Detection of Rare Items with TARGET

Guangzhe Fan, Mu Zhu
Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, {gfan@uwaterloo.ca, m3zhu@uwaterloo.ca}

In our new information-based economy, the need to detect a small number of relevant and useful items from a large database arises very often. Standard classifiers such as decision trees and neural networks are often used directly as a detection algorithm. We argue that such an approach is not optimal because these classifiers are almost always built to optimize a criterion that is suitable only for classification but not for detection. For detection of rare items, the misclassification rate and other closely associated criteria are largely irrelevant; what matters is whether the algorithm can rank the few useful items ahead of the rest, something better measured by the notion of the average precision (AP). We use the genetic algorithm to build decision trees by optimizing the AP directly and compare the performance of our algorithm with a number of standard classifiers using both simulated and real data sets.

Key words: average precision; classification; decision tree; fraud detection; genetic algorithm

History:

1. Detection of Rare Items

Suppose we have a large collection of items, $C$, of which only a fraction $\pi$ ($\pi \ll 1$) is relevant to us. We are interested in computational tools to help us identify and single out these items. The reason an item is considered relevant depends on the context of a specific problem. For example, insurance (credit card) companies are very interested in detecting fraudulent claims (transactions). In such applications, the relevant items are the few fraudulent cases among thousands or millions of normal ones.

    Typically, we have a training data set $\{(y_i, x_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$ is a vector of predictors, $y_i = 1$ if observation $i$ is relevant and $y_i = 0$ if otherwise. Supervised learning methods (e.g., classification trees, neural networks) are used to build a predictive model using the training data. The model is then used to screen a large number of new cases; it often produces a relevance score or an estimated probability for each of these new cases. The top-ranked
cases can then be passed onto further stages of investigation. Figure 1 provides a schematic illustration of this process.

If it is decided that the top 50 cases should be investigated further, we shall say that these 50 cases are the ones “detected” by the algorithm, although, strictly speaking, the algorithm really does not detect these cases \textit{per se}; it merely ranks them as being more likely than others to be what we want.

The data structure and the types of supervised learning methods often used here are similar to those encountered in a standard two-class classification problem. However, the underlying objective is very different. In particular, automatic classification is of little interest. This is because further investigation of the top-ranked candidates is almost always necessary. For example, an insurance or credit card company would seldom refuse a claim or terminate an account without confirming the suspected fraud. Therefore, we are most interested in producing an effective \textit{ranking} of all the candidates so that any further investigation (often very expensive) is least likely to be carried out in vain.

1.1. Performance Evaluation

Clearly, misclassification rate is quite irrelevant for a problem of this nature. For example, if class 1 consists of only 5% of all cases, then an algorithm that simply classifies everything into class 0 will have a misclassification rate of only 0.05, but it is clearly not a useful algorithm. See also Bolton and Hand (2002). Recently, researchers in the machine learning community
are starting to pay more attention to performance criteria other than the misclassification rate such as the area under the receiver operating characteristic (ROC) curve (see, e.g., Cortes and Mohri, 2004).

For detection problems, the most relevant evaluation criterion here is whether the relevant items can be ranked ahead of the rest by the algorithm. Even though the area under the ROC curve is also related to ranking, the performance measure most widely-used in the information retrieval community for ranking purposes is the notion of the average precision (AP; see, e.g., Deerwester et al., 1990; Dumais, 1991; Peng et al., 2003). A brief description of how the AP is defined and calculated is given in the Appendix. For most readers, it suffices to know that, if algorithm A has a larger AP than algorithm B, then algorithm A can be regarded as the better algorithm.

1.2. Challenges and Motivation

As mentioned earlier, standard classifiers such as classification trees (Breiman et al., 1984), support vector machines (SVMs; e.g., Cristianini and Shawe-Taylor, 2000), and neural networks (e.g., Ripley, 1996) are often used directly as detection algorithms. Here is why we believe such an approach is not fully optimal. Let \( f(x; \theta) \) be a general classifier completely specified by a set of parameters which we shall write simply as \( \theta \); \( \theta \) is often chosen to minimize a certain loss function, e.g., the “hinge” loss for SVMs and the exponential loss for AdaBoost (see, e.g., Hastie et al., 2001, Sections 10.6 and 12.3.2). However, these commonly used loss functions can all be viewed as continuous approximations to the misclassification rate (Hastie et al., 2001, Section 10.6) and are, therefore, not entirely appropriate for the detection problem. Certainly, the key model parameter \( \theta \) is never chosen to optimize the average precision (AP) directly.

In practice, the AP can be used to guide the construction of these classifiers to a certain degree. For example, if the underlying problem is one of detection rather than classification, one would generally choose the tuning parameters of an algorithm using the AP (rather than, e.g., the misclassification rate) as the guiding criterion. We argue that a fully optimal approach would be to use the AP to choose not only the tuning parameters but also the the model parameter \( \theta \).

This, however, is often difficult to achieve because the AP depends on how the items are ranked and is, therefore, not generally a smooth function of \( \theta \). In this article, we present an algorithm to optimize the AP directly over \( \theta \) when \( f(x; \theta) \) is restricted to be a decision
tree. Developing algorithms that directly optimize the AP without requiring \( f(x; \theta) \) to be a decision tree is part of our ongoing research program.

2. TARGET

TARGET, which stands for “Tree Analysis with Randomly Generated and Evolved Trees,” is first developed by Gray and Fan (2003). Here, we further develop TARGET into an effective detection tool for identifying rare items.

It is well-known that the recursive partitioning algorithm used by CART and other similar software for building decision trees is a greedy algorithm. At each stage the algorithm searches for a locally optimal split to grow a tree but the final product is not necessarily the best overall tree. Instead of a greedy search algorithm, TARGET uses a stochastic search algorithm known as the genetic algorithm (GA; Goldberg, 1989) to build the decision tree. Here we use the GA to optimize the tree model with respect to the average precision criterion to detect rare items, a task that is in principle not well-suited for various greedy tree search algorithms. Another example of using the GA for rare events is the paper by Weiss and Hirsh (1998). They used the GA to optimize a rule-based expert system to predict rare events in categorical time-series data.

Starting with a number of randomly generated candidates (the initial population), the GA applies the Darwinian principle of “the survival of the fittest” and gradually eliminates the weaker (less optimal) candidates and allows the stronger ones to survive and generate offsprings. This goes on for a number of generations until good solutions are produced in the end. We use \( N \) to denote the population size, which TARGET keeps fixed for all generations. Given a random initial population of size \( N \), a new generation of the same size is produced with a number of genetic operations: elitism, crossover, mutation and transplant; details are given below.

2.1. Initialization

The initial population consists of \( N \) randomly created decision trees. TARGET uses a default of \( N = 50 \). To randomly generate a tree, we start with a single root node. With probability \( p_{\text{split}} \), the node is split and two child nodes are created; otherwise, the node becomes a terminal node. If the node is split, then a split rule, which includes a split variable and a split set, is randomly chosen from all candidate split rules and assigned to that node. The
recursive node-splitting process then continues with the child nodes until no more splits are to be made. The node splitting probability $p_{\text{split}}$ is used to control the average tree size in the initial forest. The default value used by TARGET is $p_{\text{split}} = 0.5$. After a tree is randomly created, the training data are run through the tree down to the terminal nodes. Since the split rules are randomly assigned, some nodes may be empty or have too few observations. These small or empty nodes are pruned from the tree before the fitness of the tree is evaluated. We shall say more about how the fitness is evaluated below (Section 2.4).

2.2. Genetic Operations

The $N$ randomly initialized trees consist of the first generation in the evolutionary process. Given a current generation, the following genetic operations are performed, in the order listed, to create a new generation. Trees with larger fitness values are given higher probability of being selected for the genetic operations (with selection probability proportional to the trees’ fitness values). The user can specify different proportions of the new generation to be constructed by different genetic operations. The default used by TARGET is as follows: 10% by elitism, 60% by crossover, 20% by mutation and another 10% by transplant.

Elitism A fixed number (default $= 0.1 \times N$) of trees with the best fitness values in the current generation are copied to the next generation.

Crossover Two parent trees are randomly selected with probabilities proportional to their fitness values and a node is randomly chosen on each tree. Then, either the two nodes are swapped (node swap crossover) or the two sub-trees are swapped (subtree swap crossover). The user can specify the probabilities of the two types of swaps; the default used by TARGET is to assign equal probabilities to the two types. This results in two new trees or offsprings. In node-swap crossover, only the split rules are actually swapped. Moreover, crossover is not performed if two root nodes or two leaves are selected. A crossover rule is used to determine which of the four trees involved in a crossover operation (two parents and two offsprings) are added to the next generation. Possible rules include adding both offsprings, adding the better of the two offsprings, adding the best of the four, or adding the best two of the four. The choice is left up to the user; the default used by TARGET is to add the best of the four.
Mutation  Mutation introduces new “genetic material” and can help the genetic search process avoid getting trapped at local optima. We randomly select a single tree from the current population with probability proportional to its fitness value and randomly perform one of following four types of mutation:

(1) split set mutation — randomly select a node, keep the same split variable there but change its split set, randomly;

(2) split rule mutation — randomly select a node and change the entire split rule, including both the split variable and its split set;

(3) node swap mutation — randomly select two nodes and swap their split rules; and

(4) subtree swap mutation — randomly select two sub-trees within the tree and swap them.

The user can specify the probabilities for the four types of mutation; the default used by TARGET is to assign equal probabilities to the four types.

Transplant  A number (default = 0.1 × N) of new randomly generated trees are added to the next generation. This adds more “genetic material” to the evolutionary process and provides additional opportunities for the algorithm to avoid local optima.

2.3. Termination

TARGET provides a convenient graphic interface that allows the user to monitor the fitness values along the evolutionary process. The user can stop the algorithm if there is no improvement in the best tree over several generations.

2.4. The Fitness Function

Other than the ability to conduct a non-greedy search, the biggest advantage of TARGET is perhaps its ability to optimize almost any objective function. The fact that a certain objective function is not a smooth function of the model parameter θ is inconsequential. For the detection of rare items, it is easy to search for a decision tree to maximize the average precision directly. In principle, we simply use the AP as the fitness function. In reality, extra care must be taken to avoid overfitting the training data. To do so, we add a penalty
proportional to the size of the tree. The final fitness function used to evaluate a tree $T$ can be written as

$$\text{fitness}(T) = \text{AP}(T) - \alpha |T|,$$

where $|T|$ is the total number of terminal nodes.

2.4.1. The Choice of $\alpha$

The proportionality constant $\alpha$ can be regarded as a tuning parameter of the algorithm and chosen empirically by cross-validation. The default used by TARGET is $\alpha = 0.01\pi$, where $\pi$ is the fraction of rare items in the training set. If $\pi = 5\%$, this choice of $\alpha$ means an extra terminal node is permissible only if it increases the AP of the corresponding decision tree by more than 0.05%.

Since the number of observations belonging to class 1 is usually very small, our empirical experiences suggest that cross-validation can be fairly unstable for these rare target problems. To perform cross-validation, the training data must first be partitioned randomly into several groups. We have noticed high variability in the optimal $\alpha$ found by cross-validation depending on how the data are partitioned initially. Based on all the experiments we have conducted so far, however, the average choice made by cross-validation is roughly $\alpha \approx 0.01\pi$. We think this default setting actually offers a much more stable as well as computationally efficient alternative than cross-validation.

3. Experiments

We now conduct a number of experiments using both simulated and real data sets to evaluate the performance of TARGET.

3.1. Experimental Settings

We use two well-known algorithms as benchmarks: CART (Breiman et al., 1984) and Random Forest (RF; Breiman, 2001). Both are tree-based algorithms and hence appropriate benchmarks for TARGET. The RF is also well-known to be a very powerful algorithm whose predictive performance is generally difficult to beat.

The software we use to run these algorithms are the rpart and randomForest libraries from R (R Development Core Team, 2004). For TARGET, we use 500 generations while
setting all its control parameters including the penalty $\alpha$ to their default values. For Random Forest (RF), we use a forest size of 200 trees. For CART, instead of using cross-validation on the training data to prune the tree, we simply report results for two different trees, which we refer to as CART-A and CART-B.

CART-A refers to a tree pruned back to be roughly the same size as the one found by TARGET. CART-B refers to a tree that gives the best test set performance. We emphasize that CART-B is a tree selected after having peeked at the labels in the test set. It is well understood that this would give CART-B an unfair advantage. Here, we simply report this as an upper bound for the best possible performance by CART. The readers are reminded that, if we had pruned the tree with cross-validation, the resulting CART performance would not have exceeded that of CART-B.

### 3.2. Data Sets

We now briefly describe the data sets.

#### 3.2.1. A Simulated Data Set (Simulated)

In this simulated data set, $\mathbf{x} \in [0, 1]^5$ is generated uniformly inside the 5-dimensional unit cube. To generate $y$, only the first two dimensions of $\mathbf{x}$ are used: $P(y|\mathbf{x})$ is an increasing function of $x_1 + x_2$ for $y = 1$ and a decreasing function of $x_1 + x_2$ for $y = 0$; see Figure 2. The maximum of $P(y = 1|\mathbf{x})$ is equal to 0.6, which occurs if $x_1 + x_2 > 1.8$. A total of $n = 5,000$ observations are generated, of which 777 belongs to class 1. These observations are randomly split into a training and a test set consisting of 2,500 observations each.

#### 3.2.2. Breast Cancer Data (Cancer)

This is the well-known breast cancer data set from the UCI machine learning repository (ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin/). The raw data set consists of 699 observations, each with 9 predictors. There are 16 observations with missing predictors; these are not used, resulting in a total of 683 observations, of which 239 are cancer cases. We randomly select 150 observations as the test set and use the remaining 533 observations as the training set.

Notice that the cancer cases are not exactly “rare” in this data set, although the proportion is still well below 50%. The main reason why this data set is still included is because both TARGET and CART have produced, for this data set, relatively small decision trees
Figure 2: Simulation scheme. $P(y|x)$ is an increasing function of $x_1 + x_2$ for $y = 1$ and a decreasing function of $x_1 + x_2$ for $y = 0$.

that we can easily display here (Figure 3) and we believe the difference in the resulting tree structures is informative.

3.2.3. Fraud Detection Data (Fraud)

This is an automobile insurance fraud data set available from Pyle (1999). The data set consists of 15,420 records of insurance claims made between 1994 and 1996. For each record, there are 32 variables (such as the make of the car, the driver’s rating, the amount of the deductible on the policy, the area of the accident, and so on) as well as an indicator whether fraud is detected. There are altogether 923 records in the fraud category. Not all 32 variables are used for prediction. The variable “policy number” is not used and the variable “year” is only used to partition the data into a training and a test set. In particular, claims made in 1994 (a total of 6,142) are used as training data and claims made in 1995 and 1996 (a total of 9,278) are used as test data.

Most of the variables in this data set are categorical, making it difficult and, indeed, inappropriate to use kernel-based or distance-based algorithms such as support vector machines (SVMs; e.g., Cristianini and Shawe-Taylor, 2000) and K-nearest neighbors (KNN; Cover and
This is one reason why these algorithms are not used as benchmarks in our study.

Table 1: Average precision on test data.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Data Sets</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simulated</td>
<td>Cancer</td>
<td>Fraud</td>
<td></td>
</tr>
<tr>
<td>CART-A</td>
<td>0.258</td>
<td>0.875</td>
<td>0.104</td>
<td></td>
</tr>
<tr>
<td>CART-B</td>
<td>0.300</td>
<td>0.911</td>
<td>0.109</td>
<td></td>
</tr>
<tr>
<td>TARGET</td>
<td>0.320</td>
<td>0.972</td>
<td>0.130</td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.317</td>
<td>0.980</td>
<td>0.117</td>
<td></td>
</tr>
</tbody>
</table>

* CART-B has an unfair advantage; it merely provides an upper bound on the performance of CART. See Section 3.1 for details.

3.3. Results

Table 1 lists the average precision of various methods on different test data. First of all, it is clear that the breast cancer data set is considerably easier to predict than the other two data sets. In this case, both TARGET and CART have produced relatively small decision trees that we can display here. Figure 3 shows the structure of the two decision trees on the training set, which contains 184 cancer cases. This plot is informative and shows that, by maximizing the AP directly, we do get a very different tree. Table 2 shows the proportion of cancer cases correctly identified by these two decision trees as we vary the threshold on the estimated posterior probability $p \equiv P(y = 1|x)$. The TARGET tree produces a much more effective ranking of all the cases. Table 1 shows that TARGET does, in fact, outperform CART on all the data sets.

Table 2: Breast cancer data. Proportion of cancer cases correctly identified at different thresholds.

<table>
<thead>
<tr>
<th>Threshold on $p$</th>
<th>1.00</th>
<th>0.98</th>
<th>0.95</th>
<th>0.40</th>
<th>0.20</th>
<th>0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART</td>
<td>19/184</td>
<td>19/184</td>
<td>176/184</td>
<td>179/184</td>
<td>179/184</td>
<td>184/184</td>
</tr>
<tr>
<td>TARGET</td>
<td>66/184</td>
<td>148/184</td>
<td>169/184</td>
<td>180/184</td>
<td>184/184</td>
<td>184/184</td>
</tr>
</tbody>
</table>

Random Forest, on the other hand, is much more competitive than CART in this regard. This is because, by averaging results over a large number of trees, Random Forest is capable of estimating the posterior probability $P(y = 1|x)$ much more accurately. The posterior probability $P(y = 1|x)$ can be regarded as an ideal ranking function: if one had perfect estimates of the posterior probability, one could use them to rank all the items and obtain
an average precision of 100%. However, despite it being a very powerful method, Random Forest only slightly outperforms TARGET on the easiest data set, the breast cancer data. On the two more difficult data sets, Random Forest is seen to be inferior to TARGET.

Most importantly, the output from Random Forest is no longer a single (and hence easily interpretable) decision tree. That TARGET can produce a very effective ranking with a single, easily interpretable decision tree makes it a very attractive detection method in practice.

![Decision Trees](image)

Figure 3: Breast cancer data. Comparison of decision trees found by TARGET and CART.

4. Conclusion

We have argued that it is not optimal to use a standard classifier directly as a detection algorithm; a better approach for constructing a classifier $f(x; \theta)$ to be used as a detection algorithm is to use the average precision to choose not only the tuning parameters but also the the model parameter $\theta$. Experimental results using TARGET (which uses the genetic algorithm to build decision trees that maximize the AP directly) support this general argument.
Appendix: Average Precision

We shall only give a very brief summary of how the average precision is defined and computed. Out of the $t \times 100\%$ top-ranked candidates, suppose $h(t) \leq t$ are truly relevant; these are often called “hits” (as opposed to “misses”). Figure 4 provides a schematic illustration. Let $r(t) = h(t)/\pi$ and $p(t) = h(t)/t$; these quantities are respectively known as the recall and the precision in the information retrieval literature (e.g., Buckland and Gey, 1994; Gordon and Kochen, 1989). In practice, $h(t)$ and hence $r(t)$ and $p(t)$ all take values only at a finite number of points $t_i = i/n$, $i = 1, 2, ..., n$. The formula for calculating the average precision is as follows:

$$AP = \sum_{i=1}^{n} p(t_i) \Delta r(t_i)$$  \hspace{1cm} (2)

where $\Delta r(t_i) = r(t_i) - r(t_{i-1})$. Here is a concrete example. Table 3 summarizes the performance of three hypothetical methods, A, B and P. Success for a detection algorithm, again, means the relevant items are ranked ahead of the rest. In this simple example, it is easy to see that algorithm A is superior to algorithm B since it detects the three hits earlier on. By (2), we get

$$AP(A) = \sum_{i=1}^{5} p(t_i) \Delta r(t_i) = \left( \frac{1}{1} + \frac{2}{2} + \frac{3}{4} \right) \times \frac{1}{3} \approx 0.92,$$

and

$$AP(B) = \sum_{i=1}^{5} p(t_i) \Delta r(t_i) = \left( \frac{1}{1} + \frac{2}{4} + \frac{3}{5} \right) \times \frac{1}{3} = 0.70,$$
which agrees with our intuition. Algorithm P is a perfect algorithm since it ranks all the relevant items ahead of the rest; it can be easily verified from (2) that such an algorithm would have an average precision of 100%.

| Item (i) | Algorithm A | | Algorithm B | | Algorithm P |
|---------|-------------|-------------|-------------|-------------|
|         | Hit $p(t_i)$ $\Delta r(t_i)$ | Hit $p(t_i)$ $\Delta r(t_i)$ | Hit $p(t_i)$ $\Delta r(t_i)$ |
| 1       | 1 1/1 1/3 | 1 1/1 1/3 | 1 1/1 1/3 |
| 2       | 1 2/2 1/3 | 0 1/2 0   | 1 2/2 1/3 |
| 3       | 0 2/3 0   | 0 1/3 0   | 1 3/3 1/3 |
| 4       | 1 3/4 1/3 | 1 2/4 1/3 | 0 3/4 0   |
| 5       | 0 3/5 0   | 1 3/5 1/3 | 0 3/5 0   |

### Acknowledgments

This research is partially supported by the National Science and Engineering Research Council (NSERC) of Canada.

### References


