Filtering noisy continuous labeled examples[†]

José Ramón Quevedo, María Dolores García, Elena Montañés

Centro de Inteligencia Artificial. Oviedo University. Viesques, E-33271 Gijón, Spain
{quevedo, marilo, elena}@aic.uniovi.es

Abstract. It is common in Machine Learning where rules are learned from examples that some of them could not be informative, otherwise they could be irrelevant or noisy. This type of examples makes the Machine Learning Systems produce not adequate rules. In this paper we present an algorithm that filters noisy continuous labeled examples, whose computational cost is $O(N \cdot logN+NA^2)$ for N examples and A attributes. Besides, it is shown experimentally to be better than the embedded algorithms of the state-of-the art of the Machine Learning Systems.

1 Introduction

In Machine Learning environment the process of learning rules from available labeled examples (training data) is called *training* and the process of applying these learned rules to unlabeled examples (test data) is called *testing*.

An example is represented by a sequence of pairs attribute-value and a label that represents its category. The category can be symbolic or continuous. The examples have the same attributes although some values could be *missing*.

A good performance could be reached supposing that the sets of training and test data have the same distribution of the category over theirs attributes [3].

One of the most difficult tasks when dealing with real problems is to find the attributes more related to the category in the way to define a fixed distribution that a Machine Learning System (*MLS*) could learn. An additional difficulty is the possible presence of noisy examples mainly caused by the collection of them.

In this paper an algorithm that filters noisy continuous labeled examples is presented. It is shown that some *MLS* perform better using the filtered data set than using the original one.

2 Task Definition

This paper describes an algorithm that removes noisy examples from a set of continuous labeled examples producing a subset containing informative ones.

[†] The research reported in this paper has been supported in part under MCyT and Feder grant TIC2001-3579

This Noisy Continuos Labeled Examples Filter (*NCLEF*) takes an example *e* and classifies it as noisy or as informative. This classification is made according to two errors: the error committed when the current example is taking into account on the data set and the error committed when the current example is removed from the data set. The method employed to evaluate these errors is the continuous version of *knn* [1], it is used with Leaving-One-Out (*LOO*)[12] (See Fig. 1).

It is well known that *knn* is noise sensitive [1], that is, adding a noisy example to the data set the performance of *knn* would be worse. The algorithm described in this paper is based on this idea on a reverse way: "if the removal of an example produces lower error then this example is supposed to be noisy".

The algorithm has two main disadvantages. The first one is that its computacional cost is $O(N \cdot O(knn)) = O(N \cdot kAN^2) = O(kAN^3)$ for N examples and A attributes. The second one is the insignificant influence of an example over the *knn*'s error for large data sets. A Divide and Conquer method (*D&C*) is incorporated to overcome these difficulties. The resulting filter adding *D&C*, called *NCLEFDC*, makes its cost be $O(N \cdot logN + NA^2)$.

3 Related Work

This work is related with Example Selection. There are several techniques about Example Selection proposed by Wilson & Martinez [14]; Aha [2]; Aha, Kibler & Albert [1] and Cameron-Jones [5].

Blum and Langley [4] propose at least three reasons for selecting examples: purposes of computational efficiency, high cost of labeling and focusing attention on informative examples. Our algorithm pays attention to the third one in the way that it trends to remove noisy examples and keep informative ones.

Most of the algorithms for Example Selection work only on symbolic labeled examples. There are algorithms to deal with data set containing noisy continuous labeled examples which are embedded in the *MLS* (*M5'* [11], *RT* [9], *Cubist* [8]), but there is no documentation for commercial systems like *Cubist*.

In this paper is compared the performance of using *NCLEFDC* before the *MLS* with the performance of using only those *MLS*.

4 The NCLEFDC algorithm

The algorithm involves three steps which are detailed in the next subsections: the principle, the iterative algorithm and the incorporation of a *D&C*.

4.1 The Principle

The principle involves the election of the measures employed to decide if an example is noisy or not. A trivial measure could be the *knn*'s error for this example, being the noisiest example that with the highest error. That is true in most cases, but it is not

useful since there could be no noisy examples with high error and since it is difficult to find out the meaning of "high error". Fortunately, noisy examples in *knn* entails another very useful feature, namely, adding a noisy example to the data set causes an increasing of the errors of its neighbors (See Figure 1).

Taking into account this last feature, the algorithm sees the effect that an example causes to the error of its neighbors in order to decide if it is noisy or not. The error is approached by means of *LOO* with *knn* over the data set. The error for N examples is denoted by E_N, the error when removing example e from the data set is denoted by $E_{N-1}(e)$ and the error E_N but without considering the error of the example e is denoted by $E'_N(e)$. This latter error is given by equation (1).

$$E'_N(e) = \frac{E_N \cdot N - ErrorKnn(data = DataSet - \{e\}, test = \{e\})}{N-1} \quad (1)$$

$$SupposedNoisy(e) \Leftarrow E'_N(e) > E_{N-1}(e) \quad (2)$$

It could be supposed that an example is noisy in the way of equation (2). This means that the presence of this example makes that the *knn*'s error of the examples that take it as its neighbor be bigger than if the example is removed from the data set.

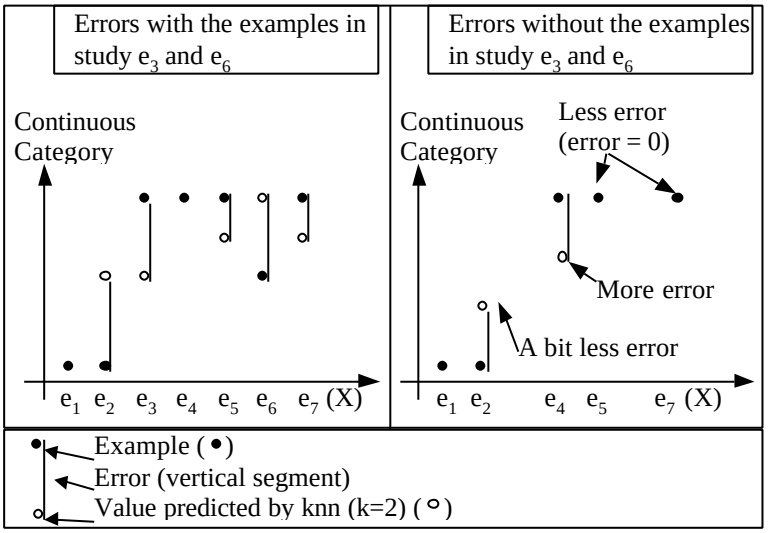

Fig. 1. Schema of a discrete step function and of how the errors vary when examples are removed. Examples e_3 and e_6 have the same error, but e_3 is informative (it is the first example of the next step) and e_6 is noisy. If e_6 is removed the errors of its neighbors (e_5 and e_7) becomes 0, but if e_3 is removed the sum of the error of its neighbors (e_2 and e_4) is higher.

The algorithm requires choosing the value of k for the *knn*. It should not be so small because it is necessary that an example has enough neighbors in order to measure its influence. It should also not be so big because the predictions of *knn* should significantly vary if an example is removed. A good value for k is generally determined via cross-validation [13] but a *bad* value is preferred in order to make *knn*

be very noise sensitive. Although the experiments show that the influence of k over *NCLEF* is not so much significant, it is chosen the best one obtained in our experiments, that is: $k=A/2+1$, where A is the number of attributes of the problem.

4.2 The NCLEF algorithm

The *NCLEF* algorithm based on the principle previously shown tries to remove the example with more error in each iteration. Over this structure it is possible to develop several versions. We prefer to make a prudent version, one that the main objective is to keep informative examples. In this way three aspects of the algorithm are changed. Firstly a new test for noisy examples is proposed. This test, described in equation (3), takes into account the number of examples previously removed in order to avoid removing informative ones. Secondly, the application of the test is limited to examples whose error is above a fixed threshold (*MinError* in equation (4)). Finally, the algorithm ends when it considers the example as not noisy.

$$prudentNoisy(e) \Leftarrow E'_N(e) \frac{N - Examplesremoved}{N} > E_{N-1}(e) \tag{3}$$

$$MinError = \overline{LOO(knn(DataSet))} + \sigma\left(LOO(knn(DataSet))\right) \tag{4}$$

In equation (4) *LOO(knn(DataSet))* is the set of errors of a *LOO* execution on the data set using *knn*. *MinError* is chosen to be the sum of the average and the typical deviation of all LOO executions. The addition of the typical deviation to the average assures that the algorithm only tries to remove examples with high *knn*'s error.

The *NCLEF* algorithm is described as follows.

```
DataSet NCLEF(DataSet DS){
  // Obtain the initial Average Error using knn
  {ExampleMaxErr,AverageErr,DeviationErr}=LOOKNN(DS);
  MinError=AverageErr+DeviationErr;

  for(ite=1;ExampleMaxErr.Error>MinError;ite++){
    // Obtain the Average Error and Example with
    // more error using knn
    {ExampleMaxErrN1,AverageErrN1}=
          LOOKNN(DS-{ExampleMaxErr});
    //If the example is noisy, it is eliminated
    if(prudentNoisy(ExampleMaxErr)){
      DS=DS-{ExampleMaxErr};
      ExampleMaxErr=ExampleMaxErrN1;
      AverageErr=AverageErrN1;
    }
    else break; // the example is not noisy
  } // end of for
 return DS;
} // end of NCLEF
```

Function *LOOKNN* applies *LOO* with *knn* to the data set given as a parameter. It returns the average error, the deviation error and the example with highest error. This information is necessary in function *prudentNoisy* to test if an example is noisy or not.

4.3 Using Divide and Conquer on NCLEF

As *NCLEF* iterates for each example and uses *knn* its order is $O(NCLEF)=O(N\cdot O(knn))=O(N\cdot kAN^2)=O(kAN^3)$. Besides, given that we choose k to be $A/2-1$, then $O(NCLEF)=O(kAN^3)=O(A^2N^3)$ for N examples and A attributes. This order makes *NCLEF* computationally unacceptable. That is the reason why *D&C* is applied.

The new algorithm, called *NCLEFDC*, divides recursively the data set in subsets, then applies *NCLEF* to each subset and finally joins all partial filtered subsets.

The goal is to divide the original data set into subsets where all the neighbors of an example in the original set were in the same subset. As this could be impossible or, at least, very computationally expensive, the *Divide* method based on the following heuristic is used: (1) To take an example e and to calculate its $\|\ \|_1$, (2) to obtain two subsets, one with the examples with more $\|\ \|_1$ than e and the other one with the examples with less $\|\ \|_1$ than e. The algorithm looks for an example that produces two subsets with similar number of elements. The attributes values are normalized between 0 and 1 to avoid the generation of concentric subsets obtained by the application of a norm. Given that all norms are equivalent in finite dimension spaces, $\|\ \|_1$ is chosen due to its faster calculus than euclidean one employed by *knn*.

The order of *NCLEFDC* is $O(NCLEFDC)=O(N/M(O(Divide)+O(knn)))$, where M is the maximum number of the size of the subsets and N/M is the number of subsets.

The order of *Divide* is $O(Divide)=O(N_{DIV}M)$, where N_{DIV} is the number of examples of the data subset. In each execution N_{DIV} could be different, so the average is estimated in the following way: Supposing that *Divide* splits the data set into two subsets with equal number of examples, the algorithm is executed 2^i times, each one with a data set of $N/2^i$ examples in depth i of the recursive algorithm. This is made until $M/2<N/2^L<M$, been L the maximum depth of the recursive algorithm. If M and N are integers such that $0<M<N$ then equation (8) represents an estimation of N_{DIV}. Then $O(Divide)$ and $O(NCLEFDC)$ are given by the equations (9) and (10) respectively.

$$2^L > \frac{N}{M} \Rightarrow 2^{L+1}-1 > \frac{2N}{M}-1 \Rightarrow \frac{N(L+1)}{2^{L+1}-1} < \frac{N(L+1)}{2N/M-1} \tag{5}$$

$$N_{DIV} = \frac{\sum_{i=0}^{L} 2^i \frac{N}{2^i}}{\sum_{i=0}^{L} 2^i} = \frac{N(L+1)}{2^{L+1}-1} \; by(5) < \frac{N(L+1)}{2N/M-1} < \frac{N(L+1)}{N/M} = M(L+1) \tag{6}$$

$$\frac{N}{2^L} > \frac{M}{2} \Rightarrow \frac{2N}{M} > 2^L \Rightarrow \log_2\left(2\frac{N}{M}\right) > L \Rightarrow 1+\log_2\left(\frac{N}{M}\right) > L \tag{7}$$

$$N_{DIV} by(6) < M(L+1) by(7) < M\left(2+\log_2\frac{N}{M}\right) \tag{8}$$

$$O(Divide) = O(N_{DIV}\,M) = O\left(M\left(2+\log_2\left(\frac{N}{M}\right)\right)M\right) = O\left(M^2\left(\log_2\left(\frac{N}{M}\right)\right)\right) \tag{9}$$

$$O(NCLEFDC) = O\left(\frac{N}{M}\left(O(Divide)+O(Knn)\right)\right) = O\left(MN\left(\log_2\left(\frac{N}{M}\right)\right)+NA^2M\right) \tag{10}$$

Fixing M to be constant in all experimentation, then $O(NCLEFDC)$ is:

$$O(NCLEFDC) = O\left(N\cdot\log_2(N)+NA^2\right) \tag{11}$$

The algorithm does not always split the data set into two subsets with exactly the same number of examples, otherwise it could split into subsets with a proportion between 40%-60%. Then, the base of the logarithm in equation (11) could be lower than 2, but even though the first addend would be always lower that N^2.

The algorithm *NCLEFDC* is described below:

```
DataSet NCLEFDC(DataSet DS,int M){
 // If there are more examples in the data set than M
 // we divide the data set into two subsets
 if(#DS>M) {
  {DS1,DS2}=Divide(DS,M);
  //  The global result is the Union of the partial
  // result of the two recursive calls to NCLEFDC
  return Union(NCLEFDC(DS1),NCLEFDC(DS2));
 } else return NCLEF(DS); // base case
} // End of program

{DataSet DS1,DataSet DS2} Divide(DataSet DS,int M){
 Min=0;
 Max=MaxNormalizedNorm1;
 Example ERand;
 for(iterations=1;iterations<M;iterations++) {
  ERand=RandomExampleBetween(DS,Min,Max);
  above=PercentExamplesWithMoreNorm1(DS,ERand);
  if(above>=40 and above=<60) break; // good solution
  // Redefine search interval
  if(above<40) Max=Norm1(ERand);
  if(above>60) Min=Norm1(ERand);
 } // End of for
 DS1=ExamplesWithLessNorm1(DS,ERand);
 DS2=ExamplesWithMoreNorm1(DS,ERand);
}// End of Divide
```

The function *Divide* searches for an example *e* whose $\| \|_1$ is a percentile between 40% and 60% in the distribution of all $\| \|_1$. This interval is fixed as an approximation of 'equal number of examples'.

5 Experimental Evaluation

A set of experiments were conducted to compare the performance of *M5'*, *Cubist* and *RT* with and without *NCLEFDC*.

The well known heterogeneous data sets of the Torgo's repository at *LIACC* [10] are used. Each experiment consists of a *Cross Validation (CV)* with 10 folds. Besides, it is employed MLC++[6] with 2032 seed to make the experiments to be repeatable.

The result of a *CV* experiment is the Medium Average Deviation (*MAD*), but in the forward tables it is shown the Relative Medium Average Deviation (*RMAD*) which is the *MAD* divided by the *MAD* of the system that always predicts the average function.

Table 1. List of the data sets of the Torgo's repository. The name, the number of examples (#Ex), the number of attributes (#Att) and the *MAD* of the system that always predict the average function (Av. MAD) are shown for each data set. Each data set is also numbered (Nº) to be referred forward using this number.

Nº	Name	#Ex	#Att	Av.MAD	Nº	Name	#Ex	#Att	Av.MAD
1	Abalone	4177	8	2,363	16	Diabetes	43	2	2,363
2	Ailerons	13750	40	0,0003	17	Elevators	16599	18	0,0046
3	Airpla.Com.	950	9	5,4852	18	Friedman Ex.	40768	10	4,0648
4	Auto-Mpg	398	4	6,5459	19	Housing	506	13	6,6621
5	Auto-Price	159	14	4600,65	20	Kinematics	8192	8	0,2156
6	Bank 32NH	8192	32	0,0903	21	Machine-Cpu	209	6	96,9004
7	Bank 8FM	8192	8	0,1236	22	MvExample	40768	10	8,8932
8	Cal. Hou.	20640	9	91174,5	23	PoleTele.	15000	48	37,2124
9	Cart Delve	40768	10	3,6069	24	Pumadyn(32)	8192	32	0,0235
10	Census(16)	22784	16	32428,2	25	Pumadyn(8)	8192	8	4,8659
11	Census(8)	22784	8	32428,2	26	Pyrimidines	74	27	0,0957
12	Com.Act	8192	21	10,6326	27	Servo	167	2	1,1662
13	Com.Act(s)	8192	12	10,6326	28	Triazines	186	60	0,1187
14	Delta Ailer.	7129	5	0,0003	29	Wisconsin	198	32	29,6833
15	Delta Eleva	9517	6	0,002					

Table 2 shows that the use of *NCLEFDC* does not improve the performance significantly because the data sets do not have enough noisy examples. Table 3 shows the results when the 10% of training data are changed by noisy examples in each execution of the *CV* (the test data are not modified). Under these circumstances the performance of *Cubist*, *M5'* and *RT* gets better. So *NCLEFDC* removes examples better than the embedded filters that use these systems.

Table 2. *RMAD* of the *MLS* with and without the *NCLEFDC* filter. It is shown the *RMAD* for each data set of Torgo's repository and the average of all *RMADs* (Av.) of a *MLS*.

	Only the systems			NCLEFDC before the systems		
	Cubist 1.10	M5'	RT 4.1	Cubist 1.10	M5'	RT 4.1
1	105,16%	101,12%	100,46%	104,63%	99,89%	100,46%
2	73,88%	66,67%	84,74%	73,88%	66,67%	84,74%
3	39,54%	33,82%	42,62%	39,54%	33,82%	39,14%
4	27,83%	28,01%	48,81%	27,83%	28,01%	48,76%
5	33,07%	31,75%	36,73%	33,65%	30,54%	36,70%
6	34,04%	34,41%	43,17%	34,79%	34,83%	34,45%
7	12,67%	11,69%	16,94%	12,60%	11,76%	17,20%
8	63,48%	64,24%	67,85%	63,06%	64,43%	71,23%
9	100,00%	50,00%	50,00%	100,00%	50,00%	50,00%
10	19,37%	20,64%	24,33%	19,37%	20,43%	24,41%
11	17,88%	18,53%	22,90%	17,88%	18,45%	22,90%
12	50,67%	51,42%	52,40%	50,65%	51,57%	52,39%
13	58,47%	64,45%	70,21%	58,14%	63,23%	70,32%
14	26,38%	28,51%	30,64%	26,81%	28,51%	30,64%
15	15,89%	17,80%	22,65%	15,80%	17,89%	22,65%
16	55,61%	56,17%	66,70%	55,52%	56,17%	66,88%
17	22,07%	22,08%	22,43%	22,07%	22,08%	22,44%
18	6,37%	8,28%	8,78%	6,56%	8,25%	8,76%
19	52,41%	58,01%	57,93%	51,60%	57,47%	57,49%
20	49,67%	54,57%	54,56%	49,28%	54,28%	54,02%
21	38,89%	35,98%	41,68%	35,45%	35,94%	42,08%
22	50,00%	36,96%	52,17%	50,00%	36,96%	52,17%
23	23,83%	26,63%	33,88%	23,83%	26,70%	33,79%
24	100,00%	33,33%	33,33%	100,00%	33,33%	33,33%
25	0,22%	0,97%	12,42%	0,22%	0,97%	12,36%
26	85,17%	81,80%	88,96%	89,89%	81,55%	89,72%
27	101,27%	97,49%	100,56%	101,07%	97,16%	96,76%
28	30,88%	28,16%	40,53%	32,18%	27,57%	42,62%
29	55,00%	55,00%	55,00%	55,00%	55,00%	55,00%
Av.	46,54%	42,02%	47,70%	46,60%	41,84%	47,36%

Table 3. *RMAD* of the *MLS* with and without the *NCLEFDC*. It is shown the *RMAD* for each data set of Torgo's repository and the average of all *RMADs* (Av) of a *MLS*. The data in each execution of a *CV* are modified with a 10% of noisy examples.

	Only the systems			NCLEFDC before the systems		
	Cubist 1.10	M5'	RT 4.1	Cubist 1.10	M5'	RT 4.1
1	98,19%	101,52%	96,18%	98,88%	98,50%	95,58%
2	76,05%	71,01%	90,44%	75,84%	69,85%	90,86%
3	41,22%	50,84%	53,20%	38,84%	40,98%	43,10%
4	54,59%	60,01%	79,86%	39,61%	43,19%	54,45%
5	38,85%	37,01%	43,45%	35,57%	34,32%	40,66%
6	46,13%	42,09%	53,64%	41,18%	40,42%	52,08%
7	19,66%	20,60%	25,72%	15,28%	14,98%	28,02%
8	64,60%	68,83%	73,61%	62,53%	64,25%	75,30%
9	100,00%	50,00%	50,00%	100,00%	50,00%	50,00%
10	35,88%	41,70%	41,97%	27,85%	33,15%	34,19%
11	26,26%	30,44%	34,39%	20,53%	23,55%	28,20%
12	53,01%	54,31%	55,26%	53,04%	53,32%	55,07%
13	64,75%	76,53%	82,57%	57,28%	69,35%	76,34%
14	31,20%	36,32%	37,61%	30,77%	36,32%	37,18%
15	33,94%	40,85%	41,38%	24,13%	32,91%	33,81%
16	58,02%	60,58%	72,82%	57,79%	59,65%	72,77%
17	23,02%	24,10%	26,27%	22,25%	22,50%	24,03%
18	14,27%	18,35%	18,98%	10,52%	12,23%	12,35%
19	57,33%	79,30%	76,55%	48,19%	62,31%	60,37%
20	50,08%	74,08%	73,12%	43,14%	56,42%	56,37%
21	42,11%	41,53%	49,54%	42,59%	39,72%	47,09%
22	52,63%	56,14%	68,42%	45,61%	42,11%	57,89%
23	25,43%	34,82%	44,17%	24,92%	32,48%	41,29%
24	100,00%	33,33%	66,67%	100,00%	33,33%	66,67%
25	7,38%	15,24%	17,32%	3,65%	7,19%	9,65%
26	97,13%	93,53%	95,25%	93,28%	95,74%	98,20%
27	157,47%	156,00%	152,13%	156,73%	152,29%	151,54%
28	45,21%	44,78%	51,63%	42,80%	40,01%	46,15%
29	55,00%	55,00%	60,00%	55,00%	55,00%	60,00%
Av.	54,12%	54,10%	59,73%	50,61%	48,83%	55,15%

6 Conclusions

This paper describes an algorithm that filters noisy continuous labeled examples from a data set. This algorithm uses *knn* to determine if an example is noisy or not. *Knn* is helped by *D&C* in order to reduce its computational cost.

The quality of this algorithm has been evaluated by two criteria: the cost associated to the filtering and the accuracy of *M5'*, *Cubist* and *RT* when they use the filtered data set instead of the original one. The cost of the algorithm is $O(Nlog_2N+A^2N)$ where N is the number of examples and A is the number of attributes.

It is shown experimentally that the accuracy of the latter systems is better when they use this filter under the presence of noisy examples. However, the accuracy is the same when there are no noisy examples.

A conclusion is that the performance of *M5'*, *Cubis*t and *RT* is worse under the presence of noisy examples. Another conclusion is that in our experiments *NCLEFDC* deals with noisy examples better than the embedded algorithms of the latter systems.

In this paper only basic principles are presented, but a lot remains could be done in this area. We are interested in the following issues: (1) to calculate automatically the stop condition of the *D&C* phase; (2) to extend this idea to a discrete labeled examples; (3) to transfer the use of *knn* as noise detector to the area of feature selection.

7 References

1. Aha, D.W., Kibler, D., Albert, M.K.: Instance based learning algorithms. Machine Learning, Vol. 6. (1991) 37-66
2. Aha,D.W.: Lazy learning. Kluwer Academic Publishers, Dordrecht. (1997)
3. Blum A.L.: Relevant examples and relevant features: Thoughts from computational learning theory. In AAAI Fall Symposium on `Relevance'. (1994) 31
4. Blum A.L., Langley. P.:Selection of relevant features and examples in machine learning. Artificial Intelligence. (1997) 245-271
5. Cameron-Jones, R.M.: Instance Selection by encoding length heuristic with random mutation hill climbing. IEEE Proc. of the 8th Australian Joint Conference on AI. World Scientific. (1995) 99-106.
6. Kohavi, R., John, G., Long, R., Manley, D., & Pfleger, K. (1994). MLC++: A machine learning library in C++. In Proc. of the 6th International Conference on Tools with Artificial Intelligence, 740-743. IEEE Computer Society Press.
7. Quinlan, J.R.: Learning with continuous classes. In Proc. 5th Australian Joint Conference on Artificial Intelligence. World Scientific, Singapore, (1992) 343-348.
8. Quinlan, J.R.: Cubist. http://www.rulequest.com/cubist-info.html
9. Torgo. L.: Functional models for regression tree leaves. In Proc. of the 14th International Conference on Machine Learning, Nashville, TN. Morgan Kaufmann. (1997) 385-393
10. Torgo. L: Regression Data Sets Repository at LIACC (University of Porto). http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html
11. Wang, Y., and Witten, I.H.. Inducing model trees for continuous classes. In Poster Papers 9th European Conf. on Machine Learning. Prague, Czech Republic. (1997) 128-137.
12. Weiss, S. M., Kulikowski, C. A.: Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning, and expert systems. MorganKaufmann, San Mateo, CA, (1991.)
13. Wettschereck, D., Dietterich, T. G.: Locally adaptive nearest neighbor algorithms in Advances of Neural Information Processing Systems 6. Morgan Kaufmann Publishers. (1994) 184-191
14. Wilson, D.R., Martinez, T.R.: Instance pruning techniques. Proc. of the 14th International Conference on Machine Learning. Morgan Kaufmann, Nashville, TN., (1997) 403-411