# A Lazy Model-Based Algorithm for On-Line Classification

Gabor Melli

DataSage, Inc. www.datasage.com
gmelli@datasage.com

**Abstract.** This paper presents a *lazy model-based* algorithm, named DBPredictor, for on-line classification tasks. The algorithm proposes a local discretization process to avoid the need for a lengthy preprocess stage. Another advantage of this approach is the ability to implement the algorithm with tightly-coupled SQL relational database queries. To test the algorithm's performance in the presence of continuous attributes an empirical test is reported against both an *eager model-based* algorithm (C4.5) and a *lazy instance-based* algorithm (*k*-NN).

## 1 Introduction

The large number of structured observations now stored in relational databases has created an opportunity for classification tasks that require a prediction for only a single event. This type of prediction will be referred to as on-line classification, to differentiate it from classification tasks that allow for batch style model induction. This paper proposes an algorithm, called DBPredictor, for such on-line classification tasks. The algorithm performs a top-down heuristic search through the `IF` *antecedent* `THEN` *consequent* rule space of the specific event to be classified. The two challenges addressed in this paper are the local discretization of numerical attributes and a tightly-coupled implementation against SQL based databases.

Section 2 contrasts the lazy model-based approach to classification form other well known approaches. Section 3 describes DBPredictor with a focus on the support for numerical attributes and an SQL interface. Section 4 presents the results of an empirical investigation into DBPredictor's accuracy with respect to the number of numerical attributes. Finally, Section 5 concludes with a paper summary.

## 2 Previous Work

An algorithm for on-line classification tasks must decide whether to use an *eager* or *lazy* approach, and whether it should be *model-based* or *instance-based*. Eager algorithms induce a complete classification structure (classifier) before they can process any classification requests. Lazy algorithms, on the other hand, commence to work immediately on classifying the given event [3]. Model-based algorithms, represent their result in a language that is richer than the language used to describe the dataset, while instance-based algorithms represent their result in the same language that is used to described the dataset [11].

Based on these descriptions, on-line classification tasks would likely benefit from a *lazy model-based* algorithm. Such an algorithm would focus its effort on classifying the particular event in question, and would also return a rationale that may help the person interpret the validity of the prediction. Two recent proposals that do make use of a lazy model-based approach include of former version of DBPredictor [8] and the LazyDT [6] (Lazy Decision Tree) algorithms. The main difference between these algorithms is the use of a rules in one and the use of decision tree paths in the other.

These two algorithms however, cannot be tightly-coupled with a SQL based relational database [2, 7] because they require datasets to be both discretized and stored in memory. The version of DBPredictor presented in this paper addresses both these issues and presents a *tightly-coupled* implementation of the SQL Interface Protocol (SIP) proposal [2, 7]. Other details of this algorithm are presented in [9].

## 3 Algorithm

DBPredictor requires three input parameters: a partially instantiated event $\boldsymbol{e}$; the attribute whose value is to be predicted $A_c$; and a dataset $D$ from the same domain as $\boldsymbol{e}$. With this information, DBPredictor performs a search through a constrained space of all the applicable `IF` *antecedent* `THEN` *consequent* classification rules. The search starts from a general *seed rule* that covers $\boldsymbol{e}$. The algorithm then generates several candidate rules that are more specialized than the previous rule. To determine which rule to further specialize on, the candidate rules are tested with a heuristic function `F()`. The rule that achieves the highest value at each specialization step is the one selected for the next round of specialization. The search proceeds until a stopping criterion is encountered.

### 3.1 `top_down_search()`

Once the seed rule $r_0$ has been composed, DBPredictor simply outputs the rule returned by a call to `top_down_search` with parameters $(r_0, (D, \boldsymbol{e}, A_c))$. This procedure performs a greedy top-down search through a constrained rule space. Procedure 3.1 presents a pseudo-code overview of `top_down_search()`. Only two of its four subprocedures are described in detail in the coming sections: `generate_antecedents()`, and `get_consequent()`. The heuristic function `F()` can be any impurity measure. Several heuristics, such as entropy and Euclidean distance have been successfully tested in [9]. Finally, `best_rule()` selects the rule with the highest estimated predictive value.

### 3.2 `generate_antecedents()`

The `generate_antecedents()` procedure returns the set of rules to be tested by the heuristic function. To constrain the search space from the $2^n$ possible attribute-value combinations, the procedure returns only the, up to $n-1$ rules possible by specializing on each of the $n$ attributes.

**Procedure 3.1** `top_down_search()`

**Input:** $(r, P)$: rule $r$ and algorithm parameters $P \Leftarrow (D, \boldsymbol{e}, \boldsymbol{c})$.
**Output:** A rule $r'$ that covers $\boldsymbol{e}$ but which cannot be further specialized.
**Method:**

```
 1: R ⇐ generate_antecedents(r, P)
 2: for all rule r' ∈ R do
 3:    r'_conseq. ⇐ get_consequent(r', P)
 4:    r'_value ⇐ F(r', r)
 5: end for
 6:
 7: best_r' ⇐ best_rule(R)
 8: if (best_r' ≠ ∅) then
 9:    return(top_down_search(best_r', P))
10: else
11:    return(r)
12: end if
```

**Symbolic Attributes** The first time a specialization is attempted on a proposition that refers to a symbolic attribute $A_i$, the proposition is simply updated from $(A_i = ANY)$ to $(A_i = \boldsymbol{e}_i)$. If the proposition on this attribute has already been specialized, then no further specialization is attempted.

**Numerical Attributes** The method described above for symbolic attributes cannot be successfully applied to continuous attributes. If, for example, $\boldsymbol{e}_2 = 6.5$ and the range on attribute $A_2$ is [0.5, 9.0], then generating the proposition $(A_2 = 6.5)$ would likely result in a rule that covers few, if any, records in the dataset. One way to overcome this situation is by discretizing all continuous attributes in the dataset before using the classification algorithm. This approach could be thought of as eager discretization because many of the regions that are made discrete are not necessary for the single classification task at hand. DBPredictor instead uses a two sided test $(A_i \in [\boldsymbol{e}_i - \delta, \boldsymbol{e}_i + \delta])$, where $\delta > 0$, on continuous attributes. After a scan through the dataset to locate the $min, max$ range for each attribute, he $\delta$ for each proposition in the seed rule is set to the larger of $(max - \boldsymbol{e}_i)$ and $(\boldsymbol{e}_i - min)$. The proposition in the example above would be initialized to $(A_2 \in [6.5 - 6, 6.5 + 6)$.

At each specialization DBPredictor makes $\delta'$ strictly smaller than the previous $\delta$ used by the parent's proposition. The decrease in $\delta$ between iterations is determined by an internally set fraction named `num_ratio` (`num`erical partitioning `ratio`):

$$\delta' \Leftarrow \frac{\delta}{\texttt{num\_ratio}}$$
$$P'_i \Leftarrow \left(A_i \in [\boldsymbol{e}_i - \delta', \boldsymbol{e}_i + \delta']\right)$$

An empirically determined default value for `num_ratio` is presented in Section 4.

### 3.3 `get_consequent()`

Once a set of rule antecedents have been generated, the `get_consequent()` procedure is used to construct each rule's consequent. For a given rule's antecedent ($r.antecedent$), the procedure executes the following SQL query to return a summary of attribute $A_c$ for all the dataset records that match the antecedent: `SELECT` $A_c$, `COUNT(*) FROM` $D$ `WHERE` $r.antecedent$ `GROUP BY` $A_c$. This implementation has the advantage that it does not require the usage of temporary tables nor the existence of a key attribute in the dataset.

## 4 Empirical Results

An empirical study was conducted to test DBPredictor's accuracy with respect to the proportion of continuous attributes in the dataset.[1] Twenty three datasets from the UCI repository [10] were used in this study: anneal, heart-h, audiology, hepatitis*, breast-w, horse-colic, chess, iris*, credit-a, letter, credit-g, liver-disease, diabetes, mushroom, echocardiogram*, segment, glass, soybean-small, hayes-roth*, tic-tac-toe*, heart, vote, and heart-c. An attempt was made to include previously studied datasets [6, 5, 12] with a wide variety of sizes and proportions of continuous attributes[2]. The benchmark algorithms for this study were the C4.5 r8 decision tree algorithm [11, 12] and the IB1 k-nearest neighbor algorithm [4]. Finally, each algorithm's true error rate on each dataset was estimated with the use of multiple ten-fold stratified cross-validation tests.

### 4.1 Tuning `num_ratio`

The first portion of the empirical study determined an appropriate value for DBPredictor's `num_ratio` internal parameter. All three algorithm's were tuned on five datasets marked with a * beside its name. The value for `num_ratio` that achieve the lowest average error rate on these five datasets was 1.5. This value was passed on to the next study. Similarly, the IB1 algorithm's *k* was set to 5 after being tuned on the same five datasets as DBPredictor.

### 4.2 Continuous Domains

To test for bias with respect to the proportion of numerical attributes in a dataset the three pairwise combinations between DBPredictor, C4.5 and IB1 were contrasted. For each pair, the datasets in which one algorithm performed significantly better[3] than the other were identified. Table 1 summarizes the results of the three pairwise tests. Rather than being skewed to a strong bias for or against continuous attribute, the results suggest that DBPredictor's bias is instead situated between that of IB1 and C4.5.

[1] An ANSI-C implementation of the algorithm can be downloaded from the `www.cs.sfu.ca/~melli/DBPredictor` Web site.

[2] the 23 datasets possessed on average 48% numerical attributes and ranged from 0% to 100%

[3] based on a two-tailed $t$-test with 99.5% confidence

**Table 1.** Average percentage of numerical attributes in the datasets that each algorithm performed significantly more accurately than another algorithm. The datasets that DBPredictor was more accurate than IB1 had 39% numerical attributes on average.

| DBP/C4.5 | DBP/IB1 | C4.5/IB1 |
|---|---|---|
| 54%/55% | 39%/52% | 20%/57% |

## 5 Conclusion

This paper presents an algorithm, named DBPredictor, that is targeted to on-line classification tasks. These tasks require the prediction of a single event's class, based on the records stored in a relational database. DBPredictor uses a lazy model-based approach in that it performs only the work it requires to classify the single event. The opportunity presented in this paper is the use of proposition specialization. A tightly-coupled SQL based implementation of the algorithm is presented, along with the results of an empirical study into the relative bias for or against datasets with numerical attributes.

## References

1. AAAI. *Thirteenth National Conference on Artificial Intelligence*. AAAI Press, 1996.
2. R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation, and experience. *IEEE Trans. Knowledge and Data Engineering*, 8:962–969, 1996.
3. D. W. Aha, editor. *Lazy Learning*. Kluwer Academic, May 1997.
4. D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
5. P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2):141–168, August 1996.
6. J. H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. [1], pages 717–724.
7. G. H. John and B. Lent. SIPping from the data firehose. In *Proceedings, Third International Conference on Knowledge Discovery and Data Mining*, pages 199–202. AAAI Press, 1997.
8. G. Melli. Ad hoc attribute-value prediction. [1], page 1396.
9. G. Melli. Knowledge based on-line classification. Master's thesis, Simon Fraser University, School of Computing Science, April 1998.
10. P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science, 1995. ftp://ics.uci.edu/pub/machine-learning-databases.
11. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
12. J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, March 1996.