9

Enhancing Greedy Policy Techniques for Complex Cost-Sensitive Problems

Camelia Vidrighin Bratu and Rodica Potolea
Technical University of Cluj-Napoca
Romania

1. Introduction

One of the most prominent domains of application for machine learning techniques is data mining, which focuses on the discovery of novel, structured and potentially useful patterns in data. Each machine learning algorithm makes assumptions regarding the underlying problems it needs to solve. These assumptions and the search strategy employed constitute the bias of the learning algorithm. This bias restricts the area of successful application of the algorithm. Also, recent research in machine learning has revealed the necessity to consider more complex evaluation measures for learning algorithms, in some problem domains. One such novel measure is the cost. There are various types of costs involved in inductive concept learning, but, for the domains we focus on, the most important are test costs and misclassification costs.

This chapter presents ProICET (Vidrighin et al, 2007), a hybrid system for solving complex cost-sensitive problems. We focus both on the theoretical principles of the approach, as well as highlight some implementation aspects and comparative evaluations on benchmark data (using other prominent learning algorithms).

The remainder of the chapter is structured as follows: section 2 reviews the search problem, and a few fundamental strategies, and provides basic definitions for data mining and decision trees. Section 3 presents the cost-sensitive problem, together with a brief survey of the most prominent cost-sensitive learners present in literature. Section 4 presents the theory behind ProICET, followed by the enhancements considered and an overview of the implementation. Section 5 presents the experimental work performed in order to validate the approach. The chapter summary is presented in the final section.

2. Basic techniques for search and knowledge extraction

Search is a universal problem-solving mechanism is various domains, including daily life. It also represents one of the main applications in computer science in general, and in artificial intelligence in particular (Korf, 1999). The problem can be formalized as finding a path (in the state space) from the root node to the goal node. In many cases, a particular path is requested, namely the one which obeys some optimization criterion. The main task here comes from the difficulty of finding the right search strategy for the particular problem to solve. In evaluating them several criteria are considered, such as completeness (the ability of the strategy to find the solution in case there is one), time/space complexity (the amount of

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

time/memory the search strategy needs to find the solution), optimality (the ability of the strategy to find the best solution, according to the optimization criterion).
The next section presents two important search strategies, with their derivatives. Moreover, their performance criteria are discussed and compared. An *uninformed search strategy* (sometimes called blind search) performs in the absence of knowledge about the number of steps or the path cost from the current state to the goal. The most prominent approaches in this category are breadth first search and depth first search. As opposed to uninformed methods, the *informed search strategy* employs problem-specific knowledge. The best first search strategy from this category is reviewed, and one of its simplest, yet effective versions, greedy search. The common pattern in all strategies is the expansion of the current node (i.e. considering its successors as candidates for finding the path to goal), while the particularity consists in the order in which the neighbors are evaluated for expansion.

2.1 Fundamental search strategies

In the *breadth first search* strategy the root node is expanded first. In the second step, all nodes generated by it are expanded, in the third step, their successors, and so on. This means that at every step the expansion process occurs for nodes which are at the same distance from the root, and every expanded node in a step is on the boundary of the covered/uncovered region of the search space. Breadth first search considers a systematic approach, by exhaustively searching the entire state space without considering the goal until it finds it. Due to the fact that the whole space is covered, the strategy is *complete* (i.e. on a finite space, the solution is found, in case there is one). Moreover, the strategy is *optimal*. The drawback is the large complexity, both in time and space: $O(b^d)$, where b represents the branching factor (i.e. number of descendents of a node) and d the depth of the space. Breadth first search can be implemented using a general search strategy with a FIFO queue for the states (Russell & Norvig, 1995).
Uniform cost search comes as a flavor of breadth first search. Assuming a cost function g(n) is considered, breadth first search is modified by expanding the lowest cost node (min g(n)) on the boundary. The default distance to the root, used by the breadth first search is replaced by some specific cost function g(n) (i.e. for breadth first search, g(n)=depth(n) by default). Thus, the systematic approach of covering the space is relaxed to reach the optimal solution faster. Dijkstra's algorithm is a uniform cost search algorithm.
The *depth first search strategy* has a similar approach, but instead of expanding nodes on the boundary, it always expands one node at the deepest level. In case the search reaches a dead end, where no expansion is possible, a node on a shallower level is considered. This way, the "horizontal" approach of covering the states space is replaced by a "vertical" one. Depth first search can be implemented by a general search strategy if a stack is used to keep the states. That is, the FIFO policy is replaced by a LIFO. The major advantage of this strategy is reduced space requirement: $O(bm)$, where m is the maximum depth. The time complexity remains in the exponential domain: $O(b^m)$. The drawback is that the method is *neither complete, nor optimal*. This is the reason why it should be avoided for spaces with large or infinite max depths.
By imposing an upper limit to the maximum depth of a path these pitfalls can be avoided. This modified strategy is implemented by *depth-limited search*. In this situation, the strategy becomes complete, if the depth of the solution is smaller than the threshold imposed, yet it is still not optimal.

The *bidirectional search* simultaneously searches both from the root (forward) and the goal (backward) and stops when the two meet in the middle. It has the advantage of being *optimal* and *complete* at the same time. Moreover, it reduces the time complexity to $O(b^{d/2})$, with the cost of increasing the space complexity to $O(b^{d/2})$. The disadvantage is that the backward search from the goal is not applicable to all problems. This requires expanding the predecessor node, rather than the successor. When the operators are reversible, computing the predecessor is not an issue. However, for some problems, calculating predecessors is very difficult. Moreover, the goal state may not be unique, meaning that the backward strategy should be started from several nodes. Finally, there is no unique way to perform the search in the two halves: the strategy is strongly dependent on the problem. Other issues, such as checking the appearance of a node on the other half, have to be considered for the bidirectional approach as well.

If some knowledge is added to the queuing function, which determines the node to expand next, the chances to find (completeness) the optimal (optimality) solution faster (time complexity) increases and we deal with an informed search strategy. The knowledge usually refers to some performance function, as a measure of the desirability of expanding a node.

Best first search strategy expands the node for which the performance function is estimated to be the best. Emphasis on estimation is important: expansion applies to the most promising node, rather than to be the one which surely leads to the best solution. Thus, the strategy doesn't necessarily deliver the optimal solution (but the one which appears to be the best according to the performance criterion). If the evaluation was precise, and we could expand the best node, it would not be a search strategy at all, but a straight path from the root to the goal. Selecting the best candidate for expansion is done using a priority queue.

Greedy (best first) search is one of the simplest strategies in this category. The knowledge added here is the estimated cost of the cheapest path from the current node to the goal. As mentioned for the generic case, this cost cannot be determined exactly; the function that estimates the cost is called a *heuristic function*, *h(n)*. The greedy strategy takes the best local decision, with no evaluation of further effort. The method resembles depth first search, as it follows a single path in the attempt to reach the goal, backing up in case a dead end is found. Because of the similarities, it has the same drawbacks with depth first search: it is *not optimal*, and *incomplete*. The time complexity is still exponential: $O(b^m)$. Even worse, due to the fact that the strategy memorizes all nodes, the space complexity is also $O(b^m)$. Although the optimality of the solution is not guaranteed, it usually finds a good solution (close to the optimal). Moreover, if some problem-specific knowledge is added, it can obtain the optimal solution. Both Prim's and Kruskal's algorithms for finding the minimum spanning tree are greedy search algorithms. Because it minimizes the estimated cost to the goal, h(n), greedy search decreases the search costs as well, by cutting search branches. This makes the strategy efficient, although not optimal.

One trap greedy strategy falls in is estimating the performance function from the current node to the goal, without taking into account the component of the function from the root to the current node (which can actually be calculated exactly, not just estimated). This cost is the selection criterion for uniform cost search ($g(n)$). Thus, the choice can be based on a fusion of the two criteria. That is, the summation of h and g is considered as performance function: $f(n)=g(n)+h(n)$. Such a strategy (similar to the branch and bound technique), of minimizing the total path cost defines the *A* search*. $f(n)$ represents the estimated cost on the cheapest path from the start node to the goal, and it incorporates $g(n)$ as an exact measure of

the path from the start node to the current node (as for *uniform cost search*), and *h(n)* as the estimation of the remainder path to the goal (as for *greedy search*). By finding a restriction that never overestimates the cost to reach the goal for *h*, the method is both *complete and optimal* (Russell & Norvig, 1995). Such a restriction is an *admissible heuristic*, which is optimistic by nature, by always underestimating the cost of solving the problem. Since *h* is admissible, the effect transfers to *f* as well (since *f*=*g*+*h*), and it underestimates the actual cost as well. *A* search* is a best first search using *f* as the evaluation function and an admissible *h* function.

2.2 Enhanced search strategies

Iterative improvement techniques are efficient practical approaches for boosting search strategies. They can be divided into two classes. The *hill-climbing strategy* makes changes to improve the current state. The algorithm does not maintain a search tree. Rather, it moves in the direction of increasing value within a loop. Although simple by nature, and efficient in practice, it suffers the drawback of becoming trapped in local optima. *Simulated annealing* represents the other class of iterative improvement strategies. It simply allows escaping a local optimum, by taking some steps to break out. It is an effective strategy for a good approximation of the global optimum in a large search space.

In *combinatorial search,* the goal is to find the best possible solution out of the feasible ones. There are two main approaches here. In *lazy evaluation* the computation is delayed until it is really needed, in contrast to *look-ahead* where, before making a decision, a few input steps are evaluated, in order to avoid backtracking at later stages. Both methods try to save both time and space in their evaluation.

Another distinctive technique is employed by *genetic algorithms*, which are essentially stochastic search methods, inspired from the principles of natural selection in biology. They employ a population of competing solutions—evolved over time—to converge to an optimal solution. Effectively, the solution space is searched in parallel, which helps in avoiding local optima, and provides straightforward parallelization possibilities. The search is an iterative process where each successive generation undergoes selection in the presence of variation-inducing operators such as mutation and recombination (crossover). A fitness function is used to evaluate individuals, and reproductive success varies with fitness.

Straightforward parallelization and the possibility of applying them in ill-defined problems make genetic algorithms attractive.

2.3 Data mining

Traditionally, *data mining* refers to the activity of extracting new, meaningful and potentially useful information from data. The term has recently expanded to the entire knowledge discovery process, encompassing several pre-/post- and processing steps. The learning step is central in any data mining process. It consists of presenting a dataset – the *training set* – to a learning algorithm, so that it learns the model "hidden" in the data. A dataset consists of a set of *instances*, each instance having a set of *predictive attributes* and a *target attribute*, the class. The aim is to predict the value of the class using the values of the predictive attributes and the model learned by the induction algorithm. In order to assess the generalization ability of the learned model (i.e. its quality), usually a *test set* is employed, consisting of instances that have not been "seen" by the model during the learning phase. Such a problem is known as a classification problem (if the class attribute is discrete), or a regression

problem (if the class is continuous). Another data mining task is clustering, which identifies similar characteristics and groups cases with similar characteristics together. In this case the class attribute is not present.

2.4 Decision trees

One of the most prominent techniques used for classification (and regression) problems in data mining are *decision trees*. They are tree structures, where each interior node corresponds to a decision attribute; an arc from a node to a child represents a possible value of that attribute. A leaf represents the value of the class attribute, given the values of the attributes present on the path from the root to that leaf. Decision tree algorithms apply a greedy search heuristic and construct the model in a top-down, recursive manner ("divide and conquer"). At every step, the algorithm considers the partition of the training set with respect to the "best" attribute (which becomes the decision attribute for that node). The selection of the "best" attribute is made according to some splitting measure. After an appropriate split has been selected, the training set is divided among the branches going out of that node into smaller subsets. The process continues until no split is considered good enough or a stopping criterion is satisfied.

The decision on which attribute to choose at a given step is based on measures provided by the information theory, namely on the entropy. It measures the uncertainty associated with a random variable. The most common attribute selection criterion is the expected reduction in entropy due to splitting on that attribute – the information gain.

While being rather simple and easy to understand, decision trees are also very robust with respect to the data quantity. Also, they require little data preparation, being able to handle both numerical and categorical data, as well as missing data. Furthermore, it is possible to validate the model using statistical tests, such as to determine its reliability.

3. Cost-sensitive learning

Traditionally, learning techniques are concerned with error minimization, i.e. reducing the number of misclassifications. However, in many real-world problems, such as fraud detection, loan assessment, oil-slick detection or medical diagnosis, the gravity of different types of classification errors is highly unbalanced.

For example, in credit assessment, given a customer loan application, the goal is to predict whether the bank should approve the loan, or not. In this situation, false positives are much more dangerous than false negatives. This means that an incorrect prediction that the credit should be approved, when the debtor is not actually capable of sustaining it, is far more damaging than the reverse situation. Another domain where different errors bear different significance and consequences is medical diagnosis (classifying an ill patient as healthy is by far riskier than the reverse situation).

In domains like these, the measure of total *cost* is introduced to determine the performance of learning algorithms. Total cost minimization is at least as important as minimizing the number of misclassification errors. This strategy is employed by cost-sensitive learning, a category of learning schemes which consider different approaches to achieve minimal costs.

As presented in (Turney, 2000), there are several types of costs involved in inductive concept learning, the most important being the *misclassification costs* and the *test costs*. These are also the focus of most cost-sensitive algorithms. Misclassification costs try to capture the

unbalance in different misclassifications. They are modeled through the use of a cost matrix $(C_{ij})_{nxn}$, where C_{ij} is the cost of misclassifying an instance of class j as being of class i. Test costs quantify the "price" of an attribute, without being restricted to its economical value. For example, in the medical domain, a test cost could represent a combination between the costs of the equipments involved in the investigation, the time spent to gather the results, the impact on the patient (psychical or physical - pain), a.s.o. Test costs are specified as attribute – value pairs.
In most real-world problems, setting the true costs is a difficult issue. If, in the case of test costs, the decision is made easier by the possibility of considering the different dimensions (time, monetary, pain, emotional implications, e.t.c.), when it comes to determining the misclassification costs we come across a more serious issue: we have to put a price on human life. Perhaps an appropriate approach here would be to experiment with several close proportions for the errors' unbalance.

3.1 Cost-sensitive algorithms

Most cost-sensitive classifiers focus on minimizing the misclassification costs. There exist, however, several algorithms which tackle test costs. Significantly less work has been done in aggregating the two cost components. This section reviews some of the most prominent cost-sensitive approaches in literature: stratification, MetaCost (Domingos, 1999) and AdaCost (Fan et. al., 2000) as misclassification cost-sensitive approaches, and Eg2 (Nunez, 1988), IDX (Norton, 1989) and CS-ID3 (Tan & Schlimmer, 1989, 1990) which consider test costs.

3.1.1 Stratification

Stratification is one of the earliest and simplest techniques for minimizing misclassification costs. It is a sampling procedure, which modifies the distribution of instances in the training set, such that the classes with a higher misclassification cost are better represented. Stratification can be achieved either through undersampling, or oversampling. While being a very simple and intuitive technique for considering the unbalance of different types of errors, the modification of the set distribution induces drawbacks, since it may bias the learning process towards distorted models. Also, each alternative has its own drawbacks: undersampling reduces the data available for learning, while oversampling increases the training time. However, the most serious limitation of this method comes from the fact that it restricts the dimension or the form of the cost matrix. For problems with more than two classes, or when the cost is dependent on the predicted class ($C_{ij} \neq C_{kj}$, where $k \neq i$), the cost matrix may become too complicated, such that proportions for each class cannot be established (Domingos, 1999).

3.1.2 MetaCost and AdaCost

More complex approaches usually involve meta-learning, and can be applied to a variety of base classifiers. The most representative in this category are MetaCost (Domingos, 1999) and AdaCost (Fan et. al., 2000).
MetaCost, introduced by Pedro Domingos, is a method for converting error-based classifiers into cost-sensitive approaches. It employs the *Bayes minimal conditional risk* principle to perform class re-labeling on the training instances. In order to determine the Bayes optimal prediction for each training example, i.e. the class which minimizes the conditional risk, an ensemble of classifiers is initially trained and employed to estimate the class probability for

each instance. After that, the risk for each class is computed, using the cost matrix settings. Each instance is then re-labeled with the class having the lowest risk. After obtaining the modified dataset, any error-based classifier will also minimize the cost while seeking to minimize zero-one loss (the error).
AdaCost is the misclassification cost-sensitive variant of the AdaBoost.M1 algorithm. Being a boosting-based approach, AdaCost employs an ensemble method, which builds a new model at each phase. Weights are assigned to each instance, and they are modified after each boosting phase, using the cost of misclassifications in the weight-update mechanism. Initially, high weights are assigned to costly instances, as opposed to AdaBoost.M1, where uniform weights are assigned to the training instances for the first boosting phase. In the empirical evaluations performed, AdaCost yielded a consistent and significant reduction in misclassification costs over AdaBoost.M1.

3.1.3 Eg2, CS-ID3 and IDX

The category of algorithms which focus on minimizing test costs is largely based on decision trees. Eg2, CS-ID3 or IDX are basically decision trees which employ a modified attribute selection criterion such as to embed the cost of the attribute in the selection decision. *Eg2*'s criterion is detailed in the section regarding the algorithm ICET.
IDX (Norton, 1989) uses a look-ahead strategy, by looking n tests ahead, where n is a parameter that may be set by the user. Its attribute selection criterion is:

$$\frac{\Delta I_i}{C_i} \tag{1}$$

where ΔI_i represents the information gain of attribute i, and C_i is its cost.
CS-ID3 (Tan & Schlimmer, 1989, 1990) uses a lazy evaluation strategy, by only constructing the part of the decision tree that classifies the current case. Its attribute selection heuristic is:

$$\frac{(\Delta I)^2}{C_i} \tag{2}$$

4. ICET – Inexpensive Classification with Expensive Tests

Introduced by Peter D. Turney as a solution to cost-sensitive problems, *ICET* (*Inexpensive Classification with Expensive Tests*) is a hybrid technique, which combines a *greedy search heuristic* (decision tree) with a *genetic algorithm*. Its distinctive feature is that it considers both test and misclassification costs, as opposed to the other cost-sensitive algorithms, which fail to consider both types of costs. Since it models real-world settings, where both the attributes and the different classification errors bear separate prices, the approach is more successful in true-life.
The technique combines two different components, on two levels:

- On the bottom level, a test cost-sensitive decision tree performs a greedy search in the space of decision trees
- On the top level, the evolutionary component performs a genetic search through a space of biases; these are used to control the preference for certain types of decision trees in the bottom layer

The components used in the initial version of ICET are: Eg2 (Nunez, 1988) for the decision tree component and GENESIS (Grefenstette, 1986) for the genetic component. *Eg2* has been implemented as a modified component of Quinlan's C4.5 (Quinlan, 1993), using ICF (Information Cost Function) as attribute selection function. For the ith attribute, ICF may be defined as follows:

$$ICF_i = \frac{2^{\Delta I_i} - 1}{(C_i + 1)^w}, \text{ where } 0 \leq w \leq 1 \quad (3)$$

This means that the attribute selection criterion is no longer based solely on the attribute's contribution to obtaining a pure split, but also on its cost, C_i. Also, the Information Cost Function contains parameter w, which adjusts the strength of the bias towards lower cost attributes. Thus, when $w = 0$, the cost of the attribute is ignored, and selection by ICF is equivalent to selection by the information gain function. On the other hand, when $w = 1$, ICF is strongly biased by the cost component.

The algorithm flow: the algorithm starts by the genetic component evolving a population of randomly generated individuals (an individual corresponds to a decision tree). Each individual in the initial population is then evaluated by measuring its fitness. Standard mutation and crossover operators are applied to the trees population and, after a fixed number of iterations, the fittest individual is returned (Fig. 1).

Each individual is represented as a bit string of $n + 2$ numbers, encoded in Gray. The first n numbers represent the bias parameters ("alleged" test costs in the ICF function). The last two stand for the algorithm's parameters *CF* and w; the first controls the level of pruning (as defined for C4.5), while w is needed by ICF.

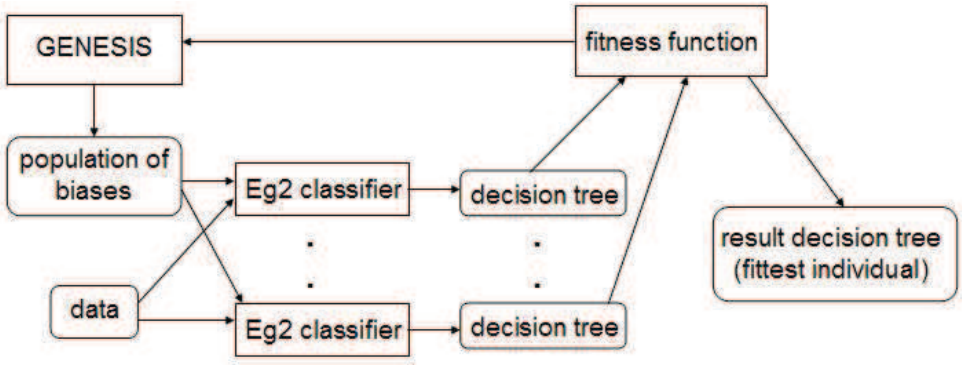


Fig. 1. The ICET technique

An important remark is that, unlike Eg2, ICET does not minimize test costs directly. Instead, it uses ICF for the codification of the individuals in the population. The n costs, C_i, are not true costs, but *bias parameters*. They provide enough variation to prevent the decision tree learner from getting trapped in a local optimum, by overrating/underrating the cost of certain tests based on past trials' performance. However, it is possible to use true costs, when generating the initial population, which has been shown to lead to some increase in performance.

Each trial on an individual consists in training and evaluating a decision tree on a given dataset, using the biases in the individual to set the attribute costs, *CF* and *w*. This is done by splitting the available dataset into two subsets: sub-training and sub-testing dataset. Since the split is random, there may be that two identical individuals will yield different outcomes (since the form of a decision tree is strongly related to the distribution in the training set – different training sets produce different trees).
In ICET, the *fitness function* for an individual is computed as the *average cost* of classification of the corresponding tree (obtained by randomly dividing the training set in two subsets, the first used for the actual tree induction and the second for error estimation). The average cost of classification is obtained by normalizing the total costs (obtained by summing the test and misclassification costs) to the test set size. Test costs are specified as attribute - cost value pairs. The classification costs are defined by a cost matrix $(C_{ij})_{nxn}$, where Cij - the cost of misclassifying an instance of class j as being of class i. If the same attribute is tested twice along the path (numeric attribute), the second time its cost is 0.
The particularity presented by ICET, of allowing the test costs (encoded inside a genetic individual) to vary freely in the search domain, and then applying the fitness evaluation to guide the individuals towards an optimal solution, increases the variability in the heuristic component. Moreover, *w* and *CF* – two key features which influence the future form of a decision tree – are also encoded in the individual, providing even more possibility of variation in the decision trees search space. Theoretically, this variability is desirable, especially for greedy algorithms such as decision tree learners – that yield unique structures for a fixed training set.

4.1 ProICET – improving the basic algorithm

Although ICET has a strong theoretical background, some enhancements can be considered, in order to boost its performance in real-world settings. Most of the changes affect the genetic component, but the training process is slightly different as well. This section also presents other implementation details, and briefly reviews the two tools employed for the current implementation.

4.1.1 Enhancements

First, and most importantly, the *single population technique* is employed as replacement strategy (instead of the multiple populations). In this technique the population is sorted according to the fitness of its elements. At each step two individuals are generated and their fitness is evaluated. According to their score, they are added to the same population their parent elements came from. Then, the individuals with the lowest fitness values are eliminated, so that the size of the population remains the same.
The single population technique has the advantage of directly implementing *elitism*: the best individuals of the current generation can survive unchanged in the next generation. Another prominent feature is the use of *ranking* in the fitness function estimation. The individuals in the population are ordered according to their fitness value, after which probabilities of selection are distributed evenly, according to their rank in the ordered population. Ranking is a very effective mechanism for avoiding the premature convergence of the population, which can occur if the initial pool has some individuals which dominate, having a significantly better fitness than the others.

Some amendments have been considered in the training process as well. Thus, the percentage of the training examples used when evaluating the fitness score of an individual in the population is now 70% of the original training set, as opposed to 50% (in the initial implementation).
The number of evaluation steps has also been increased. Due to the fact that a new generation is evolved using single population, the final result yielded by the procedure is the best individual over the entire run, which makes the decision on when to stop the evolution less critical. More than that, experiments show that usually the best individual does not change significantly after 800 steps: in more than 90% of the cases the algorithm converges before the 800th iteration, while in the rest of the cases the variations after this point are small (less than 3.5%). Therefore, the number of steps in our implementation is 1000.

4.1.2 Implementation overview

The current implementation of the improved ICET technique (*ProICET*) has been done starting from an existing implementation of revision 8 of the C4.5 algorithm, present in Weka (Witten, 2005) and a general genetic tool, GGAT (GGAT, 2002), developed at Brunel University.
Weka (*Waikato Environment for Knowledge Analysis*) is a data mining tool developed at the University of Waikato, New Zealand. It is distributed under the GPL (Gnu Public License) and it includes a wide variety of state-of-the-art algorithms and data processing tools, providing extensive support for the entire process of experimental data mining (input filtering, statistical evaluation of learning schemes, data visualization, preprocessing tools). The command-line interface it provides was particularly useful when invoking the modified decision tree learner for computing the fitness function in the genetic algorithm part of the application.

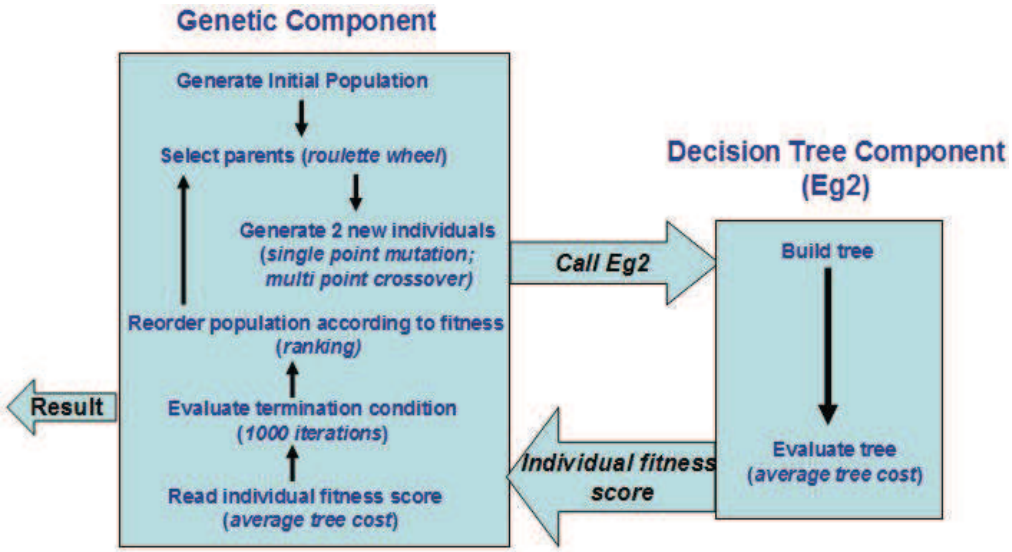


Fig. 2. ProICET main flow

GGAT is a generic GA library, developed at the Brunel University, London. It implements most genetic algorithm mechanisms. Of particular interest are the single population technique and the ranking mechanisms.
In order to obtain the Eg2 attribute selection criterion, as presented in equation (3), the information gain function of J4.8 algorithm was modified, similarly to the implementation presented in (Turney, 1995).
ProICET has been implemented within the framework provided by GGAT. For each individual, the $n + 2$ chromosomes are defined (n being the number of attributes in the data set, while the other two correspond to parameters w and CF); each chromosome is represented as a 14 bits binary string, encoded in Gray. The population size is 50 individuals. The *roulette wheel* technique is used for parent selection; as recombination techniques, we have employed *single point random mutation* with mutation rate 0.2, and *multipoint crossover*, with 4 randomly selected crossover points.
Since the technique involves a large heuristic component, the evaluation procedure assumes averaging the costs over 10 runs. Each run uses a pair of randomly generated training-testing sets, in the proportion 70% - 30%; the same proportion is used when separating the training set into a component used for training and one for evaluating each individual (in the fitness function).

5. Experimental work

A significant problem related to the original ICET technique is rooted in the fact that costs are learned indirectly, through the fitness function. Rare examples are relatively more difficult to be learned by the algorithm. This fact was also observed in (Turney, 1995), where, when analyzing complex cost matrices for a two-class problem, it is noted that: *it is easier to avoid false positive diagnosis [...] than it is to avoid false negative diagnosis [...]. This is unfortunate, since false negative diagnosis usually carry a heavier penalty, in real life.*
Turney, too, attributes this phenomenon to the distribution of positive and negative examples in the training set. In this context, our aim is to modify the fitness measure as to eliminate such undesirable asymmetries.
Last, but not least, previous ICET papers focus almost entirely on test costs and lack a comprehensive analysis of the misclassification costs component. Therefore we attempt to fill this gap by providing a comparative analysis with some of the classic cost-sensitive techniques, such as MetaCost and Eg2, and prominent error-reduction based classifiers, such as J4.8 and AdaBoost.M1.

5.1 Symmetry through stratification

As we have mentioned before, it is believed that the asymmetry in the evaluated costs for two-class problems, as the proportion of false positives and false negatives misclassification costs varies, is owed to the small number of negative examples in most datasets. If the assumption is true, the problem could be eliminated by altering the distribution of the training set, either by oversampling, or by undersampling. This hypothesis was tested by performing an evaluation of the ProICET results on the Wisconsin breast cancer dataset. This particular problem was selected as being one of the largest two-class datasets presented in the literature.

For the stratified dataset, the negative class is increased to the size of the positive class, by repeating examples in the initial set, selected at random, with a uniform distribution. Oversampling is preferred, despite of an increase in computation time, due to the fact that the alternate solution involves some information loss. Undersampling could be selected in the case of extremely large databases, for practical reasons. In that situation, oversampling is no longer feasible, as the time required for the learning phase on the extended training set becomes prohibitive.
The misclassification cost matrix used for this analysis has the form:

$$C = 100 \cdot \begin{pmatrix} 0 & p \\ 1-p & 0 \end{pmatrix}, \tag{4}$$

where p is varied with a 0.05 increment.
The results of the experiment are presented in Fig. 3. We observe a small decrease in misclassification costs for the stratified case throughout the parameter space. This reduction is visible especially at the margins, when costs become more unbalanced. Particularly in the left side, we notice a significant reduction in the total cost for expensive rare examples, which was the actual goal of the procedure.

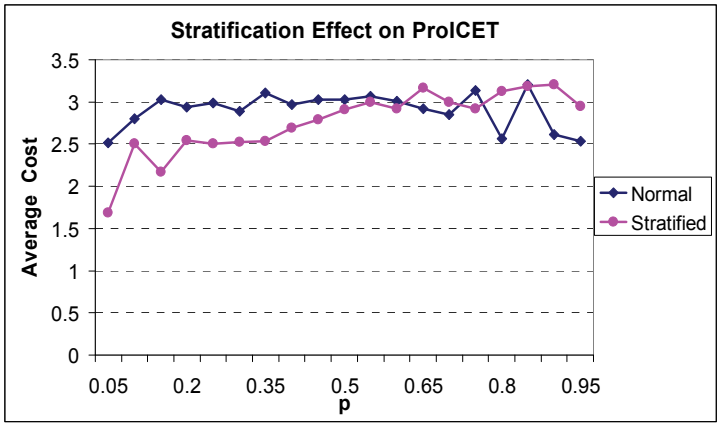


Fig. 3. ProICET average costs for the breast cancer dataset

Starting from the assumption that the stratification technique may be applicable to other cost-sensitive classifiers, we have repeated the procedure on the Weka implementation of MetaCost, using J4.8 as base classifier. J4.8 was also considered in the analysis, as baseline estimate.
The results for the second set of tests are presented in Fig. 4. We observe that MetaCost yields significant costs, as the cost matrix drifts from the balanced case, a characteristic which has been described previously. Another important observation is related to the fact that the cost characteristic in the case of J4.8 is almost horizontal. This could give an explanation of the way stratification affects the general ProICET behavior, by making it insensitive to the particular form of the cost matrix. Most importantly, we notice a general reduction in the average costs, especially at the margins of the domain considered. We

conclude that our stratification technique could be also used for improving the cost characteristic of MetaCost.

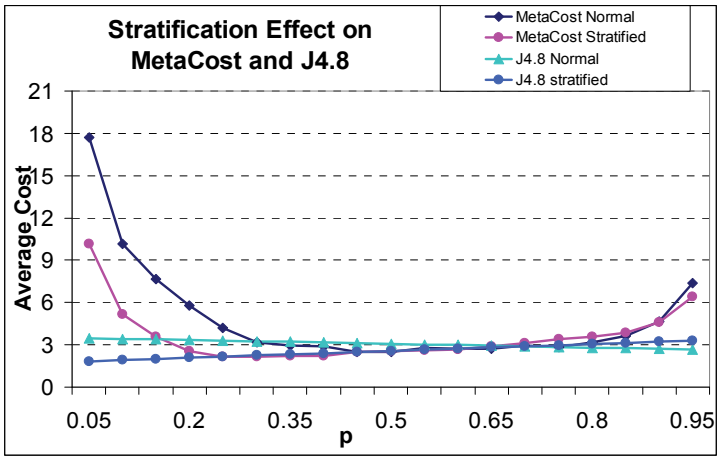


Fig. 4. Improved average cost for the stratified Wisconsin dataset

5.2 Comparing misclassification costs

The procedure employed when comparing misclassification costs is similar to that described in the previous section. Again, the Wisconsin dataset was used, and misclassification costs were averaged on 10 randomly generated training/test sets. For all the tests described in this section, the test costs are not considered in the evaluation, in order to isolate the misclassification component and eliminate any bias.

As illustrated by Fig. 5, MetaCost yields the poorest results. ProICET performs slightly better than J4.8, while the smallest costs are obtained for AdaBoost, using J4.8 as base classifier. The improved performance is related to the different approaches taken when searching for the solution. If ProICET uses heuristic search, AdaBoost implements a procedure that is guaranteed to converge to minimum training error, while the ensemble voting reduces the risk of overfitting. However, the approach cannot take into account test costs, which should make it perform worse on problems involving both types of costs.

5.3 Total cost analysis

When estimating the performance of the various algorithms presented, we have considered four problems from the UCI repository. All datasets involve medical problems: Bupa liver disorders, thyroid, Pima Indian diabetes and heart disease Cleveland. For the Bupa dataset, we have used the same modified set as in (Turney, 1995). Also, the test costs estimates are taken from the previously mentioned study. As mentioned before, the misclassification costs values are more difficult to estimate, due to the fact that they measure the risks of misdiagnosis, which do not have a clear monetary equivalent. These values are set empirically, assigning higher penalty for undiagnosed disease and keeping the order of magnitude as to balance the two cost components (the actual values are displayed in tables 1, 2, 3 and 4).

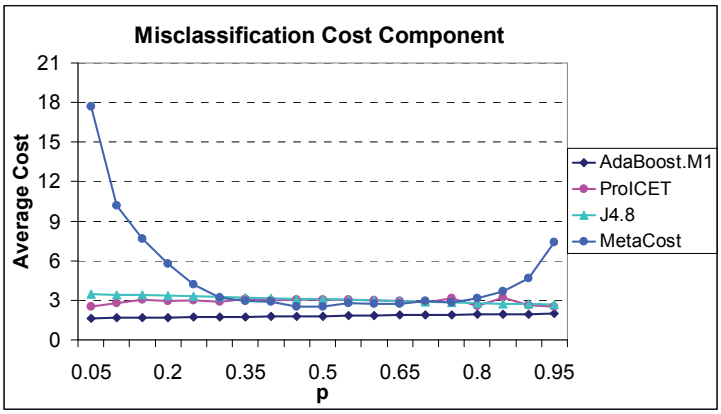


Fig. 5. A comparison of average misclassification costs on the Wisconsin dataset

Class	less than 3	more than
less than 3	0	5
more than 3	15	0

Table 1. Misclassification cost matrix for Bupa liver disorder dataset

Class	3	2	1
3	0	5	7
2	12	0	5
1	20	12	0

Table 2. Misclassification cost matrix for the Thyroid dataset

Class	less than 3	more than
less than 3	0	7
more than 3	20	0

Table 3. Misclassification cost matrix for the Pima dataset

Class	0	1	2	3	4
0	0	10	20	30	40
1	50	0	10	20	30
2	100	50	0	10	20
3	150	100	50	0	10
4	200	150	100	50	0

Table 4. Misclassification cost matrix for the Cleveland heart disease dataset

As anticipated, ProICET significantly outperforms all other algorithms, being the only one built for optimizing total costs (Fig. 6-9). ProICET performs quite well on the heart disease dataset (Fig. 6), where the initial implementation obtained poorer results. This improvement is probably owed to the alterations made to the genetic algorithm, which increase the population variability and extend the ProICET heuristic search.

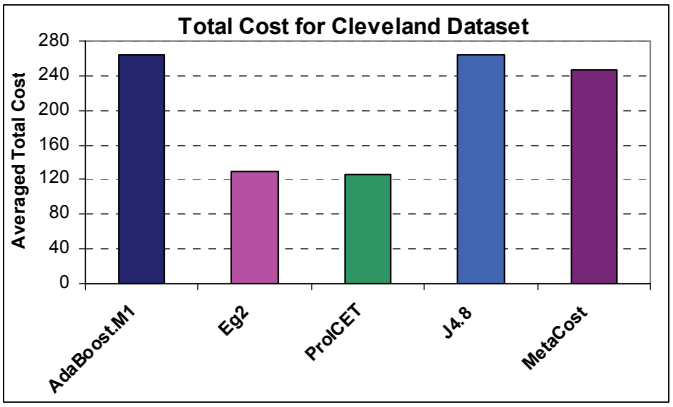


Fig. 6. Average total costs of the considered algorithms on the Cleveland dataset

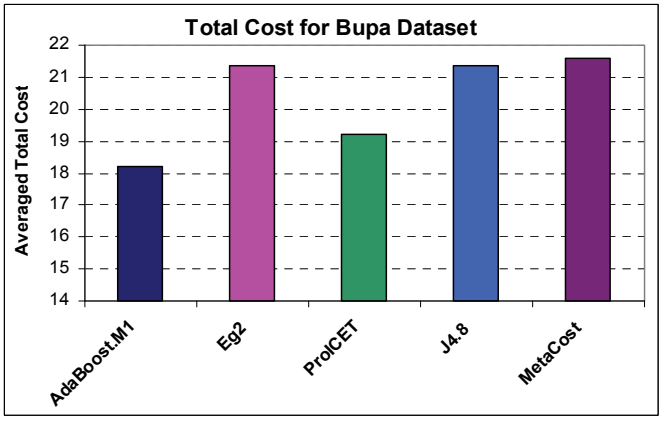


Fig. 7. Average total costs of the considered algorithms on the Bupa dataset

On the Bupa dataset (Fig. 7), AdaBoost.M1 slightly outperforms ProICET, but this is more an exception, since on the other datasets, AdaBoost.M1 yields poorer results. Moreover, the cost reduction performed by ProICET relative to the other methods, on this dataset, is very significant.

The cost reduction is relatively small in the Thyroid dataset (Fig. 8), compared to the others, but is quite large for the other cases, supporting the conclusion that ProICET is the best approach for problems involving complex costs.

6. Chapter summary

This chapter presents the successful combination of two search strategies, greedy search (in the form of decision trees) and genetic search, into a hybrid approach. The aim is to achieve increased performance over existing classification algorithms in complex cost problems, usually encountered when mining real-world data, such as in medical diagnosis or credit assessment.

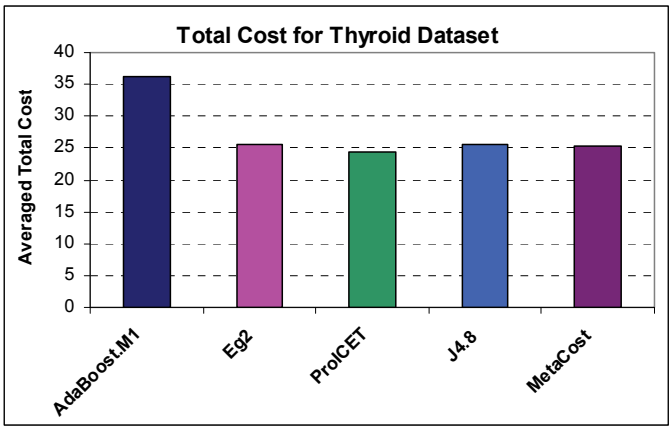


Fig. 8. Average total costs of the considered algorithms on the Thyroid dataset

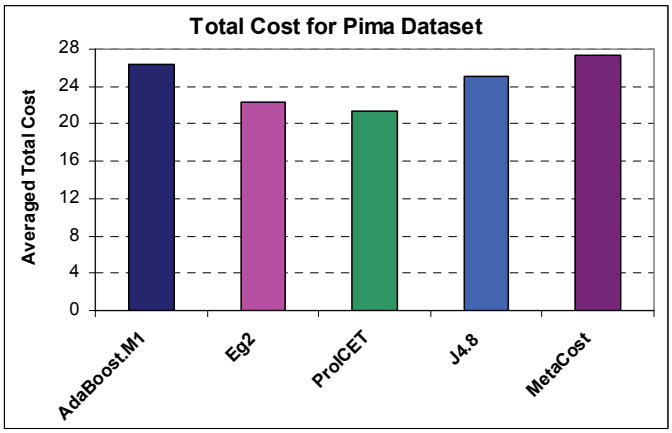


Fig. 9. Average total costs of the considered algorithms on the Pima dataset

Any machine learning algorithm is based on a certain search strategy, which imposes a bias on the technique. There are many search methods available, each with advantages and disadvantages. The distinctive features of each search strategy restrict its applicability to certain problem domains, depending on which issues (dimensionality, speed, optimality, etc.) are of importance. The dimension of the search space in most real-world problems renders the application of complete search methods prohibitive. Sometimes we have to trade optimality for speed. Fortunately, greedy search strategies, although do not ensure optimality, usually provide a sufficiently good solution, close to the optimal one. Although they have an exponential complexity in theory, since they do not explore the entire search space, they have a very good behaviour in practice, in speed terms. This makes them suitable even for complex problems. Their major drawback comes from the fact that they can get caught at local optima. Since the complexity of the search space is too large, such that the problem is intractable for other techniques, in most real problems this is an accepted disadvantage. Greedy search strategies are employed in many machine learning algorithms.

One of the most prominent classification techniques which employ such a strategy are decision trees.

The main advantages of decision trees are: an easy to understand output model, robustness with respect to the data quantity, little data preparation, ability to handle both numerical and categorical data, as well as missing data. Therefore, decision trees have become one of the most widely employed classification techniques in data mining, for problems where error minimization is the target of the learning process.

However, many real-world problems require more complex measures for evaluating the quality of the learned model. This is due to the unbalance between different types of classification errors, or the effort of acquiring the values of predictive attributes. A special category of machine learning algorithms focuses on this task – cost-sensitive learning. Most existing techniques in this class focus on just one type of cost, either the misclassification, or the test cost. Stratification is perhaps the earliest misclassification cost-sensitive approach (a sampling technique rather than an algorithm). It has been followed by developments in the direction of altering decision trees, such as to make their attribute selection criterion sensitive to test costs (in the early 90's). Later, new misclassification cost-sensitive approaches emerged, the best known being MetaCost or AdaCost. More recent techniques consider both types of cost, the most prominent being ICET.

Initially introduced by Peter D. Turney, ICET is a cost-sensitive technique, which avoids the pitfalls of simple greedy induction (employed by decision trees) through evolutionary mechanisms (genetic algorithms). Starting from its strong theoretical basis, we have enhanced the basic technique in a new system, ProICET. The alterations made in the genetic component have proven beneficial, since ProICET performs better than other cost-sensitive algorithms, even on problems for which the initial implementation yielded poorer results.

7. References

Baezas-Yates, R., Poblete, V. P. (1999). Searching, In: *Algorithms and Theory of Computation Handbook*, Edited by Mikhail J. Atallah, Purdue University, CRC Press

Domingos, P. (1999). Metacost: A general method for making classifiers cost-sensitive. *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining,* pp. 155-164, 1-58113-143-7, San Diego, CA, USA

Fan, W.; Stolfo, S.; Zhang, J. & Chan, P. (2000). AdaCost: Misclassification cost-sensitive boosting. *Proceedings of the 16th International Conference on Machine Learning*, pp. 97–105, Morgan Kaufmann, San Francisco, CA

Freund, Y. & Schapire, R. (1997). A decision-theoretic generalization of on- line learning and an application to boosting. *Journal of Computer and System Sciences*, Volume 55, Number 1, August 1997 , pp. 119-139

General Genetic Algorithm Tool (2002), GGAT, http://www.karnig.co.uk/ga/content.html, last accessed on July 2008

Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16, 122-128

Korf, R. E. (1999). Artificial Intelligence Search Algorithms, In: *Algorithms and Theory of Computation Handbook*, Edited by Mikhail J. Atallah, Purdue University, CRC Press

Norton, S.W. (1989). Generating better decision trees. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89,* pp. 800-805. Detroit, Michigan.

Núñez, M. (1988). Economic induction: A case study. *Proceedings of the Third European Working Session on Learning, EWSL-88*, pp. 139-145, California, Morgan Kaufmann.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, ISBN:1-55860-238-0, San Francisco, CA, USA

Quinlan, J. (1996) Boosting first-order learning. *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, 1160:143–155

Russell, S., Norvig, P. (1995) *Artificial Intelligence: A Modern Approach*, Prentice Hall

Tan, M., & Schlimmer, J. (1989). Cost-sensitive concept learning of sensor use in approach and recognition. *Proceedings of the Sixth International Workshop on Machine Learning, ML-89,* pp. 392-395. Ithaca, New York

Tan, M., & Schlimmer, J. (1990). CSL: A cost-sensitive learning system for sensing and grasping objects. *IEEE International Conference on Robotics and Automation*. Cincinnati,Ohio

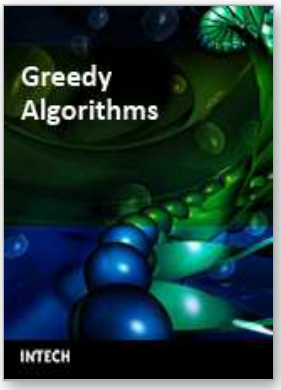
Turney, P. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, Volume 2, pp. 369–409

Turney, P. (2000). Types of cost in inductive concept learning. *Proceedings of the Workshop on Cost-Sensitive Learning, 7th International Conference on Machine Learning,* pp. 15-21

Vidrighin, B. C., Savin, C. & Potolea, R. (2007). A Hybrid Algorithm for Medical Diagnosis. *Proceedings of Region 8 EUROCON 2007*, Warsaw, pp. 668-673

Vidrighin, C., Potolea, R., Giurgiu, I. & Cuibus, M. (2007). ProICET: Case Study on Prostate Cancer Data. *Proceedings of the 12th International Symposium of Health Information Management Research,* 18-20 July 2007, Sheffield, pp. 237-244

Witten I. & Frank, E. (2005) *Data Mining: Practical machine learning tools and techniques,* 2nd *ed.* Morgan Kaufmann, 0-12-088407-0



Greedy Algorithms

Edited by Witold Bednorz

ISBN 978-953-7619-27-5
Hard cover, 586 pages
Publisher InTech
Published online 01, November, 2008
Published in print edition November, 2008

Each chapter comprises a separate study on some optimization problem giving both an introductory look into the theory the problem comes from and some new developments invented by author(s). Usually some elementary knowledge is assumed, yet all the required facts are quoted mostly in examples, remarks or theorems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Camelia Vidrighin Bratu and Rodica Potolea (2008). Enhancing Greedy Policy Techniques for Complex Cost-Sensitive Problems, Greedy Algorithms, Witold Bednorz (Ed.), ISBN: 978-953-7619-27-5, InTech, Available from:
http://www.intechopen.com/books/greedy_algorithms/enhancing_greedy_policy_techniques_for_complex_cost-sensitive_problems

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821