# On the Inverse Classification Problem and its Applications[*]

Charu C. Aggarwal
IBM T. J. Watson Research Center
charu@us.ibm.com

Chen Chen, Jiawei Han
University of Illinois at Urbana Champaign
{ cchen37, hanj }@cs.uiuc.edu

## Abstract

*In this paper, we discuss the* inverse classification problem*, in which we desire to define the features of an incomplete record in such a way that will result in a* desired *class label. Such an approach is useful in applications in which it is an objective to determine a set of actions to be taken in order to guide the data mining application towards a* desired *solution. This system can be used for a variety of decision support applications which have pre-determined task criteria.*

**Keywords:** Classification, Decision Support

## 1. Introduction

The classification problem has been widely studied in the literature because of its applicability to a wide variety of problems [2, 3]. In this paper, we introduce the interesting and related problem problem of *inverse classification*. The inverse classification problem is one in which it is desirable to determine the feature variables for an *incompletely specified* test data set. Typically, these feature variables are decision variables for an optimization or decision support application. The aim is to decide these feature variables in such a way so as that the resulting records would belong to a set of *desired* class variable values for the test data set. For the case of the training data set, both the feature and class variables are completely defined in $\mathcal{D}_{train}$. On the other hand, for the case of the test data set $\mathcal{D}_{test}$, the class variables are completely defined but the feature variables are not. Thus, each test data example has a *desired class label* associated with it. The aim of the inverse classification problem is to choose the test feature variables such that the corresponding classification accuracy with respect to the *desired* test classes is maximized.

The inverse classification problem can be used to solve a number of applications in which the features can be used to define certain actions which drive the decision support system towards a *desired* end-result. For example, in a mass marketing application, it may be desirable to send various kinds of mailers to customers in order to solicit responses from customers. The training database class variables contain 0-1 action-driven values corresponding to the true participation behavior based on certain actions. We note that while a single-attribute version of this problem is solvable using a sensitivity analysis approach on standard classification methods, it is necessary to define the more sophisticated inverse classification problem in order to model the effect of a *combination of* different action and decision variables. A related class of methods is that of re-inforcement learning [4], in which it is desirable to learn the variables of a Markov decision process in order to learn a specific outcome. However, re-inforcement learning is tailored to process-optimization criteria, whereas the inverted classification problem is tailored to a large class of problems where large amounts of training data are already available from previously tested processes. In such cases, it is prudent to design the inverse classification problem to leverage on the pre-defined training data.

We assume that the training data set $\mathcal{D}_{train}$ contains $N$ records which are denoted by $\overline{X_1} \dots \overline{X_N}$. In addition, each record is associated with class labels denoted by $l_1 \dots l_N$. We also assume that the data contains a total of $k$ class labels which are denoted by $\{1 \dots k\}$. The subsets of the data for each of the $k$ classes are denoted by $\mathcal{D}^1_{train} \dots \mathcal{D}^k_{train}$. We also assume that the cardinality of these $k$ classes are denoted by $p_1 \dots p_k$. Thus, we have:

$$\sum_{i=1}^{k} p_i = N \tag{1}$$

The test data set contains $M$ records which are denoted by $\overline{Y_1} \dots \overline{Y_M}$ along with *desired* class labels denoted by $q_1 \dots q_M$. The records $\overline{Y_1} \dots \overline{Y_N}$ are incompletely defined since some of the fields may be missing. These missing fields (denoted by $\mathcal{M}$) are typically action-oriented or decision variables which need to be chosen in order to maxi-

[*]This work was supported in part by National Science Foundation under Grants IIS-02-09199/IIS-03-08215, and an IBM faculty award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the funding agencies.

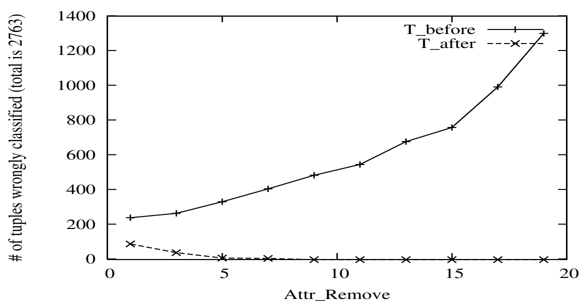

**Figure 1. Naive Bayesian: Accuracy boost on mushroom with increasing number of removed attributes**

mize the classification accuracy of the test data by a training model into the desired class label. We assume that the missing fields in the test data are denoted by $\mathcal{M}$.

## 1.1. The Inverse Classification Algorithm

These missing entries need to be chosen in order to direct the classification of the test records towards the desired class label. For the purpose of ease in exposition, we will first describe the algorithm assuming that all the attributes in the data are categorical. We note that this assumption is without loss of generality since we can use the process of discretization in order to convert the quantitative variables into categorical form. We will discuss the detailed process of discretization slightly later. As discussed earlier, the training data contains $N$ records such that each of these records contains $d$ dimensions. We assume that the number of possible categorical values for the $i$th dimension is denoted by $v(i)$, For the dimension $i$, the categorical values are denoted by $a_1^i \ldots a_{v(i)}^i$. For each categorical value for dimension $i$, we maintain an inverted list containing all the records which take on that particular value. We assume that the data points in the $q$th list for dimension $i$ is denoted by $L(i, q)$. The number of elements in each inverted list is equal to the number of records taking on that particular value. Thus, the total number of inverted lists $n_I$ is defined as follows:

$$n_I = \sum_{i=1}^{k} v(i) \tag{2}$$

The first step is to convert the entire training data into the inverted representation. This is key to making the algorithm scalable for very large databases. Let us consider a test example $T$, which has $q$ missing attributes. Let us assume that the index of the missing attributes are denoted by $i_1 \ldots i_q$. We note that the attributes can take on $v_{i_1} \ldots v_{i_q}$ possible values. We denote the number of possible combinations of filling the missing variables by $C(i_1 \ldots i_q)$. This is given by:

$$C(i_1 \ldots i_q) = \pi_{j=1}^{q} v_{i_j} \tag{3}$$

Thus, the number of possibilities for picking the missing attributes increases exponentially in practice. Therefore, it is necessary to design an algorithm which can efficiently search through the space of exponential possibilities in order to pick the missing attributes in the optimum way. In order to achieve this goal, we will use the process of constructing the combination of attributes in a bottom up roll fashion. The inverse classification algorithm uses as input the test example $T$, the target class index $q$, the gini threshold $a$, and the support threshold $s$. The support and gini thresholds define a bound used to prune the sets of dimensions which are not sufficiently discriminatory for the exploratory process. In order to achieve this goal, we use a rollup technique to construct those sets of dimensions which are extremely biased towards the desired class label. This is used to create the set $\mathcal{LF}$ which defines the combinations of decision variables which are biased towards the desired class label. The inverted representation of the data plays a key role in being able to perform the entire process in an efficient way. As a result, the algorithm can be scaled up to very large databases. A detailed description of the algorithm may be found in [1]. Next, we use the combinations of dimensions defined by $\mathcal{LF}$ to define the missing variables of $T$. We pick the set in $\mathcal{LF}$ with the next highest gini-index in order to insert values into the set $T$. This process is repeated until all values in $T$ have been filled, or the set $\mathcal{LF}$ is empty. In the event that some value of $T$ has not been filled, then the unfilledvalues correspond to the average behavior of all records belonging to the target class. In the event of categorical attributes, we pick the categorical value which occurs most frequently for the target class. A detailed description of this process may be found in [1].

## 2. Experimental Results

All our experiments are implemented by Microsoft Visual C++ 6.0 and run using a 3GHz Pentium IV machine with 1GB main memory. A key issue is the choice of data sets in order to test the algorithm. This is because there are no standard benchmarks available for the problem, nor are there readily available data sets corresponding to such decision support applications. Since there are no data sets available for testing the inverse classification problem, we need to use the data sets for testing the standard classification problem. We will describe our results on the mushroom data set from the UCI machine learning repository. More results may be found in [1]. The data set was randomly divided into a training and testing part. Specifically, two-thirds of the data was used for training and the remaining was used for testing. The training data was preserved

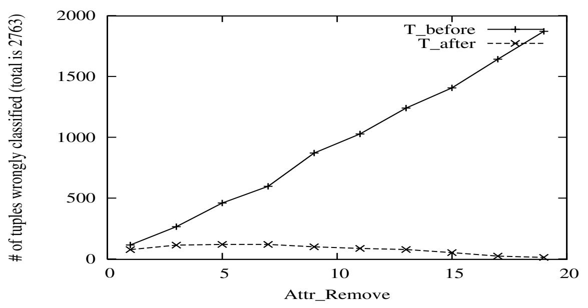

**Figure 2. REPTree: Accuracy boost on mushroom with increasing number of removed attributes**

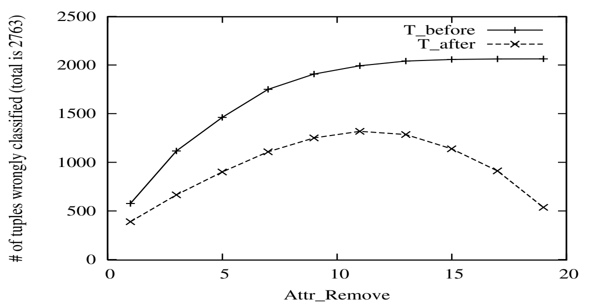

**Figure 3. Decision Table: Accuracy boost on mushroom with increasing number of removed attributes**

in its original state whereas the test data was used to create the missing action driven attributes. Specifically, for each tuple, we removed $Attr\_Remove$ entries. The iclass algorithm was used to determine the action driven missing entries. A number of off-the-shelf classifiers were used in order to determine the classification quality of the newly constructed data set on these entries. At the same time, we also determined the classification accuracy on the data set with a reduced number of attributes. Care was taken to pick a variety of different classifiers, so that there was no overfitting between the particular classifier being used and the methodology of the iclass algorithm for determining the entries in the data. Specifically, we used implementations of the Naive Bayesian classifier, REPTree, and Decision Table classifiers. These classifiers were obtained from the open-source WEKA web site [5]. The classifiers are conceptually very different from the lazy learners used in the inverse classification algorithm in order to decide the unfilled attributes.

We determined the classification accuracy using two different methods: (1) We illustrate the classification accuracy on the test data with the reduced number of attributes. The training is performed on the same set of reduced attributes. We refer to this accuracy as $T_{before}$. The accuracy of the classification process will typically reduce with fewer number of attributes. (2) For each case in which we tested with a reduced number of attributes, we used the inverted classification algorithm to fill in the missing attributes. Then, we tested the classification accuracy on the modified data with the filled in attributes. The corresponding classification accuracy is referred to as $T_{after}$.

For each case, we computed the accuracy of the classification process with increasing number of removed attributes.The results are illustrated in Figures 1, 2, and 3 respectively for the Bayes classifier, decision tree, and decision table classifiers respectively on the mushroom data set. In most cases, the value of the accuracy before substitution of the missing attributes ($T_{before}$) reduces with increasing number of missing attributes. This is because of the reduced amount of information in the smaller number of attributes. In each case, the accuracy of the classifier showed a considerable boost over the *original* data set because of the modification of the attributes in an optimized way by the inverted classification process. This is because the inverse classification algorithm learns the optimum combination of attribute values which result in the greatest matching of the feature variables to the target class label. This is useful in a decision support application, in which it is desirable to learn the most appropriate categorical decision variables for classification. The overall summary of these results is that the inverted classification approach can lead to improvements in the quality of the data, and this improvement increases with the number of attributes being replaced. In the context of a decision support application, this means that the inverted classification approach is most effective when there are a large number of decision variables. This is also the most interesting case in a real setting.

## References

[1] C. Aggarwal, C. Chen, J. Han. On the Inverse Classification Problem and its Applications. *IBM Research Report*, 2006.

[2] R. Duda, P. Hart, D. Stork. *Pattern Classification*, 2nd Edition, John Wiley and Sons, Inc.: New York, 2001.

[3] M. James. Classification Algorithms, *Wiley*, 1985.

[4] R. Sutton, A. Barto. Re-inforcement Learning: An Introduction. *MIT Press*, Cambridge, MA, 1988.

[5] I. Witten, E. Frank. Data Mining: Practical Machine Learning Tools and Techniques. *Morgan Kaufmann*, 2005.
**URL:** http://www.cs.waikato.ac.nz/˜ml/weka/book.html