Available online at www.sciencedirect.com
SCIENCE DIRECT®
Pattern Recognition Letters 27 (2006) 861–874
Pattern Recognition Letters
www.elsevier.com/locate/patrec

An introduction to ROC analysis

Tom Fawcett

Institute for the Study of Learning and Expertise, 2164 Staunton Court, Palo Alto, CA 94306, USA

Available online 19 December 2005

Abstract

Receiver operating characteristics (ROC) graphs are useful for organizing classifiers and visualizing their performance. ROC graphs are commonly used in medical decision making, and in recent years have been used increasingly in machine learning and data mining research. Although ROC graphs are apparently simple, there are some common misconceptions and pitfalls when using them in practice. The purpose of this article is to serve as an introduction to ROC graphs and as a guide for using them in research.

Keywords: ROC analysis; Classifier evaluation; Evaluation metrics

1. Introduction

A receiver operating characteristics (ROC) graph is a technique for visualizing, organizing and selecting classifiers based on their performance. ROC graphs have long been used in signal detection theory to depict the tradeoff between hit rates and false alarm rates of classifiers (Egan, 1975; Swets et al., 2000). ROC analysis has been extended for use in visualizing and analyzing the behavior of diagnostic systems (Swets, 1988). The medical decision making community has an extensive literature on the use of ROC graphs for diagnostic testing (Zou, 2002). Swets et al. (2000) brought ROC curves to the attention of the wider public with their *Scientific American* article.

One of the earliest adopters of ROC graphs in machine learning was Spackman (1989), who demonstrated the value of ROC curves in evaluating and comparing algorithms. Recent years have seen an increase in the use of ROC graphs in the machine learning community, due in part to the realization that simple classification accuracy is often a poor metric for measuring performance (Provost and Fawcett, 1997; Provost et al., 1998). In addition to being a generally useful performance graphing method, they have properties that make them especially useful for domains with skewed class distribution and unequal classification error costs. These characteristics have become increasingly important as research continues into the areas of cost-sensitive learning and learning in the presence of unbalanced classes.

ROC graphs are conceptually simple, but there are some non-obvious complexities that arise when they are used in research. There are also common misconceptions and pitfalls when using them in practice. This article attempts to serve as a basic introduction to ROC graphs and as a guide for using them in research. The goal of this article is to advance general knowledge about ROC graphs so as to promote better evaluation practices in the field.

2. Classifier performance

We begin by considering classification problems using only two classes. Formally, each instance I is mapped to one element of the set $\{\boldsymbol{p}, \boldsymbol{n}\}$ of positive and negative class labels. A *classification model* (or *classifier*) is a mapping from instances to predicted classes. Some classification models produce a continuous output (e.g., an estimate of an instance's class membership probability) to which different thresholds may be applied to predict class membership. Other models produce a discrete class label indicating only the predicted class of the instance. To distinguish between

E-mail addresses: tfawcett@acm.org, tom.fawcett@gmail.com

0167-8655/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2005.10.010

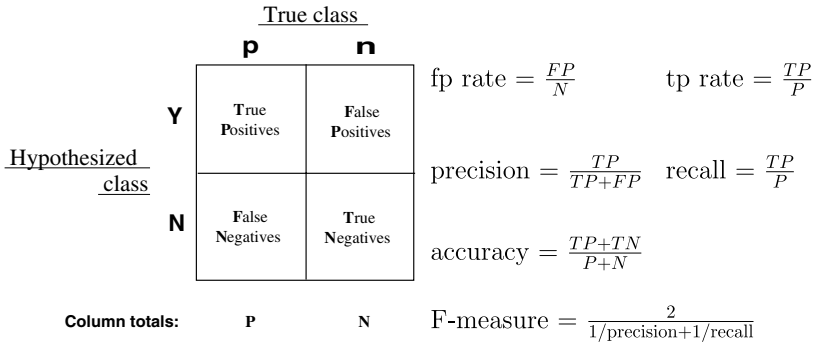


Fig. 1. Confusion matrix and common performance metrics calculated from it.

the actual class and the predicted class we use the labels $\{Y, N\}$ for the class predictions produced by a model.

Given a classifier and an instance, there are four possible outcomes. If the instance is positive and it is classified as positive, it is counted as a *true positive*; if it is classified as negative, it is counted as a *false negative*. If the instance is negative and it is classified as negative, it is counted as a *true negative*; if it is classified as positive, it is counted as a *false positive*. Given a classifier and a set of instances (the test set), a two-by-two *confusion matrix* (also called a contingency table) can be constructed representing the dispositions of the set of instances. This matrix forms the basis for many common metrics.

Fig. 1 shows a confusion matrix and equations of several common metrics that can be calculated from it. The numbers along the major diagonal represent the correct decisions made, and the numbers of this diagonal represent the errors—the confusion—between the various classes. The **true positive rate**[1] (also called *hit rate* and *recall*) of a classifier is estimated as

$$tp\ rate \approx \frac{\text{Positives correctly classified}}{\text{Total positives}}$$

The **false positive rate** (also called *false alarm rate*) of the classifier is

$$fp\ rate \approx \frac{\text{Negatives incorrectly classified}}{\text{Total negatives}}$$

Additional terms associated with ROC curves are

$$\text{sensitivity} = \text{recall}$$

$$\text{specificity} = \frac{\text{True negatives}}{\text{False positives} + \text{True negatives}} = 1 - fp\ rate$$

$$\text{positive predictive value} = \text{precision}$$

[1] For clarity, counts such as TP and FP will be denoted with upper-case letters and rates such as *tp rate* will be denoted with lower-case.

3. ROC space

ROC graphs are two-dimensional graphs in which *tp rate* is plotted on the *Y* axis and *fp rate* is plotted on the *X* axis. An ROC graph depicts relative tradeoffs between benefits (true positives) and costs (false positives). Fig. 2 shows an ROC graph with five classifiers labeled A through E.

A *discrete* classifier is one that outputs only a class label. Each discrete classifier produces an $(fp\ rate, tp\ rate)$ pair corresponding to a single point in ROC space. The classifiers in Fig. 2 are all discrete classifiers.

Several points in ROC space are important to note. The lower left point $(0, 0)$ represents the strategy of never issuing a positive classification; such a classifier commits no false positive errors but also gains no true positives. The opposite strategy, of unconditionally issuing positive classifications, is represented by the upper right point $(1, 1)$.

The point $(0, 1)$ represents perfect classification. D's performance is perfect as shown.

Informally, one point in ROC space is better than another if it is to the northwest (*tp rate* is higher, *fp rate* is lower, or both) of the first. Classifiers appearing on the left-hand side of an ROC graph, near the *X* axis, may be

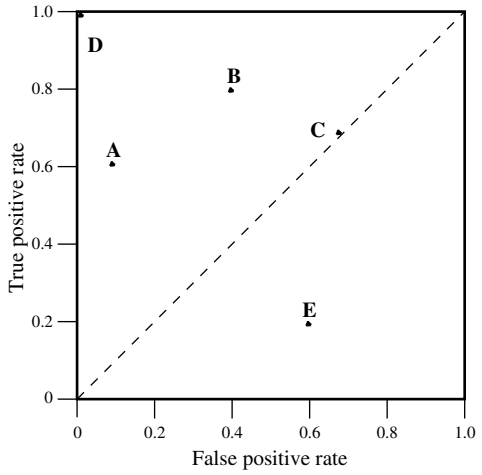


Fig. 2. A basic ROC graph showing five discrete classifiers.

thought of as "conservative": they make positive classifications only with strong evidence so they make few false positive errors, but they often have low true positive rates as well. Classifiers on the upper right-hand side of an ROC graph may be thought of as "liberal": they make positive classifications with weak evidence so they classify nearly all positives correctly, but they often have high false positive rates. In Fig. 2, A is more conservative than B. Many real world domains are dominated by large numbers of negative instances, so performance in the far left-hand side of the ROC graph becomes more interesting.

3.1. Random performance

The diagonal line $y = x$ represents the strategy of randomly guessing a class. For example, if a classifier randomly guesses the positive class half the time, it can be expected to get half the positives and half the negatives correct; this yields the point $(0.5, 0.5)$ in ROC space. If it guesses the positive class 90% of the time, it can be expected to get 90% of the positives correct but its false positive rate will increase to 90% as well, yielding $(0.9, 0.9)$ in ROC space. Thus a random classifier will produce a ROC point that "slides" back and forth on the diagonal based on the frequency with which it guesses the positive class. In order to get away from this diagonal into the upper triangular region, the classifier must exploit some information in the data. In Fig. 2, C's performance is virtually random. At $(0.7, 0.7)$, C may be said to be guessing the positive class 70% of the time.

Any classifier that appears in the lower right triangle performs worse than random guessing. This triangle is therefore usually empty in ROC graphs. If we negate a classifier—that is, reverse its classification decisions on every instance—its true positive classifications become false negative mistakes, and its false positives become true negatives. Therefore, any classifier that produces a point in the lower right triangle can be negated to produce a point in the upper left triangle. In Fig. 2, E performs much worse than random, and is in fact the negation of B. Any classifier on the diagonal may be said to have no information about the class. A classifier below the diagonal may be said to have useful information, but it is applying the information incorrectly (Flach and Wu, 2003).

Given an ROC graph in which a classifier's performance appears to be slightly better than random, it is natural to ask: "is this classifier's performance truly significant or is it only better than random by chance?" There is no conclusive test for this, but Forman (2002) has shown a methodology that addresses this question with ROC curves.

4. Curves in ROC space

Many classifiers, such as decision trees or rule sets, are designed to produce only a class decision, i.e., a ***Y*** or ***N*** on each instance. When such a discrete classifier is applied to a test set, it yields a single confusion matrix, which in turn corresponds to one ROC point. Thus, a discrete classifier produces only a single point in ROC space.

Some classifiers, such as a Naive Bayes classifier or a neural network, naturally yield an instance *probability* or *score*, a numeric value that represents the degree to which an instance is a member of a class. These values can be strict probabilities, in which case they adhere to standard theorems of probability; or they can be general, uncalibrated scores, in which case the only property that holds is that a higher score indicates a higher probability. We shall call both a *probabilistic* classifier, in spite of the fact that the output may not be a proper probability.[2]

Such a *ranking* or *scoring* classifier can be used with a threshold to produce a discrete (binary) classifier: if the classifier output is above the threshold, the classifier produces a ***Y***, else a ***N***. Each threshold value produces a different point in ROC space. Conceptually, we may imagine varying a threshold from $-\infty$ to $+\infty$ and tracing a curve through ROC space. Computationally, this is a poor way of generating an ROC curve, and the next section describes a more efficient and careful method.

Fig. 3 shows an example of an ROC "curve" on a test set of 20 instances. The instances, 10 positive and 10 negative, are shown in the table beside the graph. Any ROC curve generated from a finite set of instances is actually a step function, which approaches a true curve as the number of instances approaches infinity. The step function in Fig. 3 is taken from a very small instance set so that each point's derivation can be understood. In the table of Fig. 3, the instances are sorted by their scores, and each point in the ROC graph is labeled by the score threshold that produces it. A threshold of $+\infty$ produces the point $(0, 0)$. As we lower the threshold to 0.9 the first positive instance is classified positive, yielding $(0, 0.1)$. As the threshold is further reduced, the curve climbs up and to the right, ending up at $(1, 1)$ with a threshold of 0.1. Note that lowering this threshold corresponds to moving from the "conservative" to the "liberal" areas of the graph.

Although the test set is very small, we can make some tentative observations about the classifier. It appears to perform better in the more conservative region of the graph; the ROC point at $(0.1, 0.5)$ produces its highest accuracy (70%). This is equivalent to saying that the classifier is better at identifying likely positives than at identifying likely negatives. Note also that the classifier's best accuracy occurs at a threshold of $\geqslant 0.54$, rather than at $\geqslant 0.5$ as we might expect with a balanced distribution. The next section discusses this phenomenon.

4.1. Relative versus absolute scores

An important point about ROC graphs is that they measure the ability of a classifier to produce good *relative*

[2] Techniques exist for converting an uncalibrated score into a proper probability but this conversion is unnecessary for ROC curves.

Inst#	Class	Score	Inst#	Class	Score
1	**p**	.9	11	**p**	.4
2	**p**	.8	12	**n**	.39
3	**n**	.7	13	**p**	.38
4	**p**	.6	14	**n**	.37
5	**p**	.55	15	**n**	.36
6	**p**	.54	16	**n**	.35
7	**n**	.53	17	**p**	.34
8	**n**	.52	18	**n**	.33
9	**p**	.51	19	**p**	.30
10	**n**	.505	20	**n**	.1

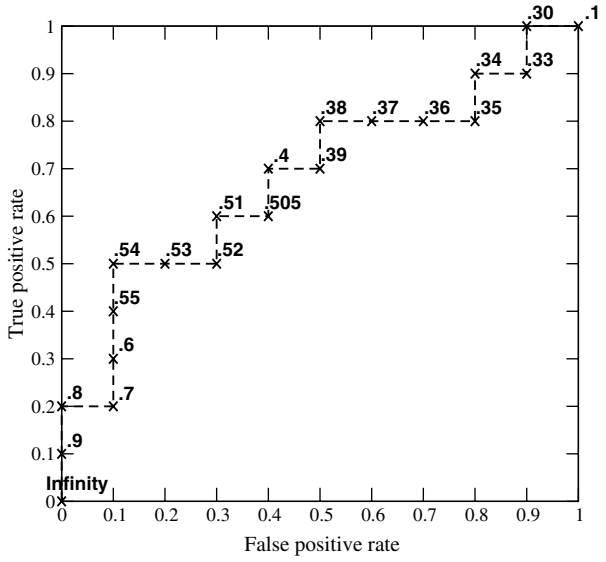


Fig. 3. The ROC "curve" created by thresholding a test set. The table shows 20 data and the score assigned to each by a scoring classifier. The graph shows the corresponding ROC curve with each point labeled by the threshold that produces it.

instance scores. A classifier need not produce accurate, calibrated probability estimates; it need only produce relative accurate scores that serve to discriminate positive and negative instances.

Consider the simple instance scores shown in Fig. 4, which came from a Naive Bayes classifier. Comparing the hypothesized class (which is ***Y*** if score > 0.5, else ***N***) against the true classes, we can see that the classifier gets instances 7 and 8 wrong, yielding 80% accuracy. However, consider the ROC curve on the left side of the figure. The curve rises vertically from $(0,0)$ to $(0,1)$, then horizontally to $(1,1)$. This indicates perfect classification performance on this test set. Why is there a discrepancy?

The explanation lies in what each is measuring. The ROC curve shows the ability of the classifier to rank the positive instances relative to the negative instances, and it is indeed perfect in this ability. The accuracy metric imposes a threshold (score > 0.5) and measures the resulting classifications with respect to the scores. The accuracy measure would be appropriate if the scores were proper probabilities, but they are not. Another way of saying this is that the scores are not *properly calibrated*, as true probabilities are. In ROC space, the imposition of a 0.5 threshold results in the performance designated by the circled "accuracy point" in Fig. 4. This operating point is suboptimal. We could use the training set to estimate a prior for $p(\boldsymbol{p}) = 6/10 = 0.6$ and use this as a threshold, but it would still produce suboptimal performance (90% accuracy).

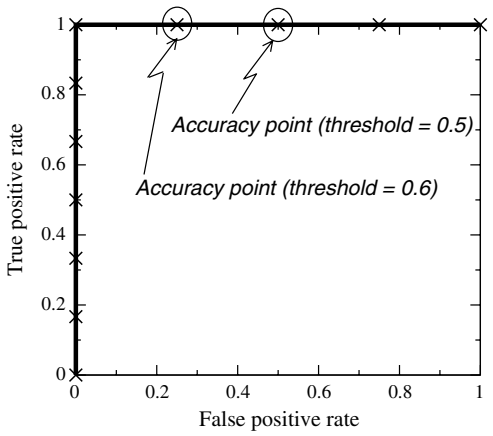
One way to eliminate this phenomenon is to calibrate the classifier scores. There are some methods for doing this (Zadrozny and Elkan, 2001). Another approach is to use an ROC method that chooses operating points based on their relative performance, and there are methods for doing this as well (Provost and Fawcett, 1998, 2001). These latter methods are discussed briefly in Section 6.

A consequence of relative scoring is that classifier scores should not be compared across model classes. One model class may be designed to produce scores in the range $[0,1]$ while another produces scores in $[-1,+1]$ or $[1,100]$. Comparing model performance at a common threshold will be meaningless.

4.2. Class skew

ROC curves have an attractive property: they are insensitive to changes in class distribution. If the proportion of positive to negative instances changes in a test set, the ROC curves will not change. To see why this is so, consider the confusion matrix in Fig. 1. Note that the class distribution—the proportion of positive to negative instances—is the relationship of the left (+) column to the right (−) column. Any performance metric that uses values from both columns will be inherently sensitive to class skews. Metrics such as accuracy, precision, lift and F score use values from both columns of the confusion matrix. As a class distribution changes these measures will change as well, even if the fundamental classifier performance does not. ROC graphs are based upon *tp rate* and *fp rate*, in which each dimension is a strict columnar ratio, so do not depend on class distributions.

To some researchers, large class skews and large changes in class distributions may seem contrived and unrealistic. However, class skews of 10^1 and 10^2 are very common in real world domains, and skews up to 10^6 have been observed in some domains (Clearwater and Stern, 1991; Fawcett and Provost, 1996; Kubat et al., 1998; Saitta and Neri, 1998). Substantial changes in class distributions are not unrealistic either. For example, in medical decision making epidemics may cause the incidence of a disease to increase over time. In fraud detection, proportions of fraud varied significantly from month to month and place to place (Fawcett and Provost, 1997). Changes in a manufacturing practice may cause the proportion of defective units



Inst no.	Class True	Class Hyp	Score
1	p	Y	0.99999
2	p	Y	0.99999
3	p	Y	0.99993
4	p	Y	0.99986
5	p	Y	0.99964
6	p	Y	0.99955
7	n	Y	0.68139
8	n	Y	0.50961
9	n	N	0.48880
10	n	N	0.44951

Fig. 4. Scores and classifications of 10 instances, and the resulting ROC curve.

produced by a manufacturing line to increase or decrease. In each of these examples the prevalence of a class may change drastically without altering the fundamental characteristic of the class, i.e., the target concept.

Precision and recall are common in information retrieval for evaluating retrieval (classification) performance (Lewis, 1990, 1991). Precision-recall graphs are commonly used where static document sets can sometimes be assumed; however, they are also used in dynamic environments such as web page retrieval, where the number of pages irrelevant to a query (N) is many orders of magnitude greater than P and probably increases steadily over time as web pages are created.

To see the effect of class skew, consider the curves in Fig. 5, which show two classifiers evaluated using ROC curves and precision-recall curves. In Fig. 5a and b, the test

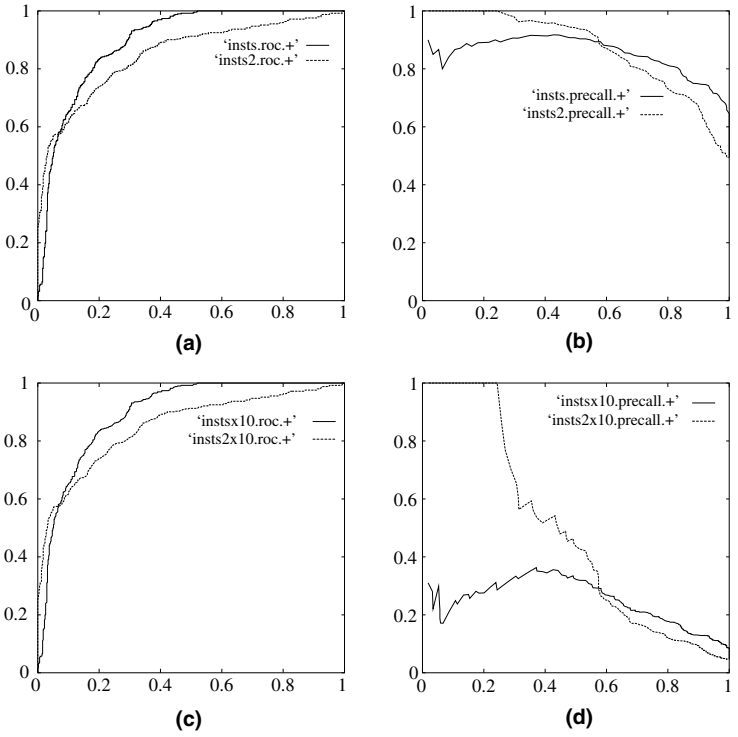


Fig. 5. ROC and precision-recall curves under class skew. (a) ROC curves, 1:1; (b) precision-recall curves, 1:1; (c) ROC curves, 1:10 and (d) precision-recall curves, 1:10.

set has a balanced 1:1 class distribution. Graph 5c and d shows the same two classifiers on the same domain, but the number of negative instances has been increased 10-fold. Note that the classifiers and the underlying concept has not changed; only the class distribution is different. Observe that the ROC graphs in Fig. 5a and c are identical, while the precision-recall graphs in Fig. 5b and d differ substantially. In some cases, the conclusion of which classifier has superior performance can change with a shifted distribution.

4.3. Creating scoring classifiers

Many classifier models are discrete: they are designed to produce only a class label from each test instance. However, we often want to generate a full ROC curve from a classifier instead of just a single point. To this end we want to generate scores from a classifier rather than just a class label. There are several ways of producing such scores.

Many discrete classifier models may easily be converted to scoring classifiers by "looking inside" them at the instance statistics they keep. For example, a decision tree determines a class label of a leaf node from the proportion of instances at the node; the class decision is simply the most prevalent class. These class proportions may serve as a score (Provost and Domingos, 2001). A rule learner keeps similar statistics on rule confidence, and the confidence of a rule matching an instance can be used as a score (Fawcett, 2001).

Even if a classifier only produces a class label, an aggregation of them may be used to generate a score. MetaCost (Domingos, 1999) employs bagging to generate an ensemble of discrete classifiers, each of which produces a vote. The set of votes could be used to generate a score.[3]

Finally, some combination of scoring and voting can be employed. For example, rules can provide basic probability estimates, which may then be used in weighted voting (Fawcett, 2001).

5. Efficient generation of ROC curves

Given a test set, we often want to generate an ROC curve efficiently from it. We can exploit the monotonicity of thresholded classifications: any instance that is classified positive with respect to a given threshold will be classified positive for all lower thresholds as well. Therefore, we can simply sort the test instances decreasing by f scores and move down the list, processing one instance at a time and updating TP and FP as we go. In this way an ROC graph can be created from a linear scan.

The algorithm is shown in Algorithm 1. TP and FP both start at zero. For each positive instance we increment TP and for every negative instance we increment FP. We maintain a stack R of ROC points, pushing a new point onto R after each instance is processed. The final output is the stack R, which will contain points on the ROC curve.

Let n be the number of points in the test set. This algorithm requires an $O(n \log n)$ sort followed by an $O(n)$ scan down the list, resulting in $O(n \log n)$ total complexity.

Statements 7–10 need some explanation. These are necessary in order to correctly handle sequences of equally scored instances. Consider the ROC curve shown in Fig. 6. Assume we have a test set in which there is a sequence of instances, four negatives and six positives, all scored equally by f. The sort in line 1 of Algorithm 1 does not impose any specific ordering on these instances since their f scores are equal. What happens when we create an ROC curve? In one extreme case, all the positives end up at the beginning of the sequence and we generate the "optimistic" upper L segment shown in Fig. 6. In the opposite

Algorithm 1. Efficient method for generating ROC points
Inputs: L, the set of test examples; $f(i)$, the probabilistic classifier's estimate that example i is positive; P and N, the number of positive and negative examples.
Outputs: R, a list of ROC points increasing by *fp rate*.
Require: $P > 0$ and $N > 0$

1: $L_{\text{sorted}} \leftarrow L$ sorted decreasing by f scores
2: $FP \leftarrow TP \leftarrow 0$
3: $R \leftarrow \langle\rangle$
4: $f_{\text{prev}} \leftarrow -\infty$
5: $i \leftarrow 1$
6: **while** $i \leqslant |L_{\text{sorted}}|$ **do**
7: **if** $f(i) \neq f_{\text{prev}}$ **then**
8: push $\left(\frac{FP}{N}, \frac{TP}{P}\right)$ onto R
9: $f_{\text{prev}} \leftarrow f(i)$
10: **end if**
11: **if** $L_{\text{sorted}}[i]$ is a positive example **then**
12: $TP \leftarrow TP + 1$
13: **else** /* i is a negative example */
14: $FP \leftarrow FP + 1$
15: **end if**
16: $i \leftarrow i + 1$
17: **end while**
18: push $\left(\frac{FP}{N}, \frac{TP}{P}\right)$ onto R /* This is (1,1) */
19: **end**

[3] MetaCost actually works in the opposite direction because its goal is to generate a discrete classifier. It first creates a probabilistic classifier, then applies knowledge of the error costs and class skews to relabel the instances so as to "optimize" their classifications. Finally, it learns a specific discrete classifier from this new instance set. Thus, MetaCost is not a good method for creating a scoring classifier, though its bagging method may be.

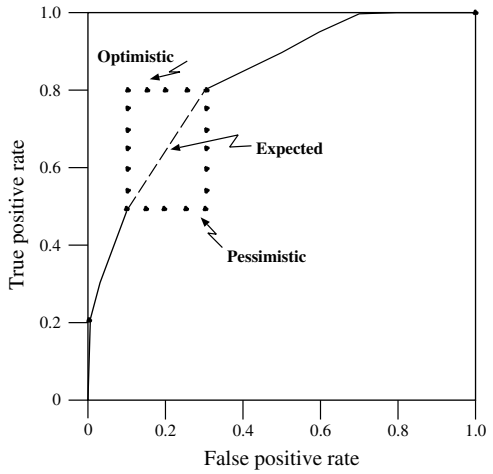


Fig. 6. The optimistic, pessimistic and expected ROC segments resulting from a sequence of 10 equally scored instances.

extreme, all the negatives end up at the beginning of the sequence and we get the "pessimistic" lower L shown in Fig. 6. Any mixed ordering of the instances will give a different set of step segments within the rectangle formed by these two extremes. However, the ROC curve should represent the *expected* performance of the classifier, which, lacking any other information, is the average of the pessimistic and optimistic segments. This average is the diagonal of the rectangle, and can be created in the ROC curve algorithm by not emitting an ROC point until all instances of equal f values have been processed. This is what the f_{prev} variable and the **if** statement of line 7 accomplish.

Instances that are scored equally may seem unusual but with some classifier models they are common. For example, if we use instance counts at nodes in a decision tree to score instances, a large, high-entropy leaf node may produce many equally scored instances of both classes. If such instances are not averaged, the resulting ROC curves will be sensitive to the test set ordering, and different orderings can yield very misleading curves. This can be especially critical in calculating the area under an ROC curve, discussed in Section 7. Consider a decision tree containing a leaf node accounting for n positives and m negatives. Every instance that is classified to this leaf node will be assigned the same score. The rectangle of Fig. 6 will be of size $\frac{nm}{PN}$, and if these instances are not averaged this one leaf may account for errors in ROC curve area as high as $\frac{nm}{2PN}$.

6. The ROC convex hull

One advantage of ROC graphs is that they enable visualizing and organizing classifier performance without regard to class distributions or error costs. This ability becomes very important when investigating learning with skewed distributions or cost-sensitive learning. A researcher can graph the performance of a set of classifiers, and that graph will remain invariant with respect to the operating conditions (class skew and error costs). As these conditions change, the region of interest may change, but the graph itself will not.

Provost and Fawcett (1998, 2001) show that a set of operating conditions may be transformed easily into a so-called *iso-performance line* in ROC space. Two points in ROC space, (FP_1, TP_1) and (FP_2, TP_2), have the same performance if

$$\frac{TP_2 - TP_1}{FP_2 - FP_1} = \frac{c(\boldsymbol{Y}, \boldsymbol{n})p(\boldsymbol{n})}{c(\boldsymbol{N}, \boldsymbol{p})p(\boldsymbol{p})} = m \tag{1}$$

This equation defines the slope of an iso-performance line. All classifiers corresponding to points on a line of slope m have the same expected cost. Each set of class and cost distributions defines a family of iso-performance lines. Lines "more northwest" (having a larger TP-intercept) are better because they correspond to classifiers with lower expected cost. More generally, a classifier is potentially optimal if and only if it lies on the convex hull of the set of points in ROC space. The convex hull of the set of points in ROC space is called the *ROC convex hull* (ROCCH) of the corresponding set of classifiers.

Fig. 7a shows four ROC curves (A through D) and their convex hull (labeled CH). D is not on the convex hull and is clearly sub-optimal. B is also not optimal for any conditions because it is not on the convex hull either. The convex hull is bounded only by points from curves A and C. Thus, if we are seeking optimal classification performance, classifiers B and D may be removed entirely from consideration. In addition, we may remove any discrete points from A and C that are not on the convex hull.

Fig. 7b shows the A and C curves again with two explicit iso-performance lines, α and β. Consider a scenario in which negatives outnumber positives by 10 to 1, but false positives and false negatives have equal cost. By Eq. (1) $m = 10$, and the most northwest line of slope $m = 10$ is α, tangent to classifier A, which would be the best performing classifier for these conditions.

Consider another scenario in which the positive and negative example populations are evenly balanced but a false negative is 10 times as expensive as a false positive. By Eq. (1) $m = 1/10$. The most northwest line of slope 1/10 would be line β, tangent to classifier C. C is the optimal classifier for these conditions.

If we wanted to generate a classifier somewhere on the convex hull between A and C, we could interpolate between the two. Section 10 explains how to generate such a classifier.

This ROCCH formulation has a number of useful implications. Since only the classifiers on the convex hull are potentially optimal, no others need be retained. The operating conditions of the classifier may be translated into an iso-performance line, which in turn may be used to identify a portion of the ROCCH. As conditions change, the hull itself does not change; only the portion of interest will.

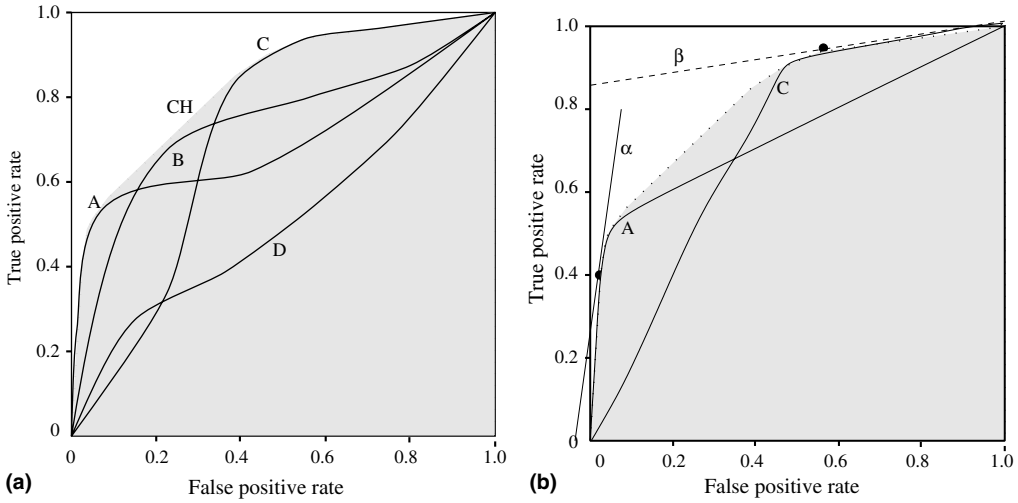


Fig. 7. (a) The ROC convex hull identifies potentially optimal classifiers. (b) Lines α and β show the optimal classifier under different sets of conditions.

7. Area under an ROC curve (AUC)

An ROC curve is a two-dimensional depiction of classifier performance. To compare classifiers we may want to reduce ROC performance to a single scalar value representing expected performance. A common method is to calculate the area under the ROC curve, abbreviated **AUC** (Bradley, 1997; Hanley and McNeil, 1982). Since the AUC is a portion of the area of the unit square, its value will always be between 0 and 1.0. However, because random guessing produces the diagonal line between $(0,0)$ and $(1,1)$, which has an area of 0.5, no realistic classifier should have an AUC less than 0.5.

The AUC has an important statistical property: the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. This is equivalent to the Wilcoxon test of ranks (Hanley and McNeil, 1982). The AUC is also closely related to the Gini coefficient (Breiman et al., 1984), which is twice the area between the diagonal and the ROC curve. Hand and Till (2001) point out that $\text{Gini} + 1 = 2 \times \text{AUC}$.

Fig. 8a shows the areas under two ROC curves, A and B. Classifier B has greater area and therefore better average performance. Fig. 8b shows the area under the curve of a binary classifier A and a scoring classifier B. Classifier A represents the performance of B when B is used with a single, fixed threshold. Though the performance of the two is equal at the fixed point (A's threshold), A's performance becomes inferior to B further from this point.

It is possible for a high-AUC classifier to perform worse in a specific region of ROC space than a low-AUC classifier. Fig. 8a shows an example of this: classifier B is generally better than A except at $\text{FPrate} > 0.6$ where A has a

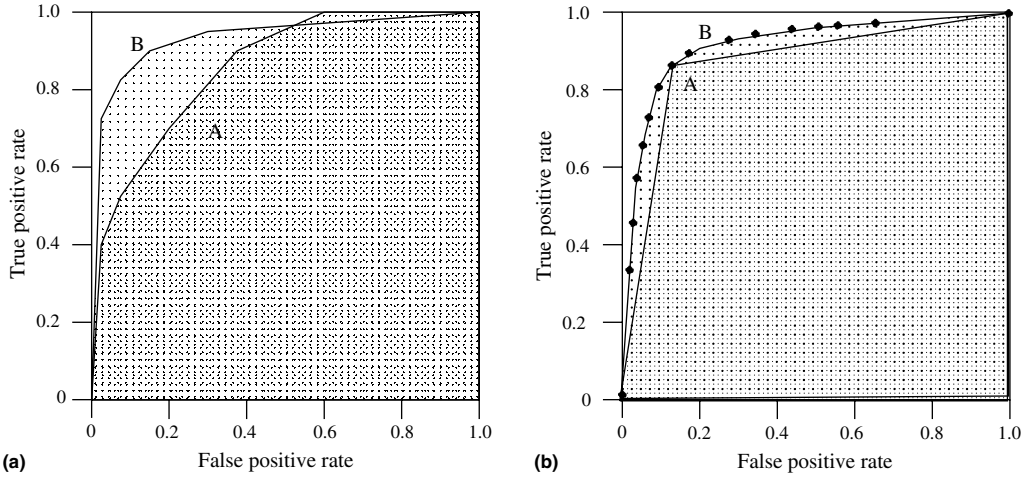


Fig. 8. Two ROC graphs. The graph on the left shows the area under two ROC curves. The graph on the right shows the area under the curves of a discrete classifier (A) and a probabilistic classifier (B).

Algorithm 2. Calculating the area under an ROC curve
Inputs: L, the set of test examples; $f(i)$, the probabilistic classifier's estimate that example i is positive; P and N, the number of positive and negative examples.
Outputs: A, the area under the ROC curve.
Require: $P > 0$ and $N > 0$

```
1: L_sorted ← L sorted decreasing by f scores
2: FP ← TP ← 0
3: FP_prev ← TP_prev ← 0
4: A ← 0
5: f_prev ← −∞
6: i ← 1
7: while i ⩽ |L_sorted| do
8:    if f(i) ≠ f_prev then
9:       A ← A + TRAPEZOID_AREA(FP, FP_prev,
             TP, TP_prev)
10:      f_prev ← f(i)
11:      FP_prev ← FP
12:      TP_prev ← TP
13:   end if
14:   if i is a positive example then
15:      TP ← TP + 1
16:   else      /* i is a negative example */
17:      FP ← FP + 1
18:   end if
19:   i ← i + 1
20: end while
21: A ← A + TRAPEZOID_AREA(N, FP_prev, N, TP_prev)
22: A ← A/(P × N)   /* scale from P × N onto the unit
    square */
23: end
1: function TRAPEZOID_AREA(X1, X2, Y1, Y2)
2: Base ← |X1 − X2|
3: Height_avg ← (Y1 + Y2)/2
4: return Base × Height_avg
5: end function
```

slight advantage. But in practice the AUC performs very well and is often used when a general measure of predictiveness is desired.

The AUC may be computed easily using a small modification of algorithm 1, shown in Algorithm 2. Instead of collecting ROC points, the algorithm adds successive areas of trapezoids to A. Trapezoids are used rather than rectangles in order to average the effect between points, as illustrated in Fig. 6. Finally, the algorithm divides A by the total possible area to scale the value to the unit square.

8. Averaging ROC curves

Although ROC curves may be used to evaluate classifiers, care should be taken when using them to make conclusions about classifier superiority. Some researchers have assumed that an ROC graph may be used to select the best classifiers simply by graphing them in ROC space and seeing which ones dominate. This is misleading; it is analogous to taking the maximum of a set of accuracy figures from a single test set. Without a measure of variance we cannot compare the classifiers.

Averaging ROC curves is easy if the original instances are available. Given test sets $T_1, T_2, \ldots, T_n$, generated from cross-validation or the bootstrap method, we can simply merge sort the instances together by their assigned scores into one large test set T_M. We then run an ROC curve generation algorithm such as algorithm 1 on T_M and plot the result. However, the primary reason for using multiple test sets is to derive a measure of variance, which this simple merging does not provide. We need a more sophisticated method that samples individual curves at different points and averages the samples.

ROC space is two-dimensional, and any average is necessarily one-dimensional. ROC curves can be projected onto a single dimension and averaged conventionally, but this leads to the question of whether the projection is appropriate, or more precisely, whether it preserves characteristics of interest. The answer depends upon the reason for averaging the curves. This section presents two methods for averaging ROC curves: vertical and threshold averaging.

Fig. 9a shows five ROC curves to be averaged. Each contains a thousand points and has some concavities. Fig. 9b shows the curve formed by merging the five test sets and computing their combined ROC curve. Fig. 9c and d shows average curves formed by sampling the five individual ROC curves. The error bars are 95% confidence intervals.

8.1. Vertical averaging

Vertical averaging takes vertical samples of the ROC curves for fixed FP rates and averages the corresponding TP rates. Such averaging is appropriate when the FP rate can indeed be fixed by the researcher, or when a single-dimensional measure of variation is desired. Provost et al. (1998) used this method in their work of averaging ROC curves of a classifier for k-fold cross-validation.

In this method each ROC curve is treated as a function, R_i, such that $tp\ rate = R_i(fp\ rate)$. This is done by choosing the maximum $tp\ rate$ for each $fp\ rate$ and interpolating between points when necessary. The averaged ROC curve is the function $\hat{R}(fp\ rate) = \text{mean}[R_i(fp\ rate)]$. To plot an average ROC curve we can sample from $\hat{R}$ at points regularly spaced along the $fp\ rate$-axis. Confidence intervals of the mean of $tp\ rate$ are computed using the common assumption of a binomial distribution.

Algorithm 3 computes this vertical average of a set of ROC points. It leaves the means in the array $TPavg$.

Several extensions have been left out of this algorithm for clarity. The algorithm may easily be extended to

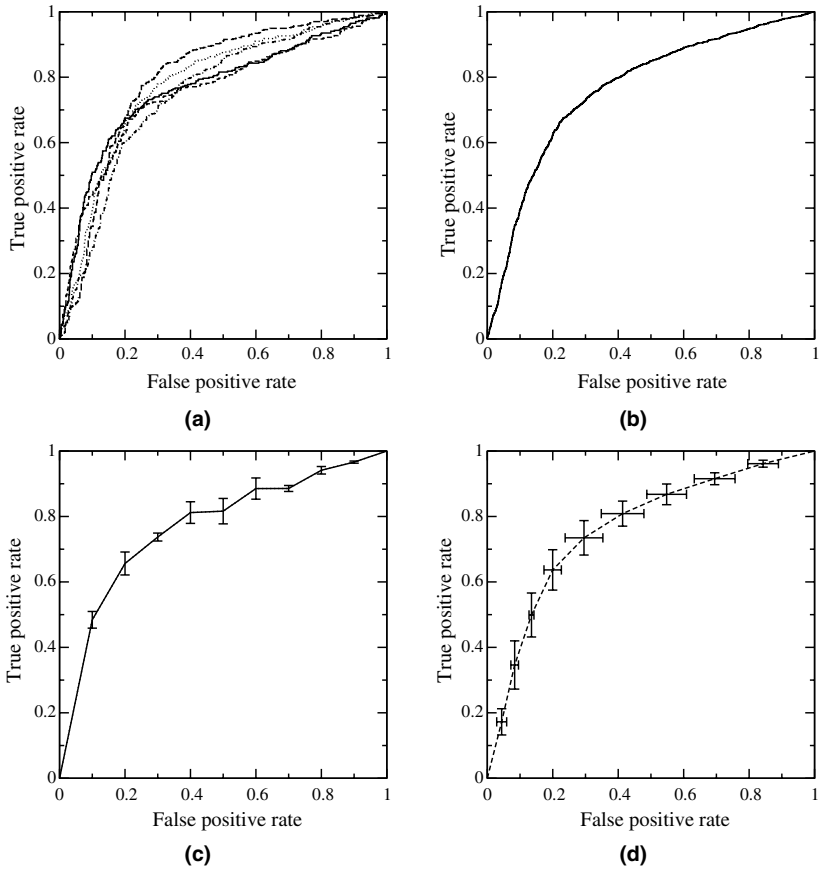


Fig. 9. ROC curve averaging. (a) ROC curves of five instance samples, (b) ROC curve formed by merging the five samples, (c) the curves of a averaged vertically and (d) the curves of a averaged by threshold.

compute standard deviations of the samples in order to draw confidence bars. Also, the function TP_FOR_FP may be optimized somewhat. Because it is only called on monotonically increasing values of *FP*, it need not scan each ROC array from the beginning every time; it could keep a record of the last point seen and initialize *i* from this array.

Fig. 9c shows the vertical average of the five curves in Fig. 9a. The vertical bars on the curve show the 95% confidence region of the ROC mean. For this average curve, the curves were sampled at FP rates from 0 through 1 by 0.1. It is possible to sample curves much more finely but the confidence bars may become difficult to read.

8.2. Threshold averaging

Vertical averaging has the advantage that averages are made of a single dependent variable, the true positive rate, which simplifies computing confidence intervals. However, Holte (2002) has pointed out that the independent variable, false positive rate, is often not under the direct control of the researcher. It may be preferable to average ROC points using an independent variable whose value can be controlled directly, such as the threshold on the classifier scores.

Threshold averaging accomplishes this. Instead of sampling points based on their positions in ROC space, as vertical averaging does, it samples based on the thresholds that produced these points. The method must generate a set of thresholds to sample, then for each threshold it finds the corresponding point of each ROC curve and averages them.

Algorithm 4 shows the basic method for doing this. It generates an array T of classifier scores which are sorted from largest to smallest and used as the set of thresholds. These thresholds are sampled at fixed intervals determined by *samples*, the number of samples desired. For a given threshold, the algorithm selects from each ROC curve the point of greatest score less than or equal to the threshold.[4] These points are then averaged separately along their X and Y axes, with the center point returned in the *Avg* array.

Fig. 9d shows the result of averaging the five curves of Fig. 9a by thresholds. The resulting curve has average points and confidence bars in the X and Y directions. The bars shown are at the 95% confidence level.

There are some minor limitations of threshold averaging with respect to vertical averaging. To perform threshold averaging we need the classifier score assigned to each point. Also, Section 4.1 pointed out that classifier scores

[4] We assume the ROC points have been generated by an algorithm like 1 that deals correctly with equally scored instances.

Algorithm 3. Vertical averaging of ROC curves
Inputs: *samples*, the number of FP samples; *nrocs*, the number of ROC curves to be sampled, *ROCS*[*nrocs*], an array of *nrocs* ROC curves; *npts*[*m*], the number of points in ROC curve *m*. Each ROC point is a structure of two members, the rates *fpr* and *tpr*.
Output: Array *tpravg*[*samples* + 1], containing the vertical averages.

```
1: s ← 1
2: for fpr_sample = 0 to 1 by 1/samples do
3:    tprsum ← 0
4:    for i = 1 to nrocs do
5:       tprsum ← tprsum + TPR_FOR_FPR(fpr_sample,
                 ROCS[i], npts[i])
6:    end for
7:    tpravg[s] ← tprsum/nrocs
8:    s ← s + 1
9: end for
10: end
1: function TPR_FOR_FPR(fpr_sample, ROC, npts)
2: i ← 1
3: while i < npts and ROC [i + 1].fpr ⩽ fpr_sample do
4:    i ← i + 1
5: end while
6: if ROC[i].fpr = fpr_sample then
7:    return ROC[i].tpr
8: else
9:    return INTERPOLATE(ROC[i], ROC [i + 1], fpr_sample)
10: end if
11: end function
1: function INTERPOLATE(ROCP1, ROCP2, X)
2: slope = (ROCP2.tpr − ROCP1.tpr)/(ROCP2.fpr −
          ROCP1.fpr)
3: return ROCP1.tpr + slope · (X − ROCP1.fpr)
4: end function
```

Algorithm 4. Threshold averaging of ROC curves
Inputs: *samples*, the number of threshold samples; *nrocs*, the number of ROC curves to be sampled; *ROCS*[*nrocs*], an array of *nrocs* ROC curves sorted by score; *npts*[*m*], the number of points in ROC curve *m*. Each ROC point is a structure of three members, *fpr*, *tpr* and score.
Output: *Avg*[*samples* + 1], an array of (*X*, *Y*) points constituting the average ROC curve.
Require: *samples* > 1

```
1: initialize array T to contain all scores of all ROC
   points
2: sort T in descending order
3: s ← 1
4: for tidx = 1 to length(T) by int(length(T)/samples) do
5:    fprsum ← 0
6:    tprsum ← 0
7:    for i = 1 to nrocs do
8:       p ← ROC_POINT_AT_THRESHOLD(ROCS[i], npts[i],
            T[tidx])
9:       fprsum ← fprsum + p.fpr
10:      tprsum ← tprsum + p.tpr
11:   end for
12:   Avg[s] ← (fprsum/nrocs, tprsum/nrocs)
13:   s ← s + 1
14: end for
15: end
1: function ROC_POINT_AT_THRESHOLD(ROC, npts, thresh)
2: i ← 1
3: while i ⩽ npts and ROC[i]. score > thresh do
4:    i ← i + 1
5: end while
6: return ROC[i]
7: end function
```

should not be compared across model classes. Because of this, ROC curves averaged from different model classes may be misleading because the scores may be incommensurate.

Finally, Macskassy and Provost (2004) have investigated different techniques for generating confidence bands for ROC curves. They investigate confidence intervals from vertical and threshold averaging, as well as three methods from the medical field for generating bands (simultaneous join confidence regions, Working-Hotelling based bands, and fixed-width confidence bands). The reader is referred to their paper for a much more detailed discussion of the techniques, their assumptions, and empirical studies.

9. Decision problems with more than two classes

Discussions up to this point have dealt with only two classes, and much of the ROC literature maintains this assumption. ROC analysis is commonly employed in medical decision making in which two-class diagnostic problems—presence or absence of an abnormal condition—are common. The two axes represent tradeoffs between errors (false positives) and benefits (true positives) that a classifier makes between two classes. Much of the analysis is straightforward because of the symmetry that exists in the two-class problem. The resulting performance can be graphed in two dimensions, which is easy to visualize.

9.1. Multi-class ROC graphs

With more than two classes the situation becomes much more complex if the entire space is to be managed. With n classes the confusion matrix becomes an $n \times n$ matrix containing the n correct classifications (the major diagonal entries) and $n^2 - n$ possible errors (the off-diagonal entries). Instead of managing trade-offs between *TP* and *FP*, we have n benefits and $n^2 - n$ errors. With only three classes, the surface becomes a $3^2 - 3 = 6$-dimensional polytope. Lane (2000) has outlined the issues involved and the prospects for addressing them. Srinivasan (1999) has shown

that the analysis behind the ROC convex hull extends to multiple classes and multi-dimensional convex hulls.

One method for handling n classes is to produce n different ROC graphs, one for each class. Call this the *class reference* formulation. Specifically, if C is the set of all classes, ROC graph i plots the classification performance using class c_i as the positive class and all other classes as the negative class, i.e.

$$P_i = c_i \tag{2}$$

$$N_i = \bigcup_{j \neq i} c_j \in C \tag{3}$$

While this is a convenient formulation, it compromises one of the attractions of ROC graphs, namely that they are insensitive to class skew (see Section 4.2). Because each N_i comprises the union of $n-1$ classes, changes in prevalence within these classes may alter the c_i's ROC graph. For example, assume that some class $c_k \in N$ is particularly easy to identify. A classifier for class c_i, $i \neq k$ may exploit some characteristic of c_k in order to produce low scores for c_k instances. Increasing the prevalence of c_k might alter the performance of the classifier, and would be tantamount to changing the target concept by increasing the prevalence of one of its disjuncts. This in turn would alter the ROC curve. However, with this caveat, this method can work well in practice and provide reasonable flexibility in evaluation.

9.2. Multi-class AUC

The AUC is a measure of the discriminability of a pair of classes. In a two-class problem, the AUC is a single scalar value, but a multi-class problem introduces the issue of combining multiple pairwise discriminability values. The reader is referred to Hand and Till's (2001) article for an excellent discussion of these issues.

One approach to calculating multi-class AUCs was taken by Provost and Domingos (2001) in their work on probability estimation trees. They calculated AUCs for multi-class problems by generating each class reference ROC curve in turn, measuring the area under the curve, then summing the AUCs weighted by the reference class's prevalence in the data. More precisely, they define

$$\mathrm{AUC}_{\mathrm{total}} = \sum_{c_i \in C} \mathrm{AUC}(c_i) \cdot p(c_i)$$

where $\mathrm{AUC}(c_i)$ is the area under the class reference ROC curve for c_i, as in Eq. (3). This definition requires only $|C|$ AUC calculations, so its overall complexity is $O(|C| n \log n)$.

The advantage of Provost and Domingos's AUC formulation is that $\mathrm{AUC}_{\mathrm{total}}$ is generated directly from class reference ROC curves, and these curves can be generated and visualized easily. The disadvantage is that the class reference ROC is sensitive to class distributions and error costs, so this formulation of $\mathrm{AUC}_{\mathrm{total}}$ is as well.

Hand and Till (2001) take a different approach in their derivation of a multi-class generalization of the AUC. They desired a measure that is insensitive to class distribution and error costs. The derivation is too detailed to summarize here, but it is based upon the fact that the AUC is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. From this probabilistic form, they derive a formulation that measures the unweighted *pairwise* discriminability of classes. Their measure, which they call M, is equivalent to:

$$\mathrm{AUC}_{\mathrm{total}} = \frac{2}{|C|(|C|-1)} \sum_{\{c_i, c_j\} \in C} \mathrm{AUC}(c_i, c_j)$$

where n is the number of classes and $\mathrm{AUC}(c_i, c_j)$ is the area under the two-class ROC curve involving classes c_i and c_j. The summation is calculated over all pairs of distinct classes, irrespective of order. There are $|C|(|C|-1)/2$ such pairs, so the time complexity of their measure is $O(|C|^2 n \log n)$. While Hand and Till's formulation is well justified and is insensitive to changes in class distribution, there is no easy way to visualize the surface whose area is being calculated.

10. Interpolating classifiers

Sometimes the performance desired of a classifier is not exactly produced by any available classifier, but lies between two available classifiers. The desired performance can be obtained by sampling the decisions of each classifier. The sampling ratio will determine where the resulting classification performance lies.

For a concrete example, consider the decision problem of the CoIL Challenge 2000 (van der Putten and Someren, 2000). In this challenge there is a set of 4000 clients to whom we wish to market a new insurance policy. Our budget dictates that we can afford to market to only 800 of them, so we want to select the 800 who are most likely to respond to the offer. The expected class prior of responders is 6%, so within the population of 4000 we expect to have 240 responders (positives) and 3760 non-responders (negatives).

Assume we have generated two classifiers, A and B, which score clients by the probability they will buy the policy. In ROC space A lies at $(0.1, 0.2)$ and B lies at $(0.25, 0.6)$, as shown in Fig. 10. We want to market to exactly 800 people so our solution constraint is $fp\ rate \times 3760 + tp\ rate \times 240 = 800$. If we use A we expect $0.1 \times 3760 + 0.2 \times 240 = 424$ candidates, which is too few. If we use B we expect $0.25 \times 3760 + 0.6 \times 240 = 1084$ candidates, which is too many. We want a classifier between A and B.

The solution constraint is shown as a dashed line in Fig. 10. It intersects the line between A and B at C, approximately $(0.18, 0.42)$. A classifier at point C would give the performance we desire and we can achieve it using linear interpolation. Calculate k as the proportional distance that C lies on the line between A and B:

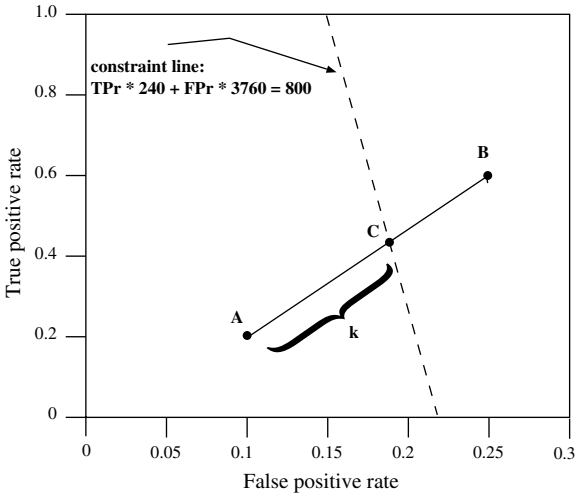


Fig. 10. Interpolating classifiers.

$$k = \frac{0.18 - 0.1}{0.25 - 0.1} \approx 0.53$$

Therefore, if we sample B's decisions at a rate of 0.53 and A's decisions at a rate of $1 - 0.53 = 0.47$ we should attain C's performance. In practice this fractional sampling can be done by randomly sampling decisions from each: for each instance, generate a random number between zero and one. If the random number is greater than k, apply classifier A to the instance and report its decision, else pass the instance to B.

11. Conclusion

ROC graphs are a very useful tool for visualizing and evaluating classifiers. They are able to provide a richer measure of classification performance than scalar measures such as accuracy, error rate or error cost. Because they decouple classifier performance from class skew and error costs, they have advantages over other evaluation measures such as precision-recall graphs and lift curves. However, as with any evaluation metric, using them wisely requires knowing their characteristics and limitations. It is hoped that this article advances the general knowledge about ROC graphs and helps to promote better evaluation practices in the pattern recognition community.

References

Bradley, A.P., 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recogn. 30 (7), 1145–1159.

Breiman, L., Friedman, J., Olshen, R., Stone, C., 1984. Classification and Regression Trees. Wadsworth International Group, Belmont, CA.

Clearwater, S., Stern, E., 1991. A rule-learning program in high energy physics event classification. Comput. Phys. Commun. 67, 159–182.

Domingos, P., 1999. MetaCost: A general method for making classifiers cost-sensitive. In: Proc. Fifth ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 155–164.

Egan, J.P., 1975. Signal detection theory and ROC analysis, Series in Cognition and Perception. Academic Press, New York.

Fawcett, T., 2001. Using rule sets to maximize ROC performance. In: Proc. IEEE Internat. Conf. on Data Mining (ICDM-2001), pp. 131–138.

Fawcett, T., Provost, F., 1996. Combining data mining and machine learning for effective user profiling. In: Simoudis, E., Han, J., Fayyad, U. (Eds.), Proc. Second Internat. Conf. on Knowledge Discovery and Data Mining. AAAI Press, Menlo Park, CA, pp. 8–13.

Fawcett, T., Provost, F., 1997. Adaptive fraud detection. Data Mining and Knowledge Discovery 1 (3), 291–316.

Flach, P., Wu, S., 2003. Repairing concavities in ROC curves. In: Proc. 2003 UK Workshop on Computational Intelligence. University of Bristol, pp. 38–44.

Forman, G., 2002. A method for discovering the insignificance of one's best classifier and the unlearnability of a classification task. In: Lavrac, N., Motoda, H., Fawcett, T. (Eds.), Proc. First Internat. Workshop on Data Mining Lessons Learned (DMLL-2002). Available from: http://www.purl.org/NET/tfawcett/DMLL-2002/Forman.pdf.

Hand, D.J., Till, R.J., 2001. A simple generalization of the area under the ROC curve to multiple class classification problems. Mach. Learning 45 (2), 171–186.

Hanley, J.A., McNeil, B.J., 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology 143, 29–36.

Holte, R., 2002. Personal communication.

Kubat, M., Holte, R.C., Matwin, S., 1998. Machine learning for the detection of oil spills in satellite radar images. Machine Learning 30 (2–3), 195–215.

Lane, T., 2000. Extensions of ROC analysis to multi-class domains. In: Dietterich, T., Margineantu, D., Provost, F., Turney, P. (Eds.), ICML-2000 Workshop on Cost-Sensitive Learning.

Lewis, D., 1990. Representation quality in text classification: An introduction and experiment. In: Proc. Workshop on Speech and Natural Language. Morgan Kaufmann, Hidden Valley, PA, pp. 288–295.

Lewis, D., 1991. Evaluating text categorization. In: Proc. Speech and Natural Language Workshop. Morgan Kaufmann, pp. 312–318.

Macskassy, S., Provost, F., 2004. Confidence bands for ROC curves: Methods and an empirical study. In: Proc. First Workshop on ROC Analysis in AI (ROCAI-04).

Provost, F., Domingos, P., 2001. Well-trained PETs: Improving probability estimation trees, CeDER Working Paper #IS-00-04, Stern School of Business, New York University, NY, NY 10012.

Provost, F., Fawcett, T., 1997. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In: Proc. Third Internat. Conf. on Knowledge Discovery and Data Mining (KDD-97). AAAI Press, Menlo Park, CA, pp. 43–48.

Provost, F., Fawcett, T., 1998. Robust classification systems for imprecise environments. In: Proc. AAAI-98. AAAI Press, Menlo Park, CA, pp. 706–713. Available from: <http://www.purl.org/NET/tfawcett/papers/aaai98-dist.ps.gz>.

Provost, F., Fawcett, T., 2001. Robust classification for imprecise environments. Mach. Learning 42 (3), 203–231.

Provost, F., Fawcett, T., Kohavi, R., 1998. The case against accuracy estimation for comparing induction algorithms. In: Shavlik, J. (Ed.), Proc. ICML-98. Morgan Kaufmann, San Francisco, CA, pp. 445–453. Available from: <http://www.purl.org/NET/tfawcett/papers/ICML98-final.ps.gz>.

Saitta, L., Neri, F., 1998. Learning in the "real world". Mach. Learning 30, 133–163.

Spackman, K.A., 1989. Signal detection theory: Valuable tools for evaluating inductive learning. In: Proc. Sixth Internat. Workshop on Machine Learning. Morgan Kaufman, San Mateo, CA, pp. 160–163.

Srinivasan, A., 1999. Note on the location of optimal classifiers in n-dimensional ROC space. Technical Report PRG-TR-2-99, Oxford University Computing Laboratory, Oxford, England. Available from: <http://citeseer.nj.nec.com/srinivasan99note.html>.

Swets, J., 1988. Measuring the accuracy of diagnostic systems. Science 240, 1285–1293.

Swets, J.A., Dawes, R.M., Monahan, J., 2000. Better decisions through science. Scientific American 283, 82–87.

van der Putten, P., van Someren, M., 2000. CoIL challenge 2000: The insurance company case. Technical Report 2000–09, Leiden Institute of Advanced Computer Science, Universiteit van Leiden. Available from: <http://www.liacs.nl/putten/library/cc2000>.

Zadrozny, B., Elkan, C., 2001. Obtaining calibrated probability estimates from decision trees and naive Bayesian classiers. In: Proc. Eighteenth Internat. Conf. on Machine Learning, pp. 609–616.

Zou, K.H., 2002. Receiver operating characteristic (ROC) literature research. On-line bibliography available from: <http://splweb.bwh.harvard.edu:8000/pages/ppl/zou/roc.html>.