Statistics and Its Interface Volume 4 (2011) 11–17

Detection of rare items with TARGET

Guangzhe Fan* and Mu Zhu

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 area under the ROC curve or 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 tree-based classifiers using both simulated and real data sets.

Keywords and phrases: Average precision, Classification, Decision tree, Fraud detection, Genetic algorithm.

1. INTRODUCTION

Suppose we have a large collection of items, $\mathcal{C}$, of which only a fraction π ($\pi \ll 1$) 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 transactions.

Typically, we have a training data set $\{(y_i, \mathbf{x}_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbf{R}^d$ 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, support vector machines, 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, and produces a relevance score or an estimated probability for each case. 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 *per se*; it merely ranks them as being more likely than others to be what we want.

*Corresponding author.

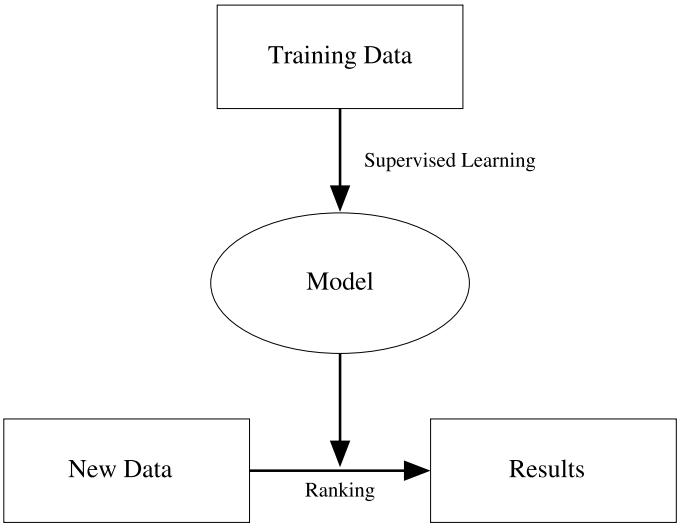

Figure 1. Illustration of the typical modelling and prediction process. Source: Zhu et al. [21].

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 *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, e.g., [1]).

For detection problems, the most relevant evaluation criterion here is whether the relevant items can be ranked

ahead of the rest by the algorithm, something that can be measured by, for example, the area under the receiver operating characteristic (ROC) curve (simply AUC for "area under the curve" below) (see, e.g., [16]), and/or the average precision (simply AP below) (see, e.g., [7, 8, 15]). A brief description of how the AP is defined and calculated, particularly for tree-based methods that often produce many tied ranks, is given in Appendix A. 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 for ranking purposes.

1.2 Challenges and motivation

As mentioned earlier, standard classifiers such as classification trees [3], support vector machines (SVMs; e.g., [6]), and neural networks (e.g., [19]) are often used directly as detection algorithms. Here is why we believe such an approach is not fully optimal. Let $f(\mathbf{x};\boldsymbol{\theta})$ be a general classifier completely specified by a set of parameters which we shall write simply as $\boldsymbol{\theta}$; $\boldsymbol{\theta}$ 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., [13, Sections 10.6 and 12.3.2]). However, these commonly used loss functions can all be viewed as continuous approximations to the misclassification rate [13, Section 10.6] and are, therefore, not entirely appropriate for the detection problem. Certainly, the key model parameter $\boldsymbol{\theta}$ is never chosen to optimize the AP or the AUC directly.

In practice, the AP and the AUC *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 or the AUC (rather than, e.g., the misclassification rate) as the guiding criterion. For ranking problems, we think it is desirable to develop a fully optimal approach which uses the AP or the AUC to choose not only the tuning parameters but also the model parameter $\boldsymbol{\theta}$.

This, however, is often difficult to achieve because the AP and the AUC depend on how the items are ranked and are, therefore, not generally smooth functions of $\boldsymbol{\theta}$. Recently, there have been some attempts in the machine learning community to optimize the AUC directly (e.g., [5]). In this article, we present an algorithm to optimize the AP directly over $\boldsymbol{\theta}$ when $f(\mathbf{x};\boldsymbol{\theta})$ is restricted to be a decision tree. Developing algorithms that directly optimize the AP without requiring $f(\mathbf{x};\boldsymbol{\theta})$ 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 [9, 12]. 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; [10]) 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 [20]. 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 offspring. 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_{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_{split} is used to control the average tree size in the initial forest. The default value used by TARGET is $p_{split} = 0.5$. 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.

- **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 offspring. 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 offspring) are added to the next generation. Possible rules include adding both offspring, adding the better of the two offspring, 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 $= 0.1 \times N$) 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.

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.

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. Then the best tree of the final generation is used as the final model.

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 $\boldsymbol{\theta}$ 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

$$\text{fitness}(T) = \text{AP}(T) - \alpha|T|, \tag{1}$$

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

The proportionality constant α 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 π is the fraction of rare items in the training set. If $\pi = 5\%$, this choice of α means an extra terminal node is permissible only if it increases the AP of the corresponding decision tree by more than 0.05%.

3. EXPERIMENTS

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

1. **Simulation 1.** In this simulation, $\mathbf{x} \in [0,1]^5$ is generated uniformly inside the 5-dimensional unit cube. To generate y, only the first two dimensions of $\mathbf{x}$ are used: $P(y = 1|\mathbf{x})$ is a piece-wise constant and increasing function of $x_1 + x_2$; see Figure 2. A total of $n = 5,000$ observations are generated, of which roughly 15% belongs to class 1. The experiments are repeated 5 times. Each time, the observations are randomly split into a training and a test set consisting of 2,500 observations each.
2. **Simulation 2.** In this simulation, we first simulate from three 4-dimensional multivariate normal populations. A total of 4,000 samples from $\text{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ are labeled as class 0; 700 samples from $\text{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ and 300 samples from $\text{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$ are labeled as class 1, where

$$\boldsymbol{\mu}_1 = (0.9, 0.5, 0.9, 0.5)^T,$$
$$\boldsymbol{\mu}_2 = (0.1, 0.1, 0.3, 0.7)^T,$$
$$\boldsymbol{\mu}_3 = (0.4, 0.0, 0.0, 0.2)^T;$$

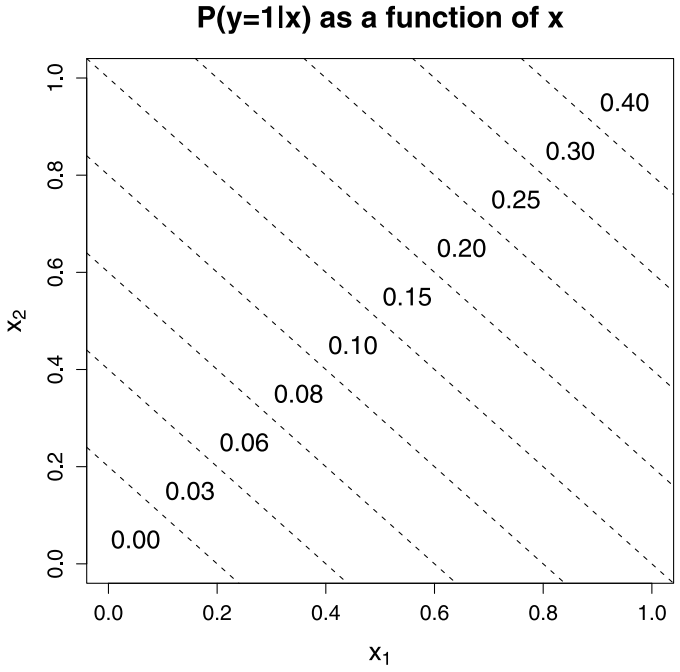

Figure 2. $P(y = 1|\mathbf{x})$ *as a function of* $\mathbf{x}$*, used to generate data for simulation 1.*

$$\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = \begin{bmatrix} 1.0 & 0.2 & 0.0 & -0.5 \\ 0.2 & 1.0 & 0.6 & -0.2 \\ 0.0 & 0.6 & 1.0 & 0.0 \\ -0.5 & -0.2 & 0.0 & 1.0 \end{bmatrix};$$

and

$$\mathbf{\Sigma}_3 = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.5 \\ 0.0 & 1.0 & -0.5 & 0.2 \\ 0.0 & -0.5 & 1.0 & 0.0 \\ 0.5 & 0.2 & 0.0 & 1.0 \end{bmatrix}.$$

In other words, there are a total of 5,000 observations, of which 20% belong to class 1. Then, 11 independent noise features, uniformly generated from $[0, 1]$, are added to every observation. The experiments are repeated 5 times. Each time, the observations are randomly split into a training and a test set consisting of 2,500 observations each.

3. **Breast cancer data.** 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 partition the data into 5 roughly equal-sized blocks, and repeat the experiments 5 times, each time using one block as the test set and the other 4 blocks as the training set.
4. **Insurance fraud data.** This is an automobile insurance fraud data set available from Pyle [17]. 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. Here, the experiments are repeated twice. In the first experiment, 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. In the second experiment, claims made in 1995 (a total of 5,195) are used as training data and those made in 1996 (a total of 4,083) are used as test data.

3.1 Performance evaluation

All experimental results are reported in Table 1. A wide range of performance measures are reported, including AP, AUC, sensitivity, specificity, misclassification rate, and the Matthews correlation coefficient (MCC; [14]).

As we have argued earlier (Section 1), misclassification error itself is not the most appropriate criterion for detection problems. Likewise, it is meaningless to look at sensitivity and specificity separately, as improving one always deteriorates the other. These measures are reported for completeness. Though MCC considers specificity and sensitivity together, the AUC and the AP do so in a much more comprehensive, balanced and sophisticated manner because all possible decision thresholds are considered; see [16] and Appendix A.

We also observe that AUC and AP are quite consistent measures of ranking quality, while other measures such as MCC, misclassification error, sensitivity and specificity, which are based on a specific (and often arbitrary) decision threshold, are not as informative for evaluating detection algorithms.

3.2 Benchmark algorithms

We use two well-known algorithms as benchmarks: CART [3] and Random Forest (RF) [2]. 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` packages from R [18].

For `rpart`, we choose the best cross-validated tree. For `randomForest`, we use its default forest size of 500 trees. Each tree casts a vote towards the class membership, and the average votes across the entire forest is used to make the final classification/ranking decision. For TARGET, we

Table 1. Experimental results. See Section 3.3 for explanations of "CART-A" versus "CART-B" and "RF-A" versus "RF-B". See Section 3.4 for further explanations of the difference between how results from TARGET and those from CART and RF are reported

	AP	AUC	MCC	Specificity	Sensitivity	Misclassification	Tree size
Simulation 1 (15% class 1) – average of 5 repeated experiments							
TARGET	.304	.644	.206	.992	.086	.150	5
	(.008)	(.006)	(.016)	(.003)	(.019)	(.001)	(.4)
CART-A	.273	.590	.192	.983	.106	.154	8
CART-B	.267	.588	.137	.787	.372	.277	7
RF-A	.298	.654	.184	.988	.089	.152	—
RF-B	.283	.648	.151	.707	.476	.329	—
Simulation 2 (20% class 1) – average of 5 repeated experiments							
TARGET	.486	.751	.347	.960	.287	.173	10
	(.012)	(.017)	(.021)	(.012)	(.047)	(.001)	(2.3)
CART-A	.441	.688	.327	.959	.272	.178	13
CART-B	.399	.649	.255	.833	.423	.248	9
RF-A	.545	.801	.289	.988	.154	.178	—
RF-B	.497	.781	.345	.753	.651	.267	—
Breast Cancer Data (35% cancer) – average of 5 repeated experiments							
TARGET	.958	.972	.859	.947	.912	.066	5
	(.012)	(.007)	(.015)	(.007)	(.022)	(.002)	(.31)
CART-A	.901	.955	.887	.954	.940	.051	6
CART-B	.923	.957	.893	.960	.935	.049	6
RF-A	.985	.993	.935	.973	.967	.029	—
RF-B	.983	.993	.936	.965	.984	.029	—
Insurance Fraud Data (6% fraud) – training with 1994 data, testing with 1995 and 1996 data							
TARGET	.139	.770	.046	.998	.013	.056	12
	(.028)	(.020)	(.071)	(.001)	(.016)	(.002)	(4.2)
CART-A	.109	.660	.090	.989	.056	.063	27
CART-B	.102	.668	.109	.886	.270	.148	17
RF-A	.120	.746	.008	.999	.000	.057	—
RF-B	.127	.737	.168	.739	.591	.269	—
Insurance Fraud Data (6% fraud) – training with 1995 data, testing with 1996 data							
TARGET	.091	.692	.023	.979	.035	.070	15
	(.012)	(.074)	(.018)	(.012)	(.024)	(.010)	(2.3)
CART-A	.075	.494	.086	.987	.061	.062	24
CART-B	.069	.495	.019	.759	.277	.265	18
RF-A	.081	.652	N/A	1.000	.000	.052	—
RF-B	.070	.600	.025	.754	.296	.270	—

evolve 500 generations each time and choose the penalty parameter, α, using cross validation (see Section 2.4). For `rpart` and TARGET, we also set a minimal node size of 5 observations for all experiments.

3.3 Unequal costs

For rare and unbalanced classification problems, it is common to use unequal costs for the two classes when fitting conventional classifiers. We use two different cost schemes to run `rpart` and `randomForest`. Scheme A assigns equal costs to false positives and false negatives, while Scheme B assigns unequal costs, taken to be inversely proportional to the observed class proportions. For example, if the class ratio is 1:9 in the training data, the ratio for the two misclassification costs is set to be 9:1. This issue is irrelevant for TARGET since it aims to optimize the AP.

To compute performance metrics that depend on making actual classifications, such as MCC, sensitivity, specificity, and misclassification error, the unequal costs are also used to determine the decision threshold.

3.4 Stability of TARGET

Recall that TARGET involves stochastic optimization, i.e., each time we run TARGET, we will obtain slightly different results on the same data set. To better understand its behavior, each experiment consisting of the same training-test pair is repeated 5 times, and the mean test-set result together with its standard deviation are recorded for each criterion. For the two simulated and the breast cancer examples, Table 1 reports the *average* of these mean results and standard deviations, averaged over 5 repetitions. We can see that the performance of TARGET is quite stable over multiple runs.

3.5 Results

Some of the main observations from Table 1 are as follows.

First, it is clear that TARGET generally performs well in terms of the AP. This is reassuring, but perhaps not too surprising, since TARGET aims to optimize the AP directly. What's surprising is that TARGET can almost always achieve this kind of performance with a relatively small tree! CART produces larger (and less effective) trees, whereas RF uses a large number of trees and must pay a big price in terms of model interpretability.

Second, we see that TARGET also performs well in terms of the AUC, another measure of ranking quality. This is particularly satisfying. Since none of the algorithms used the AUC to fit their models, the AUC can be considered a more objective performance evaluation metric than the AP.

Third, the advantage of TARGET appears to increase as the problem becomes more unbalanced, e.g., the insurance fraud data (6% fraud) versus the breast cancer data (35% cancer).

Finally, it does *not* appear that assigning unequal misclassification costs can make CART or RF better detection/ranking algorithms.

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 $f(\mathbf{x};\boldsymbol{\theta})$ 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 $\boldsymbol{\theta}$. Experimental results using TARGET (which uses the genetic algorithm to build decision trees that maximize the AP directly) support this general argument. For future research, we will consider using TARGET to optimize other interesting, non-smooth objective functions such as the AUC, building ensemble models using TARGET, and optimizing the AP and/or AUC when $f(\mathbf{x};\boldsymbol{\theta})$ is not restricted to be trees.

APPENDIX A. 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 3 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., 4, 11]. In practice, $h(t)$ and hence $r(t)$ and $p(t)$ all take values only at a finite number of points $t_i = i/n$, $i = 0, 1, 2, ..., n$. The formula for calculating the average precision is as follows:

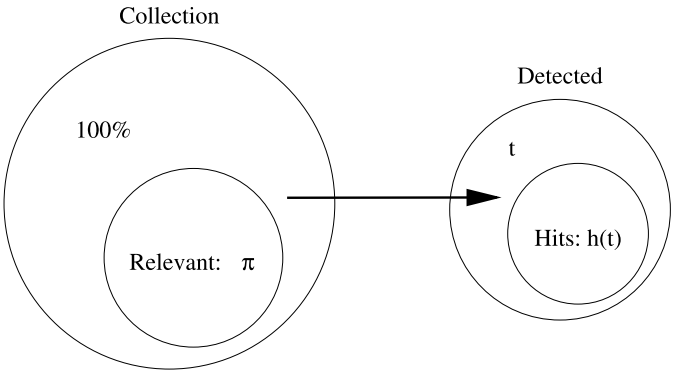

Figure 3. Illustration of a typical detection operation. A small fraction π of the entire collection $\mathcal{C}$ is relevant. An algorithm "detects" a fraction t from $\mathcal{C}$ and $h(t) \leq t$ is truly relevant. Source: Zhu et al. [21].

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

where $\Delta r(t_i) = r(t_i) - r(t_{i-1})$, and $h(t_0) = r(t_0) = 0$ by definition.

Here is a concrete example. Table 2 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

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

and

$$\text{AP(B)} = \sum_{i=1}^{5} p(t_i)\Delta r(t_i) = \left(\frac{1}{1} + \frac{2}{4} + \frac{3}{5}\right) \times \frac{1}{3} = 0.70,$$

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%.

Table 2. *Three hypothetical algorithms A, B and P*

	Algorithm A			Algorithm B			Algorithm P		
Ranked item (i)	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

A.1 Handling of ties

The case of ties (i.e., several items receive the same rank score) deserves special mention, because this happens quite often for tree-based algorithms. If all tied items have the same class label, this does not matter. Most often, however, some of the tied items will be "hits" and some will be "misses". It is not hard to see from the definition of AP above that, if we always put "hits" ahead of "misses" among the tied items, we will bias the AP upwards, and that, if we always put "misses" ahead of "hits", we will bias the AP downwards. Therefore, the correct way to break ties is to average over all possible permutations of the tied items. This can be done analytically; see, e.g., the `avgp` function from the `lagp` package in `R`.

ACKNOWLEDGMENTS

This research is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. We also thank the editor and two anonymous referees for their very useful comments, which have greatly improved our paper.

Received 4 December 2009

REFERENCES

[1] Bolton, R. J. and Hand, D. J. (2002). Statistical fraud detection: A review. *Statistical Science*, **17**(3), 235–255. MR1963313

[2] Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

[3] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth. MR0726392

[4] Buckland, M. and Gey, F. (1994). The relationship between recall and precision. *Journal of the American Society for Information Science*, **45**(1), 12–19.

[5] Cortes, C. and Mohri, M. (2004). AUC optimization vs. error rate minimization. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

[6] Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

[7] Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the Society for Information Science*, **41**(6), 391–407.

[8] Dumais, S. T. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, **23**(2), 229–236.

[9] Fan, G. and Gray, J. B. (2005). Regression tree analysis using TARGET. *Journal of Computational and Graphical Statistics*, **14**(3), 206–218. MR2137898

[10] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

[11] Gordon, M. and Kochen, M. (1989). Recall-precision trade-off: A derivation. *Journal of the American Society for Information Science*, **40**(3), 145–151.

[12] Gray, J. B. and Fan, G. (2008). Classification tree analysis using TARGET. *Computational Statistics and Data Analysis*, **52**(3), 1362–1372. MR2422741

[13] Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2001). *The Elements of Statistical Learning: Data-mining, Inference and Prediction*. Springer-Verlag. MR1851606

[14] Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta*, **405**, 442–451.

[15] Peng, F., Schuurmans, D., and Wang, S. (2003). Augmenting naïve Bayes classifiers with statistical language models. *Information Retrieval*, **7**(3), 317–345.

[16] Pepe, M. S. (2003). *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford University Press. MR2260483

[17] Pyle, D. (1999). *Data Preparation for Data Mining*. Morgan Kaufmann.

[18] R Development Core Team (2008). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0; URL http://www.R-project.org.

[19] Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press. MR1438788

[20] Weiss, G. M. and Hirsh, H. (1998). Learning to predict rare events in categorical time-series data. In *Papers from the 1998 AAAI Workshop, Predicting the Future: AI Approaches to Time-Series Problems*, pages 83–90. AAAI Press.

[21] Zhu, M., Su, W., and Chipman, H. A. (2006). LAGO: A computationally efficient approach for statistical detection. *Technometrics*, **48**, 193–205. MR2277674

Guangzhe Fan
Department of Statistics and Actuarial Science
University of Waterloo
200 University Avenue West
Waterloo, Ontario N2L 3G1
Canada
E-mail address: gfan@uwaterloo.ca

Mu Zhu
Department of Statistics and Actuarial Science
University of Waterloo
200 University Avenue West
Waterloo, Ontario N2L 3G1
Canada
E-mail address: m3zhu@uwaterloo.ca