM, kernels are used to project the data into a virtual space where it might be easier to separate them [8]. The main advantage of kernel functions is that the only operation needed to be defined in the new virtual space is the inner product $\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle$. Several kernel functions are used [15]:

- *Linear:* $\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i}^T \boldsymbol{x_j}$
- *Polynomial*: $\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\gamma \boldsymbol{x_i}^T \boldsymbol{x_j} + r)^d$
- *Gaussian* : $\kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(\frac{-\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2}{2\sigma^2})$

Applying a kernel function, the soft margin and the Wolfe dual formulation [16], the SVM problem is presented as:

$$y(\mathbf{x}_t) = sign(f(\boldsymbol{x_t})) \tag{5}$$

where $f(\boldsymbol{x})$ is defined as:

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i u_i \kappa(\boldsymbol{x_i}, \boldsymbol{x}) + w_0 \tag{6}$$

the values of α_i and u_i are found solving the following constrained optimization problem:

$$\begin{cases} \max\limits_{\alpha_i} \left[\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j u_i u_j \kappa(\boldsymbol{x_i}, \boldsymbol{x_j}) \right] \\ 0 \le \alpha_i \le C, \quad i = 1, \dots, m \\ \sum_{i=1}^{m} \alpha_i u_i = 0 \end{cases} \tag{7}$$

In the PDT approach, and for its use in the car diagnostic classification, we choose to use the Gaussian kernel. It only has σ^2 as parameter. A small value of σ^2 will lead to curved hyper plans and a high value will force the hyper plans to be straighter. In [17] it is showed that from some combinations of the hyperparameters (C, σ^2), the Gaussian kernel tends towards the linear kernel, which makes the Gaussian kernel the most general method and one that will work for a large range of datasets. The hyper parameters (C, σ^2) have to be optimized for every classification problem. In [17] an effective technique that we implemented is proposed.

As stated before, SVM produce a value that is not a probability. Indeed, SVM only give a class prediction output that will be either +1 or -1. In order to extract associated probabilities from SVM outputs several approaches have been proposed [4, 8, 18]. We will focus on Platt's approach [4]. Platt [4] proposed a technique that has been largely used in the literature. He builds a sigmoid function between the outputs $f(\boldsymbol{x})$ of the SVM and the probability of membership $p(y = i|\boldsymbol{x})$ to a class i, given the attributes of $\boldsymbol{x}$. A simple bi-class example is shown in Figure 1.

The sigmoid will have the following parametric expression:

$$p\,(y = 1|f(\boldsymbol{x})) = \frac{1}{1 + e^{af(x)+b}} \tag{8}$$

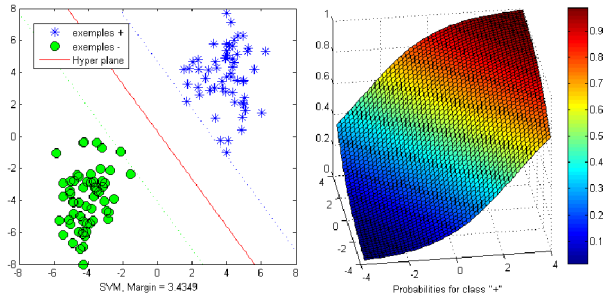

Figure 1 SVM and probability estimation for a 2D binary problem

where a and b are parameters computed from the minimization of the negative log-likelihood function [4]

$$-\sum_i t_i \log p(f(\boldsymbol{x_i})) + (1 - t_i) \log(1 - p(f(\boldsymbol{x_i})) \tag{9}$$

and t_i is the new label of the classes. 1 becomes t_+ and -1 becomes t_-. This relabeling procedure is conducted so that the sigmoid fit will be softer. These new labels are computed using the expressions:

$$t_+ = \frac{N_+ + 1}{N_+ + 2} \qquad t_- = \frac{1}{N_- + 2} \tag{10}$$

where N_+ and N_- are the number of points that belong to class 1 and class 2 respectively.

The PSVM as proposed by Platt [4] uses first a SVM classifier that has to be trained with a training set s_1. Then the sigmoid parameters (a, b) have to be found. To do so, it is recommended [4] to use a second training set s_2. The sigmoid fit requires the inputs $f(\boldsymbol{x})$ and the labels t_i. We use s_2 as a test set for the classifier. Thus we obtain the output values $f(s_2)$. Knowing the labels of s_2 (how many positives ones and how many negatives ones) we can calculate t_+ and t_- for this sub-training set. With values $(f(s_2), t_i(s_2))$ we then find (a, b) by optimizing Eq. (9) using a Newton's method with backtracking, as proposed by [19]. After these two steps, we have an a posteriori probability estimator to the bi-class problem. We can then classify new points and give them an associated probability of belonging to class 1 or 2.

3) From bi-class to multi-class problems

SVM were originally designed for bi-class classification problems, the passage to multi-class problems is still an on-going research area [6]. There are two major approaches to solve this type of problems.

The first and the one that could be more intuitive consist in formulating a cost function with Q hyperplanes (being Q the number of classes in our problem) [5].

$$\begin{cases} \min\limits_{w,\xi} \frac{1}{2} \sum_{m=1}^{Q} (\boldsymbol{w_m}.\boldsymbol{w_m}) + C \sum_{i=1}^{l} \sum_{m \ne y_i} \xi_i^m \\ (\boldsymbol{w_{y_i}}.\boldsymbol{x_i}) + b_{y_i} \ge (\boldsymbol{w_m}.\boldsymbol{x_i}) + b_m + 2 - \xi_i^m \\ \xi_i^m \ge 0, \quad i = 1, \dots, l \quad m \in \{1, \dots, Q\} \setminus y_i \end{cases} \tag{11}$$

For this formulation, the decision is given by

$$f(\boldsymbol{x}) = \max_k (\boldsymbol{w_i}.\boldsymbol{x_i}) + b_i \ , \quad i = 1, \dots, Q \tag{12}$$

This method suffers from the problem that an optimization with so many variables is more difficult to solve for the algorithm, can give slower results and even in some occasions may not converge [6].

Another approach is to divide the multi-class problem in several binary sub-problems. There are numerous methods that do the division. The most popular are *One against one* [7], *One against all* [8] , *Diagram Acyclic Graph (DAG)* [9] *and Binary decision Trees (BDT)* [10]. For sake of shortness, we will only focus on the BDT method, because it is the one we will use.

In [10], they proposed to build a binary tree in which at every node the remaining classes separated in two subgroups g_1 and g_2. A SVM classifier decides to which subgroup the new point belongs, so in which direction to move. In order to build the tree (the first step in the classification procedure) a clustering algorithm divides all the Q classes into g_1 and g_2. The algorithm calculates the gravity centers of all classes, the two classes with the biggest Euclidean distance from each other are assigned to each of the sub-groups. Then the algorithm checks the closest class to one of the sub-groups, this class is assigned to that sub-group, their gravity center is recalculated with the new points that have just been added. This is repeated until all classes have been assigned to one of the groups. In the example of Figure 2 we can see that $g_1 = \{2,3,4,7\}$ and $g_2 = \{1,5,6\}$. For each sub-group the clustering algorithm is repeated until there is no more than one class per sub-group. Those sub-groups will be called the leaves of the tree and a point that falls there will be assigned with the class of the leaf.

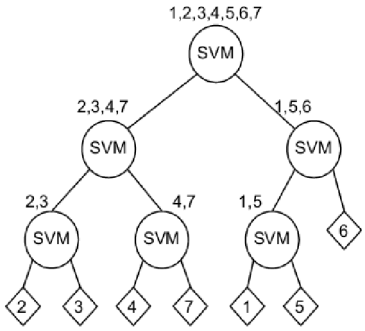

Figure 2 Illustration of SVM-BDT [10]

It is important to note that BDT testing time is smaller than other methods because the depth of the decision tree is of order $\log_2 Q$ since at every level the tree eliminates approximately half of the remaining classes. The testing time is an important characteristic that must be taken into account when choosing a classifier.

IV. Probabilistic Decision Trees (PDT)

1) Probability construction

In this paper, we propose to build a binary decision tree following the idea introduced by Madzarov [10], but instead of using a simple SVM classifier in each node, we use a SVM classifier associated with a sigmoid function (PSVM) to estimate the probability of membership to each sub-group in the node, as shown in Figure 3. The tree may be built using different criteria, for example the Euclidian distance between the gravity centers, the margin obtained by pairwise SVM [20] or some physical or functionalities criteria (mechanical, electrical, etc.). This is very interesting because in this way we introduce previous knowledge from an expert that might help the classification task. We can then build a probability function for each leaf, knowing the path that a point has to follow to reach it.

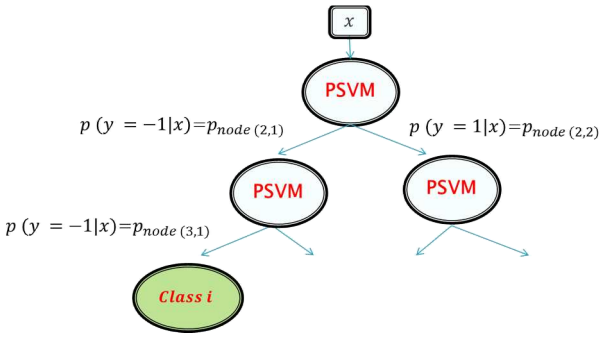

Figure 3 Example of a probability decision tree

Note that there is only one way to get to a leaf, so the probability functions are unique for a trained tree.

$$p(y = i|x) = \prod_{h=1}^{leaf} p_{node\,(h,l)} \tag{13}$$

h is the level of the tree and $h = 1$ is the root node. The previous expression states that the probability of membership of an element to the class i is calculated as the product of the probabilities of the decisions taken in all the nodes visited until arriving to the leaf. By $node(h, l)$ we mean the l node in the h level. Note that those functions are analytical, because in each node we will have an equation like 8, which only depends of $\boldsymbol{x}$ and the parameters (a, b) which are calculated before. Once the tree is built we will also have the Q probability functions, one for each class. When classifying future unknowing cases we will just have to evaluate the Q analytical functions and then chose the class with the highest score. However, we don't have to settle with just one predicted class. One of the most interesting things of the PDT is that we can have more than one prediction for one subject. We can have a list of all the possible classes, ordered after their plausibility, which is measured with the probabilities estimation. So, instead of having one predicted class like classic SVM multi-class methods, we will have several options. This will be useful for an example where the first prediction won't be correct. In classical multi-class no further information will be provided if the predicted class is not correct, because we will have a set of $Q - 1$ classes left, all of them equally plausible. Although with PDT, if the first choice wasn't the right one, we could try the second one and so on. We know that the next one will be the class with the highest probability among those that remain. Our experiments showed that in most of the datasets, the right answer is in the first choices.

2) Algorithm

To summarize the method's algorithm, the steps for building a PDT are:

Step 1 *Learning phase (construction of the classifier)*

Inputs: Training set s_1 *and* s_2

Outputs: Q probability functions (one for each class)

1. *Build a binary decision tree using* s_1.
2. *Train a SVM classifier for each node of the tree, model selection has to be done to guarantee good performance.*
3. *Fit a sigmoid to every SVM classifier trained in step 2 using* s_2*, according to Platt's method* [4] *and Lin algorithm* [19]. *Obtaining a probability function for the node.*
4. *For each leaf, build the probability function as the product of all the probability functions traversed (nodes)*

End step 1

Step 2 *Generalization phase (usage by the final user)*

Input: An unclassified example x

Output: A list of the proposed classes and their corresponding probability.

1. *Evaluate all the probability functions obtained is Step 1 for the new unclassified example.*
2. *Sort the classes according to the probabilities*

End step 2

3) Application to automotive diagnosis

PDT could be a very powerful decision helping tool in automotive diagnosis thanks to the scores associated to the predictions. As a matter of fact, when an assistant is facing a new unclassified case, the PDT will propose him a list of possible solutions, sorted by their likelihood. The assistant will try first the most likely solution. If it is not the right answer (we admit that sometimes the classifier might not sort the right answer in the first place, but the results of the applications in section V suggest that those are rare cases) the assistant will try next the second most likely solution and so on. This approach is a time saving tool to the workshop because only the more likely solutions have to be tried.

V. APPLICATIONS

The classification algorithm was coded in MATLAB, based on a SVM Toolbox [21]. Benchmark was conducted against the well-known SVM library LIBSVM [15] using *One-against-one (OvO), Diagram Acyclic Graph (DAG)* and LR-BDT. Classification tests were performed on datasets from the UCI Machine Learning Repository [22] and we have also built an artificial database that emulates the diagnosis car data. This database is constituted by 8 classes with 2 features. Each class is generated by a normal distribution in a 2D space. Some points of different classes actually are mixed. This allows us to test the sensibility of the classifiers as they have tendencies for over-fitting or not.

Datasets are described in Table 1, and one can appreciate how different they are. The number of attributes (dimensions for the SVM classifier) and the number of classes vary in a wide range, which makes the benchmark experiment very interesting and complete. Each feature was scaled between [-1, 1]. For each SVM method we conducted a model selection based on the algorithm proposed in [17]. Every score is the mean of the results obtained using a 5-fold cross validation. This eliminates the influence of randomly splitting a training and testing subset from the data

TABLE 1 DATASETS DESCRIPTION

Dataset	# attributes	# instances	# of classes
Iris[3]	4	150	3
Wine Quality[3]	11	1600	10
Ecopli[3]	7	336	8
Artificial base[4]	2	200	8

The results are show in Table 2. Only the test accuracy is shown because it is the variable that best represents the generalization capacity of classifiers. In bold the best results for each dataset are highlighted. Note that for PDT and LR-BDT the test accuracy has to be interpreted as the number of times the first prediction (the class with the highest probability) was the right class.

TABLE 2 ACCURACY COMPARISONS $(logC; log\sigma^2)$

Dataset	Test accuracy (%)			
	LIBSVM OvO	LIBSVM DAG	PDT	RL-BDT
Iris[3]	**98.67 (4;3)**	**98.67 (3.5;2)**	98 (2.5;1)	97.33
Wine Quality[3]	**63.97 (0;-1)**	59.94 (1.5;2)	61.74 (12.5;0)	59.38
Ecopli[3]	86.57 (5;5)	85.14 (2;0)	**88.57 (6.5;7)**	86.29
Artificial base[4]	**92.87 (-1;2)**	**92.87 (-7;1)**	**92.75 (2.5;2)**	91.75

Going further in PDT and LR-BDT predictions, we wanted to analyze the distribution of the right class among all the predictions made by both methods. The idea is to count how many times the right answer was the first predicted class or the second predicted class and so on. In Table 3 we show the results obtained for the five datasets. It is good to see that in all the cases we have the majority of classes well predicted on the first attempt. When the first answer isn't the right one, the second one is, in almost all the occasions. We only have a very small number of examples for which the correct class is one of the last predictions.

VI. CONCLUSION

In this paper, we have proposed a probabilistic binary decision tree as a response to features retrieving for car diagnosis.

[3] Datasets taken from the UCI *Machine Learning Repository* [22]

[4] Artificial base created by the authors

TABLE 3 ACCURACY OF PDT PREDICTIONS

Dataset	1st	2nd	3rd	4th	5th	6th	7th-11th
Iris[3] by SVM	98	2	0				
Iris[3] by RL	97.33	2	0.67				
Wine Quality[3] by SVM	61.74	15.96	8.57	7.20	2.98	3.54	0.01
Wine Quality[3] by RL	59.38	30	7.02	2.48	0.99	0.06	0.07
Ecopli[3] by SVM	88.57	6.57	2	1.14	0.29	1.14	0.29
Ecopli[3] by RL	86.29	9.43	1.14	0.29	0.29	2	0.56
Artificial base[4] by SVM	92.75	7.25	0	0	0	0	0
Artificial base[4] by RL	91.75	8.25	0	0	0	0	0

In this industrial context, the extraction of breakdown previous solution should be time saving and associated to a calibrated probability. The approach is based on the use of a binary decision tree structure with PSVM classifiers associated in the nodes to estimate the probability of membership to each sub-group. We can then build a probability function for each leaf, knowing the path that a point has to follow to reach it.

This original method has proved to be effective when tested in real word problems. It is very useful to have a score associated to a class prediction, because it allows further analysis of the results. The results showed that the PDT is an efficient and accurate method, comparable with the classic multi-class methods. In addition to that, the subsidiary predictions are a useful tool for the cases where the apparent first solution is not correct. This is an important advantage compared to the other approaches. PDT also showed to be more accurate than LR-BDT which is the other method that gives probability estimations. We may then consider the PDT as a complete method in terms of accuracy, multiple outputs and at the same time, giving a confidence index for each output. The application of PDT to an actual diagnosis database is in progress, where the performances of the PDT are tested in view of the size of the databases and the difficulties that it implies for processing them.

As perspectives we see many application fields for the proposed technique. Indeed, it can be used in any classification problem that may need more than one predicted class and scores associated. In particular, we are planning to implement for structural damage detection.

REFERENCES

[1] Y. Liu, H. H. Zhang and Y. Wu, *Hard or soft classification? large-margin unified machines,* vol. 106, Taylor & Francis, 2011, pp. 166-177.

[2] G. Wahba, "Soft and hard classification by reproducing kernel Hilbert space methods," *Proceedings of the National Academy of Sciences,* vol. 99, no. 26, pp. 16524-16530, 2002.

[3] J. David W. Hosmer and S. Lemeshow, Applied Logistic Regression, Wiley, 2004.

[4] J. Platt, "Probabilistic outputs for support vector machines and comparison to regularized likelihood methods.," in *Advances in Large Margin Classifiers*, Cambridge, MIT Press, 2000.

[5] J. Weston and C. Watkins, "Multi-class support vector machines Weston, J., & Watkins, C. (1998). Multi-class support vector machines. Technical Report CSD-TR-98-04," Department of Computer Science, Royal Holloway, University of London, London, 1998.

[6] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Neural Networks, IEEE Transactions on,* vol. 13, no. 2, pp. 415-425, 2002.

[7] J. Friedman, "Another approach to polychotomous classifcation," Stanford University, Department of Statistics, Stanford, 1996.

[8] V. Vapnik, *Statistical learning theory. 1998,* Wiley, New York, 1998.

[9] J. C. Platt, N. Cristianini and J. Shawe-Taylor, "Large margin DAGs for multiclass classification," *Advances in neural information processing systems,* vol. 12, no. 3, pp. 547-553, 2000.

[10] G. Madzarov, D. Gjorgjevikj, I. Chorbev and others, "A multi-class svm classifier utilizing binary decision tree," *Informatica,* vol. 33, no. 2, pp. 233-241, 2009.

[11] Z. Hu and C. Lo, "Modeling urban growth in Atlanta using logistic regression," *Computers, Environment and Urban Systems,* vol. 31, no. 6, pp. 667-688, 2007.

[12] P. Leung and L. T. Tran, "Predicting shrimp disease occurrence: artificial neural networks vs. logistic regression," *Aquaculture,* vol. 187, no. 1, pp. 35-49, 2000.

[13] Y. Lee, Y. Lin and G. Wahba, "Multicategory support vector machines (preliminary long abstract) Technical Report 1040," Department of Statistics, University of Wisconsin, Madison, WI, 2001.

[14] T. Fletcher, "Support vector machines explained," 2009. [Online]. Available: http://www.tristanfletcher.co.uk/. [Accessed January 2013].

[15] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology,* vol. 2, pp. 27:1--27:27, 2011.

[16] R. Fletcher, Practical methods of optimization, John Wiley & Sons, Inc., 1987.

[17] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with Gaussian kernel," *Neural computation,* vol. 15, no. 7, pp. 1667-1689, 2003.

[18] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," *The annals of statistics,* vol. 26, no. 2, pp. 451-471, 1998.

[19] H.-T. Lin, C.-J. Lin and R. C. Weng, "A note on Platt's probabilistic outputs for support vector machines," *Mach. Learn.,* vol. 68, no. 3, pp. 267-276, oct 2007.

[20] T. K. Chalasani, A. M. Namboodiri and C. Jawahar, "Support vector machine based hierachical classifieds for large class problems," in *Proceedings of the sixth International Conference on Advances in Pattern Recognition, Kolkata, India*, 2007.

[21] S. Canu, A. Rakotomamonjy, Y. Grandvalet and V. Guigue, "SVM and Kernel Methods Toolbox," 2007. [Online]. Available: http://www.mloss.org/software/view/33/. [Accessed January 2013].

[22] K. Bache and M. Lichman, "UCI Machine Learning Repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml/index.html. [Accessed March 2013].