RC24776 (W0903-107) March 26, 2009
Computer Science

IBM Research Report

A Framework for Inverse Classification

Charu C. Aggarwal
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Chen Chen, Jiawei Han
University of Illinois at Urbana Champaign

A Framework for Inverse Classification

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 determine the features to be used to create a record which 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 predetermined task criteria. We will show that the inverse classification problem is a powerful and general model which encompasses a number of different criteria. We propose a number of algorithms for the inverse classification problem which use an inverted list representation for intermediate data structure representation and classification. We validate our approach over a number of real data sets.

1 Introduction

The classification problem has been widely studied in the literature because of its applicability to a wide variety of problems such as customer segmentation, modeling, and pattern recognition. A huge amount of research [1, 2, 3, 4, 5, 6, 7, 9, 12, 13, 14] has been dedicated to the development of scalable and effective classification algorithms. In this paper, we study an interesting and related problem problem, which we refer to as *inverse classification.* In the inverse classification problem, we would like to determine the *action oriented* 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 it. On the other hand, for the case of the test data set, 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. We note that the inverse classification problem is different from the classification or imputation problem on missing data sets. In the classification problem on missing data, we try to determine the *unknown* class variable with incompletely defined features. On the other hand, in the inverse classification problem, we try to determine the *action-oriented* missing variables in order to achieve a *desired result* which is reflected in the class variable. The inverse classification problem is useful for a number of action-driven applications in which the features can be used to define certain actions which drive the decision support system towards a *desired* end-result. Some examples are as follows:

- In a mass marketing application, it may be desirable to send various kinds of mailers to customers in order to solicit responses from customers. Different kinds of mailers (eg. promotions, advertisements) may have different effectiveness on the responses of the customers. It is desirable to maximize the effectiveness of the campaign by using the correct set of mailers on the different customers. In this case, a number of feature variables may be defined corresponding to the different kinds of mailers which can be sent to the customers during a pre-specified horizon in time. In addition, the data may contain demographic or other features about the customer which are relevant to the likelihood of a positive response. We assume that the class variable is in binary form which indicates whether or not a given customer responded to a particular mailer. While the training database class variables contain 0-1 values corresponding to the true participation behavior, the test data class variables contain only 1-values corresponding to the fact that it is desirable for as many participants as possible to respond positively. Therefore, a high accuracy of test classification corresponds to a high participation rate. It is desirable to pick the action-driven feature variables such that the largest number of test examples are classified to the desired class label. 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.

- In a decision support application, it may be desirable to perform a set of actions such that the pre-defined criterion of the decision support system is optimized. We note that this pre-defined criterion may be defined in terms of the class label, and the feature variables correspond to the different actions. It is desirable to choose the feature variables such that the pre-defined criterion is optimized. While decision support systems are often designed with the help of ad-hoc optimization methods, the use of modeling techniques such as that of inverse classification can be particularly useful in creating a framework which can be used for transformation of arbitrary decision support applications.

A related class of methods is that of reinforcement learning [10, 15], in which it is desirable to learn the variables of a Markov decision process in order to learn a specific outcome. However, reinforcement 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 will define and model the inverse classification problem assuming that the attributes in the data are categorical. We will also discuss how to handle the case when the records in the data are quantitative. This is quite simple, since quantitative variables can be transformed to categorical form using discretization. We also note that since this technique will be used for action-driven applications in which the features define the actions that may or may not happen, it is more natural to use categorical variables for the incompletely defined features.

This paper is organized as follows. In the next section, we will introduce some modeling concepts of the inverse classification problem. In the same section, we will discuss some algorithms for the inverse classification problem. Section 3 will discuss the experimental results. Section 4 contains the conclusions and summary.

2 Inverse Classification Algorithm

We will first introduce some notations and definitions. We assume that the training data set $\mathcal{D}_{train}$ contains N records which are denoted by $\overline{X_1} \dots \overline{X_N}$. Each record in the training data contains d dimensions. In addition, the N records are associated with class labels denoted by $l_1 \dots l_N$, each of which is drawn from the set $\{1 \dots k\}$. Thus, there are a total of k classes. 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:

$$\sum_{i=1}^{k} p_i = N \tag{2.1}$$

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 are typically action-oriented or decision variables which need to be chosen in order to maximize the probability of a test instance classifying to the desired class label.

Our approach is to first design an intermediate representation (on the training data) which allows us to effectively examine the class behavior of different local subspaces of the data. For a given test example, we would like to use (local) combinations of dimensions which classify to the desired class label. The inverted representation of the data provides such a method for efficient access of such local combinations of dimensions.

2.1 Constructing the Inverted Statistics on the Training Data In this section, we will discuss a method for constructing the inverted statistics on the training data. The inverted statistics are useful in keeping a track of how well different local subspaces classify to the desired class label. As we will see, the process of inverted classification requires us to leverage on the efficient storage of these statistics, since the algorithm needs to search the space of missing attributes in order to find those combinations which classify to the desired class label.

For the purpose of ease in exposition, we will first introduce the notations assuming that all the attributes in the data are categorical. 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. 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 ith dimension is denoted by $v(i)$. For the dimension i, the categorical values are denoted by $a_1^i \dots a_{v(i)}^i$. For each categorical value for dimension i, we maintain an inverted list containing all the record identifiers which take on that particular value. In addition, we store the class label identifiers for the corresponding records. We assume that the data points in the qth 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:

$$n_I = \sum_{i=1}^{k} v(i) \tag{2.2}$$

We note that this simple level of storage allows us to compute the class statistics of both the individual lists as well as intersections of the inverted lists. This is useful for computing the class distribution statistics in different local subspaces of the data.

In the case of quantitative data sets, it is also possible to control the granularity of discretization in order to change the representation of the inverted list. As in the previous case, we assume that we have a total of $v(i)$ possible intervals for dimension i. In general, the value of $v(i)$ may be the same for all quantitative dimensions. This is achieved by using equi-depth discretization of the different dimensions. As in the previous case, we assume that the data points in the qth list of dimension i are denoted by $L(i, q)$.

It is also possible to use the inverted list along with a combination of quantitative and categorical attributes. This is because the different discretized values of the quantitative attributes can be treated as different values of a categorical attribute for the purpose of the inverted list representation. Therefore, we will use the above-mentioned notations such as $L(i, q)$ in a transparent way without reference to the nature of the underlying attribute.

The idea of using the inverted representation is that it is possible to examine the behavior of a local subspace by using an intersection of the data points in the different lists $L(i, q)$. For example, the data points in the local subspace corresponding to the qth list in dimension i and the r list in dimension j is given by the intersection of the lists $L(i, q)$ and $L(j, r)$. Since the class statistics are also stored in the inverted lists, the class behavior of that local subspace can also be accurately re-constructed. Since the inverted classification algorithm explores the class behavior of a potentially large number of subspaces, the ability to compute such sets of points efficiently is particularly useful.

Another statistic which we would like to be able to compute is the level of class discrimination among different subspaces. We note that not all lists are equally discriminative in terms of distinguishing between the different classes. Let $f(i, q, s)$ denote the fraction of records from the list $L(i, q)$ corresponding to the class indexed by $s \in \{1, \dots k\}$. This is easy to compute since we store the class labels along with the record identifiers in the inverted lists. Therefore, we have:

$$\sum_{s=1}^{k} f(i, q, s) = 1 \tag{2.3}$$

We would like to define the level of class discrimination among the data points in $L(i, q)$. Let $\mathcal{G}(S)$ be the gini-index for a set of data points. Then, we define the gini-index $\mathcal{G}(L(i, q))$ of list $L(i, q)$ as follows:

$$\mathcal{G}(L(i, q)) = \sum_{s=1}^{k} f(i, q, s)^2 \tag{2.4}$$

We note that the value of the gini-index $\mathcal{G}(L(i, q))$ lies in the range $(0, 1)$. The value of $\mathcal{G}(L(i, q))$ takes on a minimum value of $1/k$ when all classes are equally distributed in the corresponding set of points. This is the least discriminative case. When the entire list contains only labels from one class, the value of $\mathcal{G}(L(i, q))$ is equal to 1. This is the most discriminative case for classification.

For the case of the inverse classification problem, we are only interested in the discrimination behavior of those local subspaces corresponding to both the specified and missing attributes of test example T. A test example T is said to *activate* the qth list from dimension i, if the value of test example T for dimension i belongs to the value-range[1] for the qth list of dimension i. In general, we refer to the index of the list for dimension i for test example T, by $\mathcal{I}_T(i)$. Therefore, for a *fully specified* test example T, the relevant lists are denoted by $L(1, \mathcal{I}_T(1)) \dots L(i, \mathcal{I}_T(i)) \dots L(d, \mathcal{I}_T(d))$. We note that since a smaller number of lists are fully specified in most cases, a fewer number of lists are activated.

In general, the set of data points in the subspace intersection of the relevant lists for test example T and set of dimensions S is determined by the intersection of the corresponding inverted lists. This intersection is denoted by $\mathcal{Q}(S, T)$, and is defined as follows:

$$\mathcal{Q}(S, T) = \cap_{i \in S} L(i, \mathcal{I}_T(i)) \tag{2.5}$$

Thus, the subspace locality $\mathcal{Q}(S, T)$ is defined by the intersection of the data points in the relevant inverted

[1] For the case of categorical attributes, we use the relevant categorical value rather than the value-range.

lists for the test example T and set of dimensions S. We use the gini index $G(\mathcal{Q}(S,T))$ in order to quantify the level of discrimination in the set $\mathcal{Q}(S,T)$. This quantification can be used in order to define the discrimination of the set of dimensions S in the locality of the test example defined by T. The dominant class label in $\mathcal{Q}(S,T)$ can be used in order to define the class of test example T. We note that the inverted data structure can also be used for conventional classification. In general, a variety of approaches can be used in order to find a set of dimensions in S, such that the gini-index of $\mathcal{Q}(S,T)$ is as large as possible. Some examples of such strategies are sampling or greedy construction of the dimension set S. Similarly, the data structure can also be used for inverse classification by finding those local subspaces which are consistent with the specified values of the test instance, and which classify to the desired class label.

2.2 Leveraging the Inverted Statistics In this section, we will discuss how the inverted statistics are leveraged in order to construct the missing attributes for the test instance. The key is to identify combinations of dimensions from the missing attribute statistics which are biased towards the desired class variable. Since the inverted representation of the training data (along with class statistics) is already available, we will use it in order to achieve our goal.

Let us assume that the test instance T contains q missing attributes which are indexed by $i_1 \dots i_q$. We note that the attributes can take on $v(i_1) \dots v(i_q)$ possible values. We denote the number of possible combinations of filling the missing variables by $C(i_1 \dots i_q)$. This is given by:

$$C(i_1 \dots i_q) = \pi_{j=1}^{q} v(i_j) \tag{2.6}$$

Thus, the number of possibilities for picking the missing attributes implicitly defines the search space for a given test instance. The size of this space is exponentially related to the number of missing attributes. It is our goal to leverage the inverted representation of the training data to efficiently search through the space of exponential possibilities in order to optimally pick the missing attributes. In order to achieve this goal, we will use a roll-up approach. For this purpose, we use the inverted lists from the training data in order to track the *candidate missing value possibilities* for the test instance. The aim is to keep a set of candidate missing value possibilities which classify to the desired class label. In addition, we would like the corresponding subspaces to have a sufficient number of points in them (support requirement), and also be discriminatory among the different classes.

```
Algorithm InvertedClassify(Test Example: $T$,
  Target Class Index: $qc$, Gini Threshold: $a$,
  Support: $s$);
begin
  Compute all possible singleton sets $L_1$ for
     test example $T$ with support at least $s$ and
     gini-index at least $a$;
  Prune $L_1$ to contain only dimension sets in
     which the dominant class is the target
     class $qc$;
  Return any sets to $L_1$ which correspond
  to non-missing variables;
  $k = 1$;
  while $L_k$ contains at least 1 element;
     begin
     Join $L_k$ and $L_1$ to form $C_{k+1}$;
     Prune all elements in $C_{k+1}$ with gini less
       than $a$ and support less than $s$ to
       form $L_{k+1}$;
     Prune all elements in $C_{k+1}$ in which
     the dominant class is different from target
     class $qc$;
     $L_{k+1} = C_{k+1}$;
     $k = k + 1$;
     end
  $\mathcal{LF} = \cup_{i=1}^{k} L_i$;
  while missing values of $T$ have not been filled
       and $\mathcal{LF}$ is non-empty
     begin
     Pick highest gini-index set $H$ from $\mathcal{LF}$
       and add to missing values in $T$ using the
     corresponding values in $H$;
     Remove any sets from $\mathcal{LF}$ which are
        inconsistent with filled values of $T$;
     end;
  if $\mathcal{LF}$ is empty
  fill remaining missing values of $T$ with average
    values of all records taking on the target
    class $qc$;
end
```

Figure 1: The Inverse Classification Algorithm

The inverse classification algorithm uses as input the inverted lists from the training phase, the test example T, the target class index qc, 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. The first step is to construct the set of all singleton inverted lists L_1. We note that L_1 is basically all the inverted list possibilities (from the training data) for the missing variables which have sufficiently high gini index and support, and also classify to the desired class label qc as the dominant class for that inverted list. In addition, the list L_1 also contains one element for each of the non-missing elements. Since, there are $v(i_j)$ possibilities for the missing variable i_j, the total number of possibilities is given by:

$$N(i_1 \ldots i_q) = \sum_{j=1}^{q} v(i_j) + (d - q) \tag{2.7}$$

The inverse classification algorithm prunes many of these singletons because they do not satisfy the gini index and support threshold requirements. In addition, we prune any of the singletons which do not classify to the desired class, and whose attribute values have not already been specified in T. We do not prune the inverted lists corresponding to specified values in T, because they have already been decided and the effects on other attributes needs to be considered, even if they do not satisfy the gini-index, support or class requirements. Once the singleton set L_1 has been generated, we use pairwise joins on the elements in L_1 in order to generate the set C_2. Each pairwise join essentially consists of the process of intersecting two inverted lists. We note that the pairwise joins need to be performed only on those elements in L_1 which belong to different dimensions. This is because the process of performing the intersection on two lists (for disjoint ranges) in the same dimensions results in the empty set. In the next step, we prune those elements from C_2 which do not have the desired level of support, gini- value, or for which the dominant class label is not the desired target class. At this point, we set the value of L_2 to C_2.

This process is repeated in roll-up fashion for L_k and C_k for general values of k. In general, we construct C_{k+1} by performing a join of L_k with L_1. As in the previous case, we prune those elements of C_{k+1} which do not have the desired support, gini-index, or for which the dominant class is different from the target class. The set L_{k+1} is then set to the pruned set C_{k+1}. We note that for larger values of k, fewer and fewer patterns will satisfy the support requirements, and eventually, the set L_{k+1} will be empty. The process of constructing C_{k+1} from L_k and L_1 is repeated iteratively, until the set C_{k+1} is empty. At this point, the set of elements in $\mathcal{LF} = \cup_{i=1}^{k} L_i$ form the set of different possibilities for the values which may be chosen for the missing values in the test example T. We note that different combinations of possibilities may be inconsistent with one another since they may suggest different values for the missing variables.

Next, we will discuss how $\mathcal{LF}$ may be used in order to iteratively construct the missing values in the test example T. In this process, we repeatedly pick the set with the highest gini-index from $\mathcal{LF}$. We substitute these values in T with these picked values from $\mathcal{LF}$. In each iteration, we also delete those sets in $\mathcal{LF}$ which are *inconsistent* with the filled values of T. A set in $\mathcal{LF}$ is said to be inconsistent with T if the value of any of its attributes is inconsistent with a filled value of T. In the next iteration, 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, and the set $\mathcal{LF}$ is empty, we need to pick the unfilled values of T. This is a rare circumstance, since $\mathcal{LF}$ will usually contain some singleton from L_1 which can be used to fill T. Nevertheless, the situation is a possibility when the gini or support threshold is chosen to be too high. In such cases, we choose the value of that feature variable using 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. This is however a rare situation for a rudimentary special case, and does not affect the overall behavior of the algorithm. The inverse classification algorithm is illustrated in Figure 1.

3 Experimental Results

All 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 note that these data sets are also typically not decision support problems; however, by removing the attribute values from real data sets, it is possible to know the effectiveness of the inverse classification algorithm in learning the relationships in the attribute values which optimize the classification accuracy. As long as the data sets used for testing correspond to real domains, the results are not reflective of a pre-defined or artificial distribution. In such cases, it can be assumed the quality of the results will provide guidance in understanding how well the technique will work in real domains. We will describe our results on a number of data sets which were obtained from the UCI machine learning repository [8]. Each 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 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 inverted classification 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 inverted classification algorithm for determining the entries in the data. Specifically, we used the following classifiers which were obtained from the open-source WEKA algorithms repository [16]:

- An implementation of a Naive Bayesian classifier.
- An implementation of a decision tree classifier which is referred to as REPTree.
- The implementation [16] of the decision table classifier discussed in [11].

The accuracy of the classifiers on the newly constructed data set were used to test the effectiveness of the technique. The aim is to show that the use of the inverted classification method results in an increase in accuracy of classification on the substituted attributes. This shows that by changing the nature of the decision variables, it is possible to improve the accuracy of decision based classification processes. We further note that the Naive Bayesian, REPTree and the Decision table classifiers are conceptually very different from the lazy learners used in the inverse classification algorithm in order to decide the unfilled attributes. Therefore, such classifiers can be considered "righteous" judges which do not overfit to the particular methodology of the inverted classification algorithm. Furthermore, if the accuracy across the different classifiers improved with the use of different kinds of data sets, then this is evidence of the effectiveness of the inverted classification method in defining correct values of the corresponding decision variables.

	mush.	car	tic-tac.	nursery
Dim.	22	6	9	8
Card.	5.68	4.5	3	4.375
Records	8124	1728	958	12960
Classes	2	4	2	5

Table 1: Dataset Characteristics

3.1 Descriptions of data We used the following four data sets from the UCI machine learning repository: mushroom, car, tic-tac-toe, and nursery. The data sets were chosen in order to reflect a variety of dimensionalities, data set sizes, and class distributions. The corresponding characteristics are listed in Table 1. In this Table, the term *cardinality* refers to the average number of values per attribute. All datasets have nominal attributes which can be used in a natural way for testing the effectiveness of the inverted classification approach. In all cases, the target class label for the test data was set to its same value as in the unmodified data. We note from Table 1, that many of these problems are multi-class problems. Therefore, the algorithm was required to simultaneously learn the correct feature substitution for different target labels for different test instances. We note that the different distributions of feature characteristics of different class labels could affect the nature of the best strategy which should be used for the inverse classification problem. This ensures that the accuracy boosting results are quite robust since they are not tailored to a particular kind of distribution or relationship between the features and class variables.

3.2 Accuracy Results In this section, we will discuss the accuracy results from the use of the inverted classification approach. In each case, we have illustrated 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 of the data sets, and settings (T_{before} and T_{after}), we computed the accuracy of the classification process with increasing number of removed attributes. The results are illustrated in the following figures:

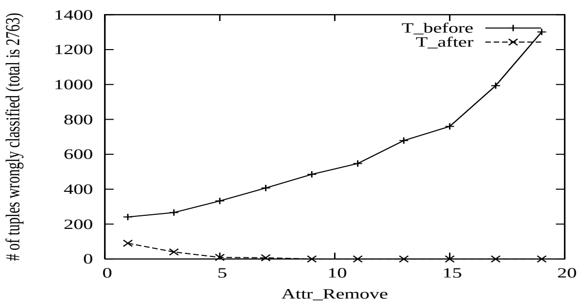

Figure 2: Naive Bayesian: Accuracy boost on mushroom with increasing number of removed attributes

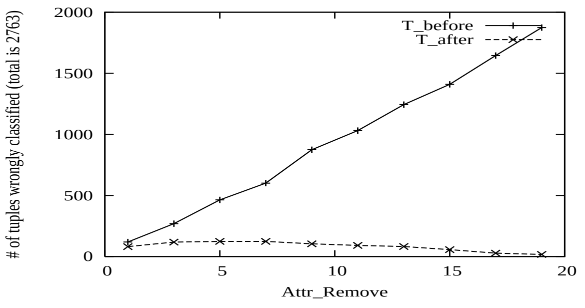

Figure 3: REPTree: Accuracy boost on mushroom with increasing number of removed attributes

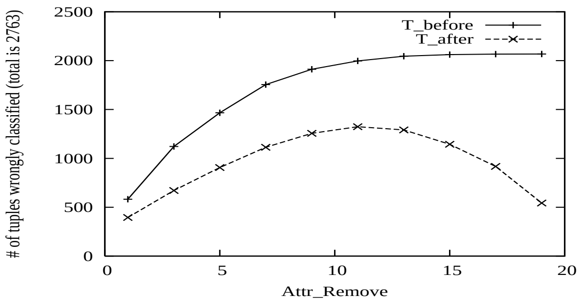

Figure 4: Decision Table: Accuracy boost on mushroom with increasing number of removed attributes

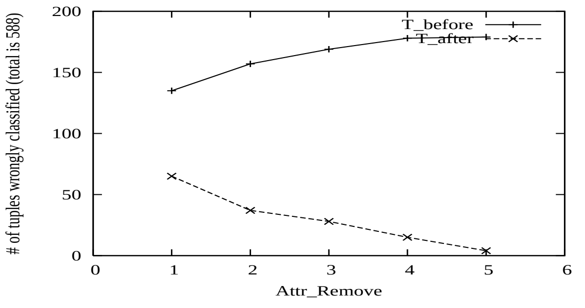

Figure 5: Naive Bayesian: Accuracy boost on car with increasing number of removed attributes

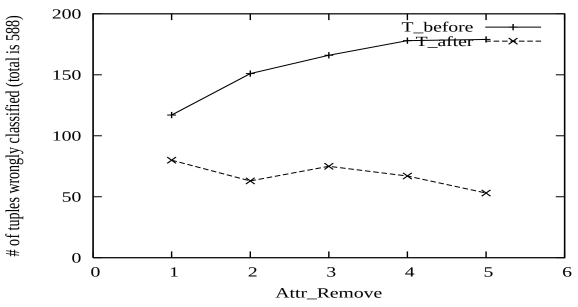

Figure 6: REPTree: Accuracy boost on car with increasing number of removed attributes

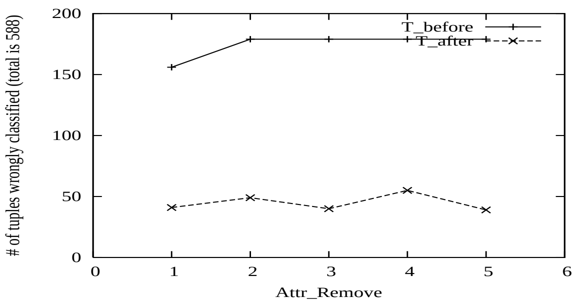

Figure 7: Decision Table: Accuracy boost on car with increasing number of removed attributes

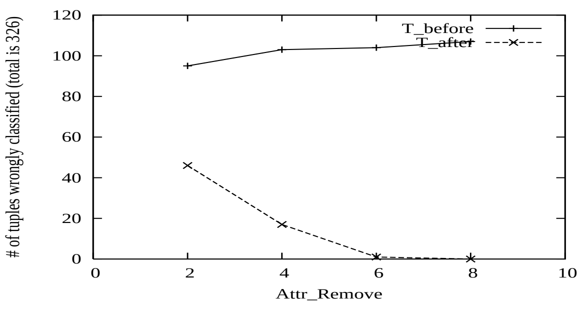

Figure 8: Naive Bayesian: Accuracy boost on tic-tac-toe with increasing number of removed attributes

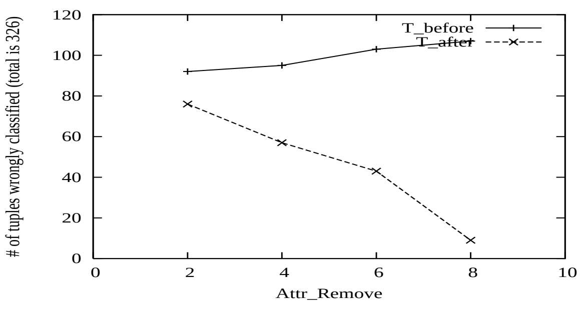

Figure 9: REPTree: Accuracy boost on tic-tac-toe with increasing number of removed attributes

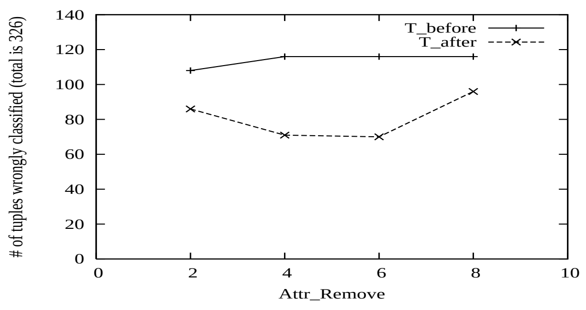

Figure 10: Decision Table: Accuracy boost on tic-tac-toe with increasing number of removed attributes

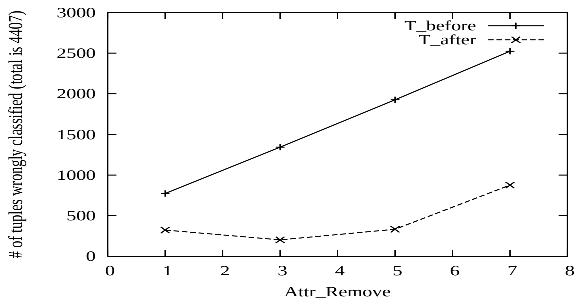

Figure 11: Naive Bayesian: Accuracy boost on nursery with increasing number of removed attributes

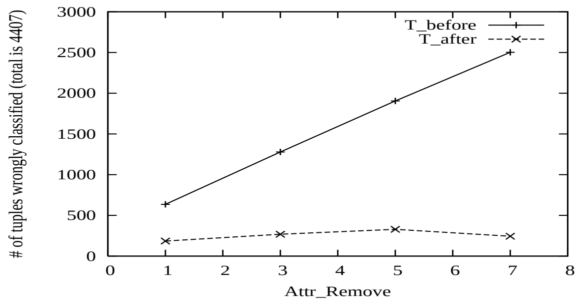

Figure 12: REPTree: Accuracy boost on nursery with increasing number of removed attributes

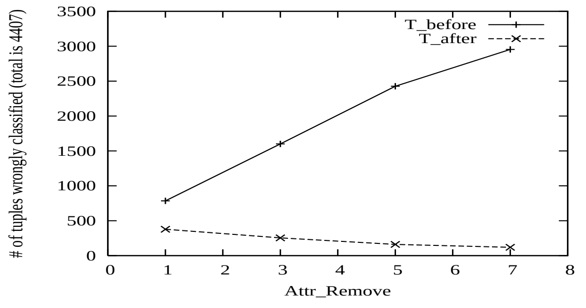

Figure 13: Decision Table: Accuracy boost on nursery with increasing number of removed attributes

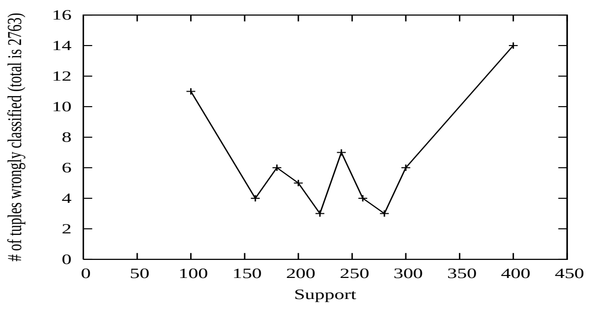

Figure 14: Sensitivity for mushroom: Accuracy with Support

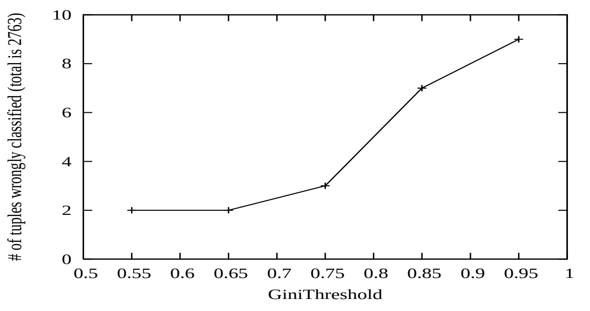

Figure 15: Sensitivity for mushroom: Accuracy with GiniThreshold

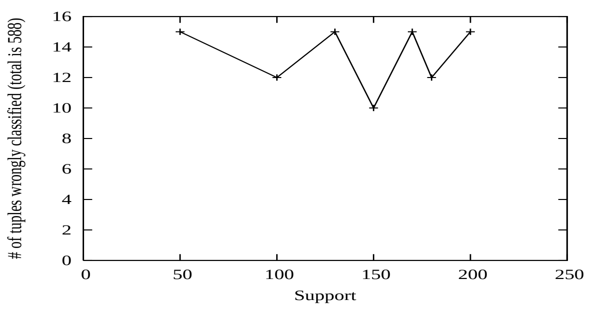

Figure 16: Sensitivity for car: Accuracy with Support

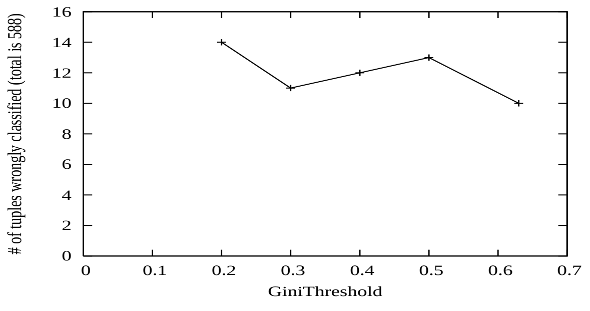

Figure 17: Sensitivity for car: Accuracy with GiniThreshold

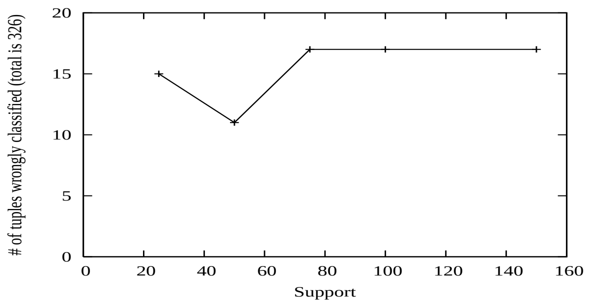

Figure 18: Sensitivity for tic-tac-toe: Accuracy with Support

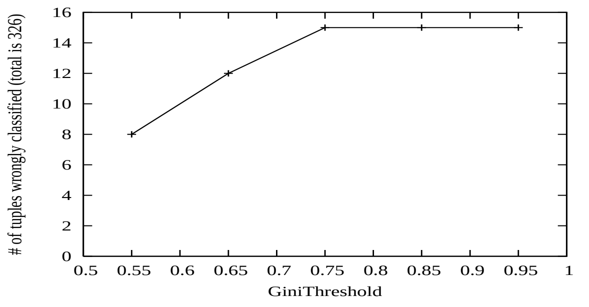

Figure 19: Sensitivity for tic-tac-toe: Accuracy with GiniThreshold

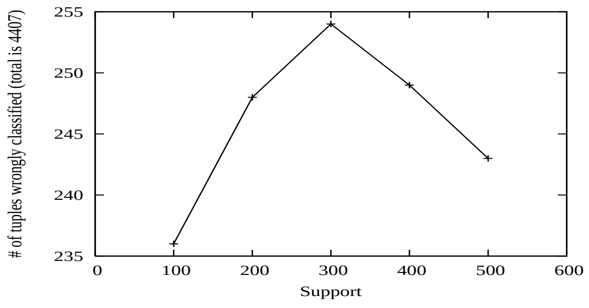

Figure 20: Sensitivity for nursery: Accuracy with Support

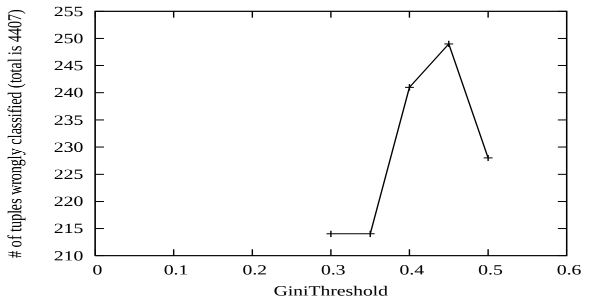

Figure 21: Sensitivity for nursery: Accuracy with GiniThreshold

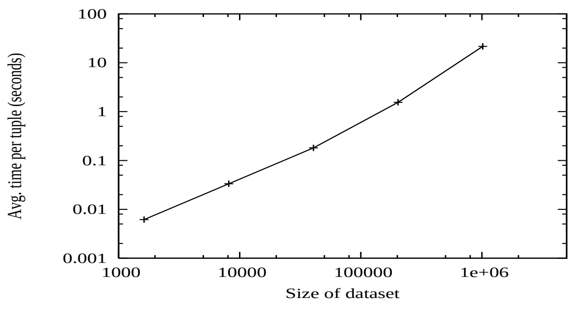

Figure 22: Time complexity on mushroom w.r.t. data size

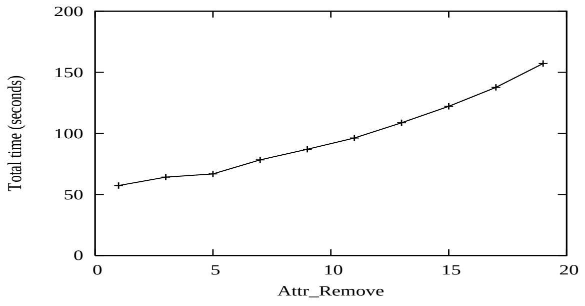

Figure 23: Time complexity on mushroom w.r.t. Attr_Remove

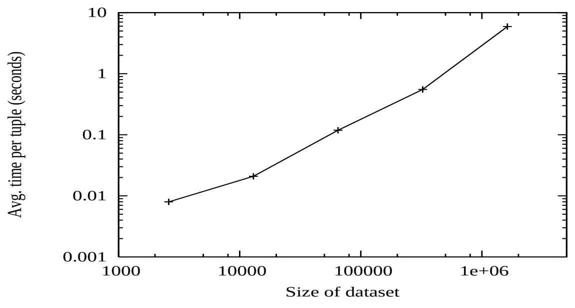

Figure 24: Time complexity on nursery w.r.t. data size

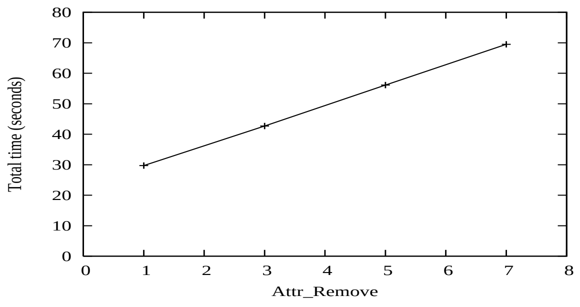

Figure 25: Time complexity on nursery w.r.t. Attr_Remove

- The mushroom data set results are illustrated in Figures 2, 3, and 4 for the Bayes classifier, decision tree, and decision table classifiers, respectively.
- The results for the car data set are illustrated in Figures 5, 6, and 7 respectively.
- The results for the tic-tac-toe data set are illustrated in Figures 8, 9, and 10 respectively.
- The results for the nursery data set are illustrated in Figures 11, 12, and 13 respectively.

In most cases, 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 a small number of cases, the accuracy may actually increase slightly when the number of missing attributes are increased. For example, the decision table classifier shows this behavior on the mushroom and tic-tac-toe data sets. This can be observed from Figures 4 and 10. A similar behavior is exhibited in Figure 11 by the naive bayesian classifier on the nursery data set. These anomalous behaviors are a result of some instances in which the removal of some of the attributes had the indirect effect of feature selection for that particular classifier. It is also interesting to see, that this kind of behavior was not just data set specific but was also classifier specific.

At the same time, we should note that the overall trend should only be judged on the basis of the behavior of all three classifiers. It is easy to see that the broad trend supports a reduction in accuracy when the number of missing attributes are increased. In each figure, we have labeled the curve illustrating the variation in accuracy with increasing number of attributes by T_{after}. We make the following observations over the different data sets:

(1) 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.
(2) The amount of boosting depended both upon the combination of the classifier and data set being used. For example, for the nursery data set, the REPTree classifier seemed to provide better boosting, whereas for the other data sets, the Naive Bayes classifier seemed to provide better boosting results. This shows that even though different data sets may be more amenable to different kinds of classifiers, the overall trend of accuracy boosting as a result of feature modification remains unchanged. This also tends to indicate that the boosting effects are not a result of overfitting to a particular kind of classifier, but they are a result of the effective decision feature combinations created by the inverse classification algorithm.

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.

3.3 Parameter Sensitivity and Tuning Process

In this subsection, we will test the sensitivity of the inverted classification method to parameters such as the optimal setting of *Support* and *GiniThreshold*. In Figure 14, we fix the value of *GiniThreshold* at 0.75 and explore how the accuracy varies according to *Support*. From the results, it seems that the optimum values of the *Support* should be in the range of 160 to 300. However, the key observation is that the accuracy variations across different values of the parameters are not very large. In Figure 15, we have illustrated the variation of the accuracy with the gini index after fixing the support at 220. From the results of Figure 15, it is clear that the accuracy stabilizes from 0.55 to 0.65. We note that the gini index value of 0.65 is only slightly larger than the gini index of the random class distribution from the data set. This was a general trend which was observed over different data sets.

We have illustrated the variations in the accuracy with the gini-index and support for the other data sets in Figures 14, 15, 16, 17, 18, 19, 20, and 21 respectively. From the graphs, we can see that the inverted classification method is not very sensitive to parameter variations. This is good news from an application point of view, since it means that it is possible to apply the method effectively over a wide range of parameter values.

The general observations from the different data sets also provided some understanding of the behavior of the inverted classification algorithm which can guide the parameter setting procedure:

- In general the threshold gini-index was about 10% to 20% times larger than that computed according to the training set class distribution. In general, the default gini-index from the training set distribution served as an effective default value which was used to compute the accuracy graphs.

- The support threshold could be defined well by the average number of data points per nominal attribute value. This is because this choice of parameters ensures the joins of the inverted lists not to explode in terms of the computation of the discriminative patterns. The accuracy for this range also continued to be quite robust.

3.4 Scalability tests For scalability tests, we computed the running time with increasing training data size. We artificially enlarge the mushroom data set by "repeating" the data set several times in a manner that random noise is added to some of the attributes. By using this procedure, the resulting data set is not just a duplicated version of the original data set, but an enlarged version of the data set with a similar distribution. For the enlarged data set we set a *Support* which is proportional to the data set size, whereas the $GiniThreshold$ is held constant. The results are illustrated in Figure 22. The results illustrate a linearly scalable scenario. This is an encouraging result, since it indicates scalability of the method for very large data sets. Similar results were obtained for the nursery data set, which are illustrated in Figure 24.

We also tested the scalability of the method with increasing number of missing attributes. The results for the mushroom and nursery data sets are illustrated in Figures 23 and 25 respectively. While the running times increase faster than linearly, this increase is not significantly faster than linear. The reason for the super-linear increase in time complexity is that the inverted classification algorithm needs to test various combinations of attributes in order to find the optimum values of the decision attributes. In practice, the use of inverted lists significantly speeds up the search process. In higher dimensionality the size of each inverted list is also small, and the intersections are much faster. Therefore, the running times scale effectively with number of missing attributes.

4 Conclusions

In this paper, we introduce the inverted classification method, a technique for determining of decision attributes in order to achieve a desired result. This inverse relationship is very useful in a number of applications in which we are trying to achieve a desired result, rather than trying to determine the classification behavior based on pre-defined features. Such an approach is more useful for *action-driven* applications in which we are trying to find the appropriate values of the decision variables in order to optimize particular business applications. Such methods can bridge the divide between *analysis-driven* data mining and *action-oriented* data mining. While the latter has not been studied as extensively as the former, it is often more useful in many real scenarios. We tested the inverted classification algorithm on a wide variety of data sets, and show the effectiveness of the method in boosting the classification accuracy towards a desired result. In general, this effectiveness continues to be observed over a wide range of parameter variations. We also illustrated the scalability of the method and show that it can be used efficiently for large data sets or a large number of decision variables.

References

[1] K. Alsabti, S. Ranka, V. Singh. CLOUDS: A Decision Tree Classifier for Large Datasets. *KDD Conference*, 1998.

[2] L. Breiman, J. Friedman. R. A. Olshen, C. J. Stone. Classification and Regression Trees. Wadsworth, Belmont, 1984.

[3] C. E. Brodley, P. E. Utgoff. Multivariate Decision Trees. *Machine Learning*, Vol 20, pp. 63-94, 1995. *University of Mass. Tech. Report*, UM-CS-1992-083, 1992.

[4] L. Breslow. D. Aha. Simplifying Decision Trees. *Knowledge Engineering Review*, Vol. 12, No. 1, pages 1-40, 1997.

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

[6] J. H. Friedman. A recursive partitioning decision rule for non-parametric classifiers. *IEEE Transactions on Computers*, C-26, pp. 404-408, 1977.

[7] J. Gehrke, V. Ganti, R. Ramakrishnan, W.-Y. Loh. BOAT: Optimistic Decision Tree Construction. *ACM SIGMOD Conference Proceedings*, pp. 169-180, 1999.

[8] S. Hettich, C. Blake, C. Merz. UCI Repository of Machine Learning Databases, *University of California, Irvine, Department of Information and Computer Science*, 1998.
URL: www.ics.uci.edu/˜mlearn

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

[10] L. Kaelbling, M. Littman, A. Moore. Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research*, Vol. 4, pp. 237-285, 1996.

[11] R. Kohavi. The Power of Decision Tables. In *Proc. European Conference on Machine Learning*, 1995.

[12] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[13] R. Rastogi, K. Shim. PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning. *VLDB Conference*, 1998.

[14] P. Smyth, A. Gray, U. M. Fayyad. Retrofitting Decision Tree Classifiers Using Kernel Density Estimation. *Proceedings of the International Conference on Machine Learning*, 1995.

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

[16] I. Witten, E. Frank. Data Mining: Practical machine learning tools with Java implementations. *Morgan Kaufmann Publishers*, San Francisco, CA, 2000.
URL: http://www.cs.waikato.ac.nz/˜ml/weka/book.html