Iterative Bayes

João Gama

LIACC, FEP-University of Porto, Rua Campo Alegre 823, 4150 Porto, Portugal

Abstract

Naive Bayes is a well-known and studied algorithm both in statistics and machine learning. Bayesian learning algorithms represent each concept with a single probabilistic summary. In this paper we present an iterative approach to naive Bayes. The Iterative Bayes begins with the distribution tables built by the naive Bayes. Those tables are iteratively updated in order to improve the probability class distribution associated with each training example. In this paper we argue that Iterative Bayes minimizes a quadratic loss function instead of the 0–1 loss function that usually applies to classification problems. Experimental evaluation of Iterative Bayes on 27 benchmark data sets shows consistent gains in accuracy. An interesting side effect of our algorithm is that it shows to be robust to attribute dependencies.

Keywords: Naive Bayes; Iterative optimization; Supervised machine learning

1. Introduction

Pattern recognition literature [5] and machine learning [17] present several approaches to the learning problem. Most of them are in a probabilistic setting. Suppose that $P(C_i | \tilde{x})$ denotes the probability that example $\tilde{x}$ belongs to class $i$. The zero-one loss is minimized if, and only if, $\tilde{x}$ is assigned to the class $C_k$ for which $P(C_k | \tilde{x})$ is maximum [5]. Formally, the class attached to example $\tilde{x}$ is given by the expression

$$\argmax_i P(C_i | \tilde{x}).$$

Any function that computes the conditional probabilities $P(C_i | \tilde{x})$ is referred to as discriminant function. Given an example $\tilde{x}$, the Bayes theorem provides a method...
to compute $P(C_i|\tilde{x})$:

$$P(C_i|\tilde{x}) = \frac{P(C_i)P(\tilde{x}|C_i)}{P(\tilde{x})}. \quad (2)$$

$P(\tilde{x})$ can be ignored, since it is the same for all the classes, and does not affect the relative values of their probabilities. Although this rule is optimal, its applicability is reduced due to the large number of examples required to compute $P(\tilde{x}|C_i)$. To overcome this problem several assumptions are usually made. Depending on the assumptions made we get different discriminant functions leading to different classifiers. In this work we study one type of discriminant function, that leads to the naive Bayes classifier.

1.1. Naive Bayes classifier

Assuming that the attributes are independent given the class, $P(\tilde{x}|C_i)$ can be decomposed into the product $P(x_1|C_i) \cdots P(x_a|C_i)$. Then, the probability that an example belongs to class $i$ is given by

$$P(C_i|\tilde{x}) \propto P(C_i) \prod_j P(x_j|C_i). \quad (3)$$

The classifier obtained by using the discriminant function (3) and the decision rule 1 is known as the naive Bayes classifier. The term naive comes from the assumption that the attributes are independent given the class.

1.1.1. Implementation details

All the required probabilities are computed from the training data. To compute the prior probability of observing class $i$, $P(C_i)$, a counter, for each class is required. To compute the conditional probability of observing a particular attribute-value given that the example belongs to class $i$, $P(x_j|C_i)$, we need to distinguish between nominal attributes, and continuous ones. In the case of nominal attributes, the set of possible values is a numerable set. To compute the conditional probability we only need to maintain a counter for each attribute-value and for each class. In the case of continuous attributes, the number of possible values is infinite. There are two possibilities. We can assume a particular distribution for the values of the attribute and usually the normal distribution is assumed. As alternative we can discretize the attribute in a preprocessing phase. The former has been proved to yield worse results than the latter [3,4]. Several methods for discretization appear in the literature. A good discussion about discretization is presented in [4]. In [3] the number of intervals is fixed to $k = \min(10; \text{nr. of different values})$ equal width intervals. Once the attribute has been discretized, a counter for each class and for each interval is used to compute the conditional probability.

All the probabilities required by Eq. (3) can be computed from the training set in one step. The process of building the probabilistic description of the data set is very
fast. Another interesting aspect of the algorithm is that it is easy to implement in an incremental fashion because only counters are used.

1.1.2. Analysis of the algorithm

Domingos and Pazzani [3] show that this procedure has a surprisingly good performance in a wide variety of domains, including many where there are clear dependencies between attributes. They argue that the naive Bayes classifier can approximate optimality when the independence assumption is violated as long as the ranks of the conditional probabilities of classes given an example are correct.

Some authors [12,13] suggest that this classifier is robust to noise and irrelevant attributes. They also note that the learned theories are easy to understand by domain experts, most due to the fact that the naive Bayes summarizes the variability of the data set in a single probabilistic description, and assumes that these are sufficient to distinguish between classes.

Some average-case analysis of the performance of naive Bayes on domains with known characteristics has been presented, for example [14]. Although the analysis is restricted to boolean variables and M-N concepts, it looks like a promising research direction.

1.1.3. Improvements

A few techniques have been developed to improve the performance of the naive Bayes classifier. Some techniques apply different naive Bayes classifiers to different regions of the input space. For example:

- Langley has presented the recursive naive Bayes [13]. An algorithm that recursively constructs a hierarchy of probabilistic concept descriptions. The author concludes that “the results are mixed and not conclusive, but they are encouraging enough to recommend closer examination”.
- Kohavi has presented the naive Bayes tree [10]. It is a hybrid algorithm. It generates a regular univariate decision tree, but the leaves contain a naive Bayes classifier built from the examples that fall at this node. The approach retains the interpretability of naive Bayes and decision trees, while resulting in classifiers that frequently outperform both constituents, especially in large data sets.

Other techniques have built new attributes that reflect interdependencies between original attributes. For example:

- Kononenko has presented the semi-naive Bayes classifier [12]. He attempted to join pairs of attributes, making a cross-product attribute, based on statistical tests for independence. The experimental evaluation was inconclusive.
- Pazzani has presented the constructive Bayesian classifier [19]. It employs a wrapper model [9] to find the best Cartesian product attributes from existing nominal attributes. It also considers deleting existing attributes. It has been shown to improve the naive Bayes classifier.

Techniques that address the problem of the presence of continuous attributes are also present in the literature:
• George John [8] has presented the flexible Bayes that uses, for continuous attributes, a kernel density estimation (instead of the single Gaussian assumption) but retains the independence assumption. The estimated density is averaged over a large set of kernels:

$$P(\tilde{x}|C_i) = \frac{1}{nh} \sum_j K \left( \frac{x - \mu_j}{h} \right),$$

where $h$ is the bandwidth parameter and $K$ the kernel shape $K = g(x, 0, 1)$. Experimental evaluation on UCI data sets shows that flexible Bayes achieves significantly higher accuracy than naive Bayes on many domains.

• Gama [7] has presented a linear Bayes algorithm that uses a multivariate normal distribution to compute the class membership probability for the set of continuous attributes. It has been shown to improve naive Bayes classifiers using either discretization or a single Gaussian distribution for each attribute. Some techniques introduce an explicit search for a better hypothesis. For example:

• Webb and Pazzani [23] have presented an extension to the naive Bayes classifier, where a numeric weight is inferred for each class using a hill-climbing search. During classification, the naive Bayesian probability of a class is multiplied by its weight to obtain an adjusted value. The use of this adjusted value in place of the naive Bayesian probability is shown to improve significantly predictive accuracy.

2. Related work

The work of Webb and Pazzani [23] clearly illustrates the benefits of adjusting the priors of a class. In their work a numeric weight is inferred for each class. During classification, the naive Bayes probability of a class is multiplied by its weight to obtain an adjusted value. This process has been shown to improve significantly the predictive accuracy.

Our method, instead of adjusting the priors of a class, adjusts the conditional probabilities $P(x_j|C_i)$. In our perspective this has similarities with the process of computing the weights in a linear machine [5]. Defining one Boolean attribute for each value of an attribute and applying logarithms to Eq. (3) we obtain

$$\log(P(C_i|\tilde{x})) \propto \log(P(C_i)) + \sum_j \log(P(x_j|C_i)).$$ (4)

This equation shows that naive Bayes is formally equivalent to a linear machine. Training a linear machine is an optimization problem that has been strongly studied for
example in the neural network community [21] and machine learning [2,17]. In this section we review the work done in the machine learning community.

The absolute error correction rule [5] is the most common method used to determine the coefficients for a linear machine. This is an incremental algorithm that iteratively cycles through all the instances. At each iteration each example is classified using the actual set of weights. If the example is misclassified then the weights are updated. Supposing that the example belongs to class \(i\) and is classified as \(j\) with \(i \neq j\), then the weights \(W_i\) and \(W_j\) must be corrected. The correction is accomplished by \(W_i \leftarrow W_i + kY\) and \(W_j \leftarrow W_j - kY\), where the correction

\[
k = \frac{(W_j - W_i)^T Y}{2Y^T Y} + \varepsilon
\]

causes the updated linear machine to classify the instance correctly (\(\varepsilon\) is a small constant). If the instances are linearly separable, then cycling through the instances allows the linear machine to partition the instances into separate convex regions. If the instances are not linearly separable, then the error corrections will not cease, and the classification accuracy of the linear machine will be unpredictable. To deal with this situation two variants often referred in the literature are:

1. **Pocket algorithm**, that maximizes the number of correct classifications on the training data. It stores in \(P\) the best weight vector \(W\), as measured by the longest run of consecutive correct classifications, called the *pocket count*. A LM based on the pocket vector \(P\) is optimal in the sense that no other weight vector visited so far is likely to be more accurate classifier.

2. **Thermal training**, that converges to a set of coefficients by paying decreasing attention to large errors. This is done by using the correction factor: \(c = \beta^2/(\beta + k)\) where \(\beta\) is annealed during training and \(k\) is given by Eq. (5). The training algorithm repeatedly presents examples until the linear machine converges.

Also the *logistic discriminant* [16] optimizes the coefficients of the linear machine using gradient descent. Given a set of vectors of coefficients \(W\), the algorithm maximizes a conditional likelihood that is given by

\[
L(W_1, \ldots, W_{q-1}) = \prod_{\tilde{x} \in C_1} P(C_1|\tilde{x}) \prod_{\tilde{x} \in C_2} P(C_2|\tilde{x}) \cdots \prod_{\tilde{x} \in C_q} P(C_q|\tilde{x}).
\]

The vector of coefficients is updated only after all the instances have been visited.

3. **Iterative Bayes**

The naive Bayes classifier builds for each attribute a two-contingency table that reflects the distribution on the training set of the attribute-values over the classes.

Consider the Balance-scale data set [1]. This is an artificial problem available at the UCI repository. This data set was generated to model psychological experimental results. This is a three-class problem, with four continuous attributes. The attributes are the left weight, the left distance, the right weight, and the right distance. Each example is classified as having the balance scale tip to the right, tip to the left, or to be
balanced. The correct way to find the class is the greater of left_distance × left_weight and right_distance × right_weight. If they are equal, it is balanced. There is no noise in the data set.

Because the attributes are continuous the discretization procedure of naive Bayes applies. In this case each attribute is mapped to five intervals. In an experiment using 565 examples in the training set, we obtain the contingency table for the attribute left_W that is shown in Table 1.

After building the contingency tables from the training examples, suppose that we want to classify the following example:
left_W: 1, left_D: 5, right_W: 4, right_D: 2, Class: Right.

The output of the naive Bayes classifier will be something like:

Observed Right Classified Right [ 0.277796 0.135227 0.586978 ].

It says that a test example that is observed to belong to class Right is classified correctly. The following numbers are the probabilities that the example belongs to each one of the classes. Because the probability $P(\text{Right}|\tilde{x})$ is greater, the example is classified as class Right. Although the classification is correct, the confidence on this prediction is low (59%). Moreover, taking into account that the example belongs to the training set, the answer, although correct, does not seem to fully exploit the information in the training set.

This is the problem that we want to address in this paper: Can we improve the confidence levels of the predictions of naive Bayes, without degrading its performance? The method that we propose begins with the contingency tables built by the standard naive Bayes scheme. This is followed by an iterative procedure that updates the contingency tables. The algorithm iteratively cycles through all the training examples. For each example, the corresponding entries in the contingency tables are updated in order to increase the confidence on the correct class. Consider again the previous training example. The value of the attribute left_W is 1. This means that the values in column I1 in Table 1 are used to compute the probabilities of Eq. (3). The desirable update will increase the probability $P(\text{Right}|\tilde{x})$ and consequently decreasing both $P(\text{Left}|\tilde{x})$ and $P(\text{Balanced}|\tilde{x})$. This could be done by increasing the contents of the cell (I1; Right) and decreasing the other entries in the column I1. The same occurs for all the attribute-values of an example. This is the intuition behind the update schema that we follow. Also the amount of correction should be proportional to the difference $1 - P(C_i|\tilde{x})$. 

---

### Table 1

A naive Bayes contingency table

<table>
<thead>
<tr>
<th>Attribute: left_W (discretized)</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>I5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>14.0</td>
<td>42.0</td>
<td>61.0</td>
<td>71.0</td>
<td>72.0</td>
</tr>
<tr>
<td>Balanced</td>
<td>10.0</td>
<td>8.0</td>
<td>8.0</td>
<td>10.0</td>
<td>9.0</td>
</tr>
<tr>
<td>Right</td>
<td>86.0</td>
<td>66.0</td>
<td>49.0</td>
<td>34.0</td>
<td>25.0</td>
</tr>
</tbody>
</table>
The contingency table for the attribute $left_W$ after the iterative procedure is given in Table 2.1 Now, the same previous example, classified using the contingency tables after the iteration procedure gives

$\text{Observed Right Classified Right } [\begin{array}{c}0.210816 \\ 0.000175 \\ 0.789009\end{array}]$.

The classification is the same, but the confidence level of the predict class increases while the confidence level on the other classes decreases. This is the desirable behavior.

The iterative procedure uses a hill-climbing algorithm. At each iteration, all the examples in the training set are classified using the current contingency tables. The evaluation of the actual set of contingency tables is done using the following equation:

$$
\frac{1}{n} \sum_{i=1}^{n} \left(1.0 - \argmax_{j} p(C_j | \tilde{x}_i)\right)
$$

where $n$ represents the number of examples and $j$ the number of classes. The iterative procedure proceeds while the evaluation function decreases till the maximum of 10 iterations.

The pseudo-code for the update function is shown in Fig. 1. To update the contingency tables, we use the following heuristics:

1. If an example is correctly classified then the increment is positive, otherwise it is negative. To compute the value of the increment we use the following heuristic $(1.0 - p(\text{Predict} | \tilde{x}))/#\text{Classes}$. That is, the increment is a function of the confidence on predicting class $\text{Predict}$ and of the number of classes.

2. For all attribute-values observed in the given example, the increment is added to all the entries for the predict class and half of the increment is subtracted to the entries of all the other classes.

The contingency tables are incrementally updated each time a training example is seen. This implies that the order of the training examples could influence the final results. This set of rules guarantees that after one example is seen, the update schema

---

1 The update rules tries to maintain constant the number of examples, that is the total sum of entries in each contingency table. Nevertheless, we avoid zero or negative entries. In this case the total sum of entries could exceed the number of examples.
Input:
A contingency Table, a given Example, predicted class, increment

Procedure Update(Table, Example, predict, increment)
For each Attribute
   For each Class
      If (Class == predicted)
         Table(Attribute, Class, Example(Attribute value)) +=increment
      Else
         Table(Attribute, Class, Example(Attribute value)) -=increment
      EndIf
   Next Class
Next Attribute
End

Fig. 1. Pseudo-code for the update function. The increment is a positive real number if the example is correctly classified and a negative number otherwise.

will increase the probability of the correct class. Moreover, there is no guarantee of improvement for a set of examples.

3.1. Iterative Bayes and loss functions

In classification problems the usual goal is to minimize the error rate. The prediction for each test example is either correct, if the prediction agrees with the actual value, or incorrect, if it does not. This is called a 0–1 loss function: the loss is either 0 if the prediction is correct, or 1 if it is not. Most learning algorithms, namely naive Bayes, can associate a probability to each prediction. It is natural to take this probability into account when judging correctness.

One criterion that is frequently used to evaluate probabilistic prediction is the quadratic loss function. For a single example the quadratic loss function is given by

\[ \sum_j (p_j - t_j)^2, \]

(8)

where \( p_j \) represents the probability that the example belongs to class \( j \) given by the learning model and \( t_j \) represents the true probability that the example belongs to class \( j \). In classification problems \( t_j \) is not known. Usually \( t_j \) is 1 for the true class and 0 for all the other classes. Eq. (8) can be written as

\[ 1 - 2p_i + \sum_j p_j^2, \]

(9)

where \( i \) represents the correct class and \( j \) all the others. The update rules of Iterative Bayes always increase \( p_i \). For a single example the update rules decreases the quadratic loss. For several examples the loss function is summed over them all. As many other
algorithms that employ heuristic search there is no guarantee of convergence to an optimal minimum.

3.2. Discussion

Elkan [6] has presented a successful boosting naive Bayes algorithm by re-weighting the examples. Later, Ridgeway et al. [20] extend the work of Elkan by combining the different models required by boosting into a single one, improving the interpretability of the final model. This is done by re-writing the AdaBoost combining schema in the form of log-odds and decomposing the sigmoid function using the Taylor approximation. The update schema in Iterative Bayes can be regarded as similar to the re-weighting schema in boosting. In Boosting each iteration increases and decreases the weight of the misclassified and the well-classified examples, respectively. Moreover, Boosting generates a set of classifiers that are aggregated by weighted voting. In Iterative Bayes there is only one classifier that is successively revised. The changes in the model are guided by the increase of the probability of the prediction in the correct class.

One hypothesis as to the good performance of boosting algorithms is that they increase the margins on the training data and this gives better performance on test data [22]. The margin is defined as the difference between the probability predicted for the actual class and the highest probability predicted for the other classes. Schapire shows that boosting increases the margin exponentially with the number of aggregated classifiers. Iterative Bayes linearly increases the margin for each example.

Recently, Nigam et al. [18] have presented an algorithm for learning from labeled and unlabeled documents based on a combination of the Expectation-Maximization (EM) algorithm and a naive Bayes classifier. In a first step a naive Bayes classifier is trained from the labeled examples. In a second step the EM is applied to estimate the missing data, that is the class of the unlabeled examples. In the E-step the current classifier is used to estimate the probability that each document belongs to each class and the M-step re-estimate the classifier using the class membership given by the E-step. In Iterative Bayes, only the labeled examples from the training set are used. Moreover, we generate a model that is incrementally adapted each time one example is re-seen.

4. Empirical evaluation

We have compared a standard naive Bayes against our Iterative proposal on 27 benchmark data sets from UCI [1].

For each data set we estimate the error rate of each algorithm using a 10-fold stratified cross-validation. To minimize the variance of the training set, we repeat this process 10 times, each time using a different permutation of the data set. The final estimator of the error is the average of the 10 iterations.

The results are presented in Table 3. The empirical evaluation shows that the proposed method not only improves the error rate on 20 benchmark data sets but also improves the global error rate. Results have been compared using paired t-tests. The confidence level was set to 99.9%. A + (−) sign means that the Iterative Bayes
obtains a better (worse) result with statistical significance. On 13 data sets the Iterative Bayes produces better results and only loses in one data set. A summary of comparative statistics is presented in Table 4. Note that the means of the error rate are significantly different with high probability as given by the $p$ value using the Wilcoxon rank-signed paired-test.
4.1. Bias–variance decomposition

The Bias–variance decomposition of the error rate [11] is a useful tool to understand the behavior of learning algorithms. Using the decomposition proposed in [11] we have analyzed the error decomposition of naive Bayes and Iterative Bayes on the data sets under study. To estimate the bias and variance, we first split the data into training and evaluation sets. From the training set we obtain 10 bootstrap replications used to build 10 classifiers. We ran the learning algorithm on each of the training sets and estimate the terms of the variance and bias using the generated classifier for each point \( x \) in the evaluation set. All the terms were estimated using frequency counts.

Fig. 2 presents, for each data set, a comparison between the algorithms. On these data sets, we verify that the reduction of the error rate observed with Iterative Bayes is mainly due to a reduction on the bias component. Table 5 presents an average summary of the results.

<table>
<thead>
<tr>
<th></th>
<th>Bias</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>16.87</td>
<td>1.7</td>
</tr>
<tr>
<td>Iterative Bayes</td>
<td>14.42</td>
<td>2.2</td>
</tr>
</tbody>
</table>

4.2. Learning times

Iterative Bayes is necessarily slower than naive Bayes. Fig. 3 shows the accumulative learning times for the experiments reported in this paper. This figure suggests that Iterative Bayes has the same time complexity as Bayes although with greater slope.
5. The attribute redundancy problem

The naive Bayes is known to be optimal when attributes are independent given the class. To classify a new instance naive Bayes uses all the attributes. So, it can be misleading by the presence of both redundant and irrelevant attributes. An irrelevant attribute does not affect the underlying structure of the data, a redundant attribute does not provide anything new in describing the underlying structure of the data [15]. In this section we examine the behavior of Iterative Bayes in the presence of redundant and irrelevant attributes.

To have insights into the influence of attribute dependencies in the behavior of Iterative Bayes, we have performed a set of controlled experiments using the Balance-scale UCI data set. We have chosen this data set because it is noise free and all the original attributes are relevant to determine the correct class.

5.1. Redundant attributes

We ran naive Bayes and Iterative Bayes on the original Balance-scale data set. We obtained a redundant attribute by duplicating one attribute. We re-ran naive Bayes and Iterative Bayes on the new data set. Table 6 presents the results of duplicating one, two, and three attributes. The results were obtained using 10-cross validation.

We can observe that while the error rate of the naive Bayes doubles, the Iterative Bayes almost maintains the same error rate. While naive Bayes strongly degrades its performance in the presence of a redundant attribute the Iterative Bayes is not affected.

5.2. Irrelevant attributes

In this case we obtain irrelevant attributes using the expression $k \times \text{Random}()$. We have evaluated the behavior of Iterative Bayes using one, two and three irrelevant attributes using different values of $k$. Table 6 presents the results using 10-cross val-
Table 6
A case study on redundant and irrelevant attributes

<table>
<thead>
<tr>
<th>Redundant attributes</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original data</td>
<td>1 att. red.</td>
<td>2 att. red.</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>$9.11 \pm 1.6$</td>
<td>$17.43 \pm 3.8$</td>
<td>$17.08 \pm 4.1$</td>
</tr>
<tr>
<td>Iterative Bayes</td>
<td>$7.99 \pm 1.3$</td>
<td>$7.19 \pm 2.1$</td>
<td>$6.88 \pm 2.6$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Irrelevant attributes</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original data</td>
<td>1 att. irr.</td>
<td>2 att. irr.</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>$9.11 \pm 1.6$</td>
<td>$12.46 \pm 2.9$</td>
<td>$13.07 \pm 4.8$</td>
</tr>
<tr>
<td>Iterative Bayes</td>
<td>$7.99 \pm 1.3$</td>
<td>$8.32 \pm 1.6$</td>
<td>$9.28 \pm 2.4$</td>
</tr>
</tbody>
</table>

idation. Again we observe that Iterative Bayes is much more resilient to noise than naive Bayes. These results indicate that Iterative Bayes could be a method to reduce the well-known bottleneck of attribute dependencies in naive Bayes learning.

6. Conclusions and future work

In this paper we have presented an iterative approach to naive Bayes. The Iterative Bayes begins with the distribution tables built by the naive Bayes followed by an optimization process. The optimization process consists of an iterative update of the contingency tables in order to improve the probability class distribution associated with each training example. Instead of minimizing a 0–1 loss function as usual in classification problems, Iterative Bayes minimizes a quadratic loss function. The Iterative Bayes uses exactly the same representational language as naive Bayes. As such, both models have the same degree of interpretability. Experimental evaluation of Iterative Bayes on 27 benchmark data sets shows minor but consistent gains in accuracy. In 13 data sets statistical significant gains were obtained, and only in one data set a significant degradation was observed. An interesting side effect of our algorithm is that it shows a stronger independence of attribute dependencies.

Future research will explore different updates rules. Moreover, the independence over redundant attributes requires a closer examination. We intend to perform a more extensive study of this issue.

Acknowledgements

Gratitude is expressed to the financial support given by the FEDER and PRAXIS XXI, the Plurianual support attributed to LIACC, Esprit LTR METAL project, Sol-Eu-Net project, and ALES project. I would like to express my gratitude to the anonymous referees for their constructive comments.
References


