Multi-dimensional ROC Analysis with Decision Trees*

Technical Report 17-Jan-2002

C. Ferri-Ramírez[1] P. Flach[2] J. Hernández-Orallo[1]

[1] Dep. Sistemes Informàtics i Computació, Univ. Politècnica de València (Spain)
{cferri, jorallo}@dsic.upv.es
[2] Department of Computer Science, University of Bristol (United Kingdom)
peter.flach@bristol.ac.uk

Abstract

In this paper, we review the fundamental aspects of cost-sensitive learning and ROC analysis, highlighting the limitations of current approaches for problems with more than two classes and proposing some extensions and alternatives. In particular we address these problems for decision tree learners, although most of the results can be extended to other classifier systems. Among the new proposals included in this paper, we highlight two alternative representations for ROC curves that can be used for multidimensional problems, and new measures for evaluating cost matrix irregularity. We also discuss the problems of extending previous theoretical results on the efficient re-assignment of classes of a two-class rule-based model to multi-class problems and we outline some directions for covering a multidimensional ROC space.

Keywords: Cost-sensitive learning, ROC analysis, decision tree learning, convex-hull approximation, model representation, misclassification cost.

1 Introduction

Cost-sensitive learning is a more realistic generalisation of predictive learning. Success of a learnt model is measured in terms of cost minimisation rather than in terms of error minimisation. When cost matrices are provided a priori, i.e. when learning takes place, the matrix has to be fully exploited to obtain models that minimise cost. Although several methods have been developed to take cost matrices into account, none of them exploit the whole information of cost matrix, especially in problems with more than two classes. In a very similar way, ROC analysis [21][26] has been proven very useful for evaluating classifiers, especially when the cost matrix is not known a priori. However, its applicability has only been shown for problems with two classes. Although ROC analysis can be extended naturally in theory for multi-dimensional problems [25], practical issues (computational complexity and representational comprehensibility, especially) preclude its use in practice.

In general, a classifier is learnt to use its predictions for decision support. Since predictions can be wrong, it is important to know what the effect is when the predictions are incorrect. In

* This work has been partially supported by CICYT under grants TIC98-0445-C03-C1, TIC2001-2705-C03-01 and by Generalitat Valenciana under grant GV00-092-14. This work has been performed during two research stays in the Department of Computer Science of the University of Bristol. The stays have been granted by Universitat Politècnica de València and by Generalitat Valenciana.

this way, it is more appropriate to redefine the quality of a classifier in terms of the expected results in the context where it is going to be used. Accuracy (or error), i.e., percentage of instances that are correctly classified (respectively incorrectly classified) has been traditionally used as a measure of the quality of classifiers. However, in most situations, not every misclassification has the same consequences. For instance, a wrong diagnosis or treatment can have different cost and dangers depending on which kind of mistake has been done. In fact, it is usually the case that misclassifications of minority classes into majority classes (e.g. predicting that a system is safe when it is not) have greater costs than misclassifications of majority classes into minority classes (e.g. predicting that a system is not safe when it actually is). Obviously, the costs of each misclassification are problem dependent, but it is almost never the case that they would be uniform for a single problem. Consequently, accuracy is not generally the best way to evaluate the quality of a classifier or a learning algorithm.

As a result of this rationale and the increasing power and applicability of learning systems, there has been an increasing interest in assessing classifiers according to cost minimisation. Although there can be other kinds of cost associated with predictions [28] (e.g. test cost), the most relevant ones are misclassification costs, i.e., the cost of classifying an instance of class *a* into class *b*. All these misclassification costs for a specific problem can be arranged in a *c*-dimensional matrix, with *c* being the number of classes. This matrix is called a cost matrix.

The use of cost matrices for the generation of classifiers that minimise the resulting prediction cost instead of the prediction error has been incorporated in a few aspects of a few learning systems by changing some criteria or measures used by these methods [19][3][14]. Nonetheless, it is also common to use a learning system that is not cost-sensitive and to modify the class distribution of the training data set to obtain a classifier that adjusts itself to a specific cost matrix and the class distribution of the test set if known [16][8][5].

However, a change of class distribution is usually done by stratification (or re-balancing), i.e., either by under-sampling or by over-sampling. Stratification presents some problems though. Weiss and Provost state it clear [30], "under-sampling throws out potentially useful data while over-sampling increases the size of the training set and hence the time to build a classifier [...]. Cost-sensitive learning methods appear to us to be a more direct an appropriate method for dealing with class imbalance than artificially modifying the class distribution via under-sampling or over-sampling, if the cost of acquiring and learning from the data is not an issue".

Apart from the disadvantages of stratification in a bi-dimensional problem (where the dimension d=2), there are additional and more relevant disadvantages in multi-class problems where d>2. When d>2, we have more independent cells in the cost and confusion matrices than class frequencies, so it is more and more difficult to drive a learner to the desired point in the space. For instance, with 4 classes, there is a 12-dimensional space associated ($4 \cdot (4–1)$). Trying to drive a new classifier in a 12-dimensional space seems very difficult if only 4 parameters (the four class distributions) can be changed by using weights. In other words, a problem with k classes has a k-dimensional matrix with $k \cdot (k\text{-}1)$ degrees of freedom. However, "since it is only the relative values of the weights that matter, only k-1 of the input weights are independent, and only $k(k\text{-}1)$-1 of the weights in an arbitrary cost matrix are independent. Hence, [the] simple technique of weighting the inputs does not directly extend to problems with more than 2 classes" [17]. The experimental results of several methods using just the weight show that the improvement is not very significant and that it becomes lower and lower as the number of classes increase. "One possible explanation for the poor results is that this method requires converting a cost matrix Cost(j,i) to a cost vector $C(j)$ resulting in a single quantity to represent the importance of avoiding a particular type of error" [19].

Therefore it seems reasonable to devise methods that exploit all the detailed information of a class matrix.

The usefulness of cost-sensitive learning does not only apply when the cost matrix is known a priori. If the cost matrix is not know, one or many classifiers can be generated in order to behave well in the wider range of circumstances or contexts as possible. The Receiver Operating Characteristic (ROC) analysis [21][26] provides tools to select a set of classifiers that would behave optimally and reject some other useless classifiers.

A fully cost-sensitive system can be applied for creating a good ROC curve. In the bi-dimensional case, after some classifiers have been obtained, it is easy to recognise the point where to look for a next one, in order to cover the whole space and construct a finer ROC curve. In the bi-dimensional case, this can be done just by changing arbitrary the class distributions, which turns to be equal to generate arbitrary cost matrices. However, in the multi-dimensional case, there is a need to direct the search in a very precise way, and class distributions do not provide enough degrees of freedom. The result of using just class distribution will be that many areas may be left uncovered. It may even be the case that a random search could be better.

The main problem of a problem for more than 2 classes is complexity. A confusion matrix obtained from a problem of c classes has c^2 positions, and $(c \cdot (c-1))$ dimensions (d), i.e. all of the possible misclassification combinations, are needed to evaluate a classifier. The problem is that the complexity of computing a convex hull of n points for d dimensions has cost $O(n^d)$ [25]. For instance, for a problem of 4 classes, we would have cost 20^{12} for evaluating 20 classifiers.

In this paper, we review the fundamental aspects of cost-sensitive learning, and ROC analysis, highlighting the limitations of current approaches for problems with more than two classes and proposing some extensions and alternatives. In particular we address these problems for decision tree learners, although most of the results can be extended to other classifier systems. Among the new proposals included in this paper, we highlight two alternative representations for ROC curves that can be used for multidimensional problems, measures for evaluating cost matrix discrepancy, and a discussion about the extension of previous theoretical results on the exponential assignments to a decision tree to more than two classes.

The paper is organised as follows. In section 2 we describe the basic notions about cost-sensitive learning: cost matrices, confusion matrices and many derived matrices and measures. We also introduce the traditional ROC analysis and some extensions. Section 3 centres on how to perform ROC analysis for more than two dimensions, especially new kinds of representations and what particularities appear in class assignment. In section 4, several methods for using cost information in learning are discussed whereas in section 5 their effectiveness is evaluated experimentally. The results show that these methods are almost useless and it is just enough to reassign the classes a posteriori. These two sections are an extension of part of the contents found in [9]. In section 6 several methods to cover the ROC space are discussed (either by re-assigning classes of one tree or several trees) and the problem of extending these methods to more than two dimensions is highlighted. Section 7 concludes the paper with the open question raised and future directions.

2 Notions about cost-sensitive learning

In this section we include some classical notions about confusion matrices, cost matrices and ROC analysis. Those familiar with these issues should skip this section until subsection 2.4.

2.1 Misclassification Results and the Confusion/Misclassification Matrix

Given a classifier, it is usual that its accuracy could be lower than 100%, let us say, for instance, 87,5%. In this case, it may be interesting to know to which class the misclassified 12,5% goes and how this error is distributed.

A Confusion Matrix is a very practical and intuitive way of seeing such a distribution. Given 100 test examples and a classifier, an example of a Confusion Matrix for three classes {a, b, c} might be as follows:

		Actual a	Actual b	Actual c
Predicted	a	20	2	3
	b	0	30	3
	c	0	2	40

This matrix is understood as follows. From the hundred examples, 20 were of class 'a' and all were correctly classified, 34 were of class 'b' from which 30 were correctly classified as 'b', 2 misclassified as 'a' and 2 misclassified as 'c'. Finally, 46 were of class 'c' from which 40 were correctly classified as 'c', 3 misclassified as 'a' and 3 misclassified as 'b'.

We use the notation M for this matrix and M(*i,j*) refers to the element of predicted class *i* and actual class *j*.

Note that it is easy to obtain derived information from the confusion matrix. The vertical sums give the distribution of classes in the actual examples; the horizontal sums give the distribution of classes produced by the classifier.

A Confusion *Ratio* Matrix can also be obtained by normalising each column to one:

		Actual a	Actual b	Actual c
Predicted	a	1.0	0.059	0.065
	b	0.0	0.882	0.065
	c	0.0	0.059	0.87

This matrix is represented by $\bar{M}$. Note that this matrix cannot be used instead of the original one because we have lost the class distribution.

It is also easy to derive the so-called *accuracies per class*, which are defined as the examples that are correctly classified of each class divided by the number of examples of that class in the dataset. In the previous example:

	Actual a	Actual b	Actual c
Predicted Accuracy	100%	88,2%	87,0%

Another useful derived information is the Actual-side Condensed Confusion Matrix. For the previous examples, it would be:

		Actual a	Actual b	Actual c
Predicted	Correct	20	30	40
	Incorrect	0	4	6

The first row represents the examples that have been correctly classified. The second row represents the examples that have been incorrectly classified.

Finally, another useful derived information is the following Predicted-side Condensed Confusion Matrix. For the previous examples, it would be:

		Correct	Incorrect
Predicted	a	20	5
	b	30	3
	c	40	2

The first column represents the examples that have been correctly classified. The second column represents the examples that have been incorrectly classified.

From either of the previous matrices a summarised vector can be obtained:

Correct	Incorrect
90	10

Which gives the most simplified result: 90 instances have been correctly classified and 10 instances have been misclassified. From here it is easy to see that the accuracy is 90% and the error rate is 10%.

As we have discussed before, many times accuracy is a much too simplified measure of the quality of a classifier. For instance, given a dataset whose distribution of classes is (p_a= 0.85, p_b= 0.1, p_c= 0.05), i.e., most of the examples are of the class 'a', a simple classifier predicting everything into class 'a' would have 85% of accuracy.

Apart from using confusion matrices to avoid this kind of oversimplification, which can be misleading for assessing quality of classifiers, other measures have been introduced by using the separate predictive accuracies ($PAcc_i$). These can be based on the mean of the separate predictive accuracies, the product, the maximum or derived from informativeness measures. All of them try to balance the results of the accuracy of different classes, in order to improve the results for minority classes.

2.2 Processing given cost information

Given a classification problem domain, the information about the cost of correct or wrong classification can be given in very different degrees of detail.

2.2.1 Cost Matrix

The most complete way to provide the information about misclassification costs is a Cost Matrix (also known as Loss Matrix), which indicates the costs for correct and incorrect classifications. An example of a Cost Matrix for three classes {a, b, c} might be as follows:

		Actual		
		a	b	c
Predicted	a	-2.5	4	2
	b	2.1	-3.5	0
	c	1.2	1.3	-4

This example shows the usual portrait, the diagonal of the matrix shows the costs for correct classification (-2.5, -3.5, -4). These values are usually negative or zero, because a correct classification has benefits instead of costs. The other values represent different cases of misclassification. For instance, the value 2.1 in cell (b,a) means that classifying incorrectly an 'a' instance as a 'b' instance has a cost of 2.1.

We use the notation C for this matrix and C(i,j) refers to the element of predicted class *i* and actual class *j*.

From this matrix and the confusion matrix it is very easy to compute the cost of a classifier for a given dataset, just as the 1 by 1 matrix product, given a Resulting Matrix:

$$R(i,j) = M(i,j) \cdot C(i,j)$$

For instance, for the previous two examples, the resulting matrix would be:

		Actual		
		a	b	c
Predicted	a	-60	8	6
	b	0	-105	0
	c	0	2.6	-160

And just adding all the elements of the matrix, we have the overall cost:

$$Cost = \sum_i \sum_j R(i,j)$$

For small datasets or when these costs are computed for parts of a model (e.g. a part of a decision tree), it may be useful to compute a Laplace-corrected confusion matrix before, obtained as [18]:

$$M'(i,j) = n \frac{M(i,j) + \lambda}{c^2 \lambda + n}$$

where λ determines the strength of the Laplace correction, *c* the number of classes and *n* the cardinality.

2.2.2 *Cost Matrix Properties*

Elkan [7] presents some desirable cost matrix properties for the two classes case. Here we extend those properties to multiclass problems.

It is reasonable to assume that costs are greater for misclassification than for correct classification. This reasonableness condition can be stated as:

$$\forall i, j, j \neq i : C(i,i) < C(j,i)$$

The < sign could be replaced by a ≤ to make this condition looser. Another property that can be expected from a cost matrix is that there are no dominant rows, i.e., a row for which all the costs are less than the corresponding costs of other row. More formally.

$$\neg \exists j,k : \forall i : C(k,i) < C(j,i)$$

If this would not be the case, there will be some classes that will never be predicted in any situation whatsoever.

Although the previous properties are followed by all the examples we use in this paper, a cost matrix not following these properties do not have to produce problems with the methods and algorithms discussed in this work.

2.2.3 *Abridged Cost Matrices*

Another way to provide cost information is by an actual-side abridged cost matrix, where misclassified costs are not particularised for each of the remaining classes. For instance, an example of an actual-side abridged cost matrix for three classes {a, b, c} might be as follows:

		Actual		
		a	b	c
Predicted	Correct	-2	-2	-3
	Incorrect	1.25	3	1

This first row shows the costs for correct classification (-2, -2, -3). These values are usually negative or zero, because a correct classification has benefits instead of costs. The other row represents different cases of misclassification, independently to where the misclassification has gone. For instance, the value 1 in cell (Incorrect, c) means that classifying incorrectly a 'c' instance as either an 'a' or 'b' instance has a cost of 1.

The use of this kind of abridged cost matrix is given when the cost is associated with the problem (actual), for instance in the case stopping some existing loss. For instance, a diagnosis problem, where an unsuccessful diagnosis of an illness x has always the same cost (the economical cost of the diagnosis and the cost of not having identified the illness x on time), independently of the predicted diagnosis. That is, the actual illness is relevant, but not where the prediction errors (the failed diagnosis) go, supposing that the failed diagnosis is detected soon by an expert and has no further consequences.

It is easy to see that any abridged cost matrix can be corresponded with a full cost matrix, just repeating the misclassification cost for each class. For instance, the following cost matrix would give the same results as intended by the previous abridged cost matrix.

		Actual		
		a	b	c
Predicted	a	-2	3	1
	b	1.25	-2	1
	c	1.25	3	-3

Due to this equivalence, in what follows, given an abridged cost matrix we will use the corresponding full cost matrix instead.

In the same way, some applications provide with the reverse predicted-side abridged cost matrix, where misclassified costs are not particularised for each of the remaining classes. For instance, an example of an abridged cost matrix for three classes {a, b, c} might be as follows:

		Actual	
		Correct	Incorrect
Predicted	a	-2.5	4
	b	0	5
	c	-1	2

The first column shows the costs for correct classification (-2.5, 0, -1). These values are usually negative or zero, because a correct classification has benefits instead of costs. The other column represents different cases of misclassification, independently to where the misclassification was originated. For instance, the value 2 in cell (c, Incorrect) means that classifying incorrectly as a 'c' an instance (either 'a' or 'b') has a cost of 1.

The use of this kind of abridged cost matrix is more usual and is given when the cost is associated with the solution (the prediction), for instance in the case of the use of some resource or some investment. For instance, a treatment drug can be defined in this way, where an unsuccessful use of a drug has always the same cost (the economical cost of the drug and the cost of not having cured the illness), independently of the actual good drug. That is, the use of the good (actual) drug is irrelevant for the cost (or ethically it is irrelevant), but any of the fails has an extra cost.

It is easy to see that any predicted-side abridged cost matrix can be corresponded with a full cost matrix, just repeating the misclassification cost for each class. For instance, the following cost matrix would give the same results as intended by the previous abridged cost matrix.

		Actual		
		a	b	c
Predicted	a	-2.5	4	4
	b	5	0	5
	c	2	2	-1

Due to this equivalence, in what follows, given a predicted-side abridged cost matrix we will use the corresponding full cost matrix instead.

2.2.4 Weight Vector and Stratification

Another way of providing information about the relevance of each class is by giving a weight vector, indicating that some classes are more important than others. Apart from the fact that now we are talking about relevance instead of cost, there should be a way to convert this information to a cost matrix, in order to work in a similar way as in the previous cases.

The issue here is that there are many different ways to create a cost matrix from a weight vector, because the matrix provides more information.

One reasonable way is to understand the weight vector as a stratification (*over-sampling or under-sampling*) vector, i.e., to understand the value of the vector given for each class as a class frequency modifier to the dataset, also known as *class probability distribution*.

For instance, from this weight vector:

	a	b	c
Weights	3	5	2

If we would use over-sampling with these weights, we would have that the frequency of class 'a' would be multiplied by 3/2, and the frequency of class 'b' by 5/2, by conveniently duplicating some of the examples.

It is easy to show that the corresponding cost matrix to this over-sampling would be:

		Actual		
		a	b	c
Predicted	a	-3	5	2
	b	3	-5	2
	c	3	5	-2

Although, as we have said, there are many different ways to construct a cost matrix from a weight vector, in the rest of this paper we will use the latter stratification equivalence.

Note that, in general, it is not possible to reduce a cost matrix to a weight vector without losing information when d>2.

This is related to the fact that for two dimensions, the following matrix:

		Actual	
		a	b
Predicted	a	-c	d
	b	c	-d

is equivalent for assigning classes to:

		Actual	
		a	b
Predicted	a	0	d
	b	c	0

This is easy to show. Consider a rule with the following distribution (n_a, n_b). Then, if we have that *a* is assigned to minimise cost if and only if

$$n_a\, C(0,0) + n_b\, C(1,0) < n_a\, C(1,0) + n_b\, C(1,1)$$

For the second matrix we have that this is the case if and only if:

$$n_b \cdot d < n_a \cdot c$$

Since C(0,0) = C(1,1) = 1, and C(1,0)= d, C(1,0)= c.

For the first matrix we have that this is the case if and only if:

$$n_a\,(-c) + n_b\, d < n_a\, c + n_b\,(-d)$$

iff

$$2\, n_b\, d < 2 n_a\, c$$

iff

$$n_b \cdot d < n_a \cdot c$$

This result does not hold for more than two classes.

2.2.5 *Cost matrix normalisation*

Given a cost matrix it is easy to see that we can divide any element of the matrix by a constant and the resulting overall cost will be divided by that constant. However, using this property to normalise a matrix does not give any further advantage.

There is, though, a usual normalisation performed to a cost matrix. Provided the cost matrix always has lower costs for correct classifications than for incorrect classification (reasonableness property), the following normalisation can be performed:

1. For each column *i*, select the correct classification cost C(*i*,*i*).
2. Add –C(i,i) to each value of the *i*th column.

Let us see it with an example. Given the following cost matrix:

		Actual a	Actual b	Actual c
Predicted	a	-3	15	3
	b	2	-4	0
	c	7	2	-1.5

The corresponding normalised matrix would be:

		Actual a	Actual b	Actual c
Predicted	a	0	19	4.5
	b	5	0	1.5
	c	10	6	0

However, the normalised matrix cannot be used instead of the original one, because the correction of the derived cost is not independent to a specific dataset distribution. For instance, given the following class absolute frequencies: Na = 100, Nb= 200, Nc= 150, the original cost should be computed as:

$$\text{OriginalCost} = 100 \cdot (-3) + 200 \cdot (-4) + 150 \cdot (-1.5) + \text{Derived Cost}$$

2.2.6 *Cost matrix « positivation »*

If the problem with a cost matrix is that there are negative values, a better way to modify the matrix is to "positive" it, i.e., to subtract to each cell the lowest negative value of the whole matrix. For instance, for the previous original matrix, we would have:

Actual

		a	b	c
Predicted	a	1	19	7
	b	6	0	4
	c	11	6	2.5

In this case, the "positivated" matrix *can* be used instead of the original one, because the correction of the derived cost is independent to a specific dataset distribution. For instance, given the following class absolute frequencies: Na = 100, Nb= 200, Nc= 150, the original cost should be computed as:

$$\text{OriginalCost} = (100 + 200 + 150) \cdot (-4) + \text{Derived Cost}$$

2.2.7 *Cost matrix standardisation*

We call a "unity" matrix, the matrix which sums 1, i.e.

$$1 = \sum_i \sum_j C(i, j)$$

An easy way to make a matrix follow this property is just divide each cell by the sum of the entire matrix.

When a matrix has been positivated and it has been modified to sum 1, we call it a *standardised* cost matrix. It is easy to show that a standardised matrix can be used instead of the original one.

2.2.8 *Cost matrix irregularity*

It is important to assess the irregularity of a cost matrix, because cost-sensitive and non-cost-sensitive learners will behave similar for regular cost matrices but they will differ for matrices where costs are quite irregular. For instance, the following matrix:

		Actual		
		a	b	c
Predicted	a	1	19	7
	b	6	0	4
	c	11	6	2.5

seems more irregular than the next one:

		Actual		
		a	b	c
Predicted	a	1	10	10
	b	6	0	6
	c	8	8	2.5

and this one more irregular than:

		Actual		
		a	b	c
Predicted	a	1	10	10
	b	10	0	10
	c	10	10	2.5

Let us introduce some measures of the irregularity of a cost matrix. The first and most naïve way to measure the irregularity of a cost matrix is the use of variance. Given an $n \times n$ cost matrix, its irregularity can be obtained by computing the variance or the standard deviation of the $n \times n$ values ($m = n \times n$). However, this would depend on the absolute values of the matrix.

$$sd = \sqrt{\frac{m\sum_{i,j} C(i,j)^2 - \left(\sum_{i,j} C(i,j)\right)^2}{m(m-1)}}$$

Consequently, a better way to compute the irregularity of a cost matrix is to standardise the matrix and then compute the standard deviation.

It may also be interesting to compute the irregularity in either the vertical or the horizontal way, in order to know in which direction the irregularity is given. For that, we give the following definition (which normalises each column independently:

$$Horiz\_Irreg = \frac{1}{n}\sum_j \sqrt{\frac{n\sum_i \left[\frac{C(i,j)}{\sum_i C(i,j)}\right]^2 - \left(\sum_i \frac{C(i,j)}{\sum_i C(i,j)}\right)^2}{n(n-1)}}$$

In a similar way, the Vertic_Irreg can be defined.

Another way to measure the irregularity of a cost matrix is by computing a measure related to its random expectancy. The random expectancy of a cost matrix can be defined as what the cost would be if predictions would be generated in a random way. This means (assuming uniform distribution) that the cost would be the one given by a confusion matrix where all the cells are equal, so giving a resulting matrix which is just proportional to the cost matrix. In a similar way, the change expectancy can be computed by considering what would be the average effect of interchanging a prediction in a confusion matrix (adding one to one cell and subtracting one to other cell). This change expectancy (or sensitivity) can be computed in the following way:

$$Sensitivity = \frac{1}{n}\sum_j \frac{\sum_{i,k,i\neq k}\left(C(i,j) - C(k,j)\right)}{n(n-1)}$$

Note that it only makes sense to consider the changes from one predicted class to other predicted class, because the other changes are not possible (an actual class can never change to a different actual class). Obviously, the minimum value for sensitivity is equal to 0 (given for uniform matrices) and the maximum value for sensitivity is equal to $2 \cdot Sum/n^2$, where Sum is the sum of values in the cost matrices. This maximum is given by a matrix full of zeros that has a Sum value in one cell.

A relative value of sensitivity can be given using this maximum:

$$RSensitivity = \frac{Sensitivity}{2 \cdot Sum/n^2} = n^2 \frac{Sensitivity}{2 \cdot Sum}$$

which yields a value between 0 and 1 of irregularity.

2.3 Traditional ROC Analysis

The Receiver Operating Characteristic (ROC) analysis [21][26] allows the evaluation of classifier performance in a more independent and complete way than just using accuracy. ROC analysis has usually been presented for two classes, because it is easy to define, to interpret and it is computationally feasible.

In ROC analysis with two classes the following notation is used for the confusion matrix:

		Actual: T	Actual: F
Predicted	T	True Positives (TP)	False Positives (FP)
	F	False Negatives (FN)	True Negatives (TN)

And as we have said, the ROC Analysis is performed using the ratios, which are easily obtained:

- True Positive Rate: TPR = TP/(TP+FN)
- False Negative Rate: FNR = FN/(TP+FN)
- False Positive Rate: FPR = FP/(FP+TN)
- True Negative Rate: TNR = TN/(FP+TN)

This forms a confusion ratio matrix $\bar{M}$:

		Actual: T	Actual: F
Predicted	T	TPR	FPR
	F	FNR	TNR

Logically, TPR = 1 – FNR and FPR = 1 – TNR. These latter equivalences allow for just selecting two of the ratios and construct a bidimensional space with them, with a working area between (0,0) and (1,1).

ROC analysis is based on plotting the true-positive rate over the y-axis and the false-positive rate over the x-axis. For instance, given the following confusion matrix:

		Actual: T	Actual: F
Predicted	T	30	30
	F	20	70

we would have a TPR= 0.6 and a FPR= 0.3. We can easily draw this point in the ROC space (also called Lorentz diagram [10]) as is shown in Figure 1.

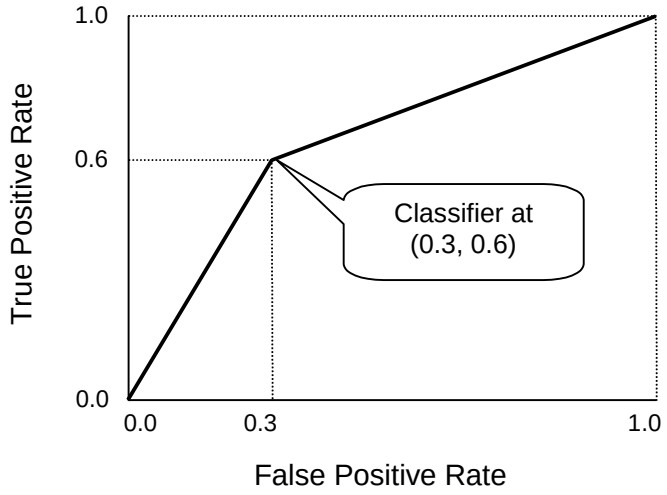


Figure 1. Example of ROC diagram

The points (0,0) and (1,1) represent, respectively, the classifier that classifies anything as negative and the classifier that classifies anything as positive. These are usually known as trivial classifiers.

There are many interesting things about the ROC analysis (summarised from [21]):

- Once a test set is presented, the actual cost given by the classifier can be computed from the actual probabilities of the two classes (p(T) and p(F)) and the cost matrix C(·, ·), by:

  $$\text{Cost} = p(T) \cdot (FNR) \cdot C(F,T) + p(F) \cdot (FPR) \cdot C(T,F)$$

  assuming that correct classifications have no cost.

  From the previous formula, two classifiers 1 and 2 will have the same cost when:

  $$\frac{TPR_2 - TPR_1}{FPR_2 - FPR_1} = \frac{p(F)\cdot C(T,F)}{p(T)\cdot C(F,T)} = m$$

  This m can be seen as the slope of a line. All classifiers in the same line have the same performance and the lines are called *iso-performance* lines.

- Given two classifiers, we can obtain as many derived classifiers as we want all along the segment that connects them, just by voting them with different weights. Consequently, any point "below" that segment will have more cost for any class distribution and cost matrix, because it has lower true positive rate and higher false positive rate.
- According to that property, given several classifiers, one can discard the classifiers that fall under the convex hull formed by the points representing the classifiers. The convex hull that is formed with all the classifiers (jointly with the points (0,0), (1,0) and (1,1)) is known as ROC Curve.
- Due to the previous rationale, the best learning system is the one that produces a set of classifiers that maximises the Area Under the ROC Curve (AUC).

A typical snapshot of a ROC diagram is illustrated in Figure 2. Suppose that the five classifiers plotted have been obtained A, B, C, D and E. Classifiers B and E fall under the ROC Curve (the convex hull formed by the five classifiers and the two trivial classifiers). Hence, B and E are discarded.

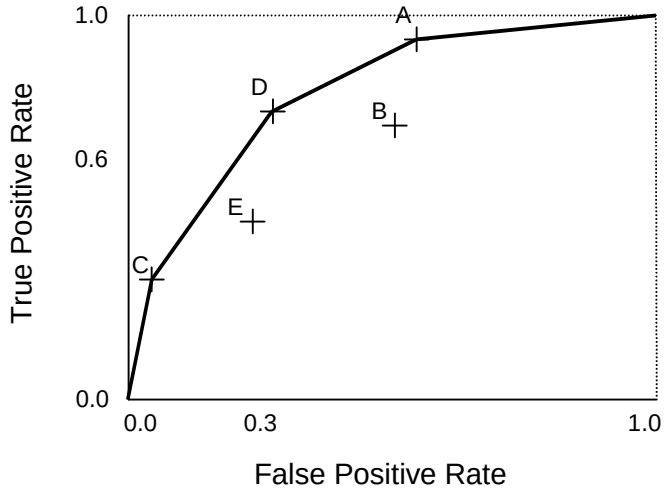


Figure 2. Example of a ROC curve

2.4 Extending ROC Analysis

Although the previous section shows the usual way to represent ROC space, it is not, in our opinion, a very coherent way, since the true class is represented incrementally for correct predictions and the false class is represented incrementally for incorrect predictions. Moreover, as we will see, this choice is not extensible for more than two classes. Consequently, for the previous classifier, we have a FNR= 0.4 and a FPR= 0.3, and we can draw the ROC space as is shown in Figure 3.

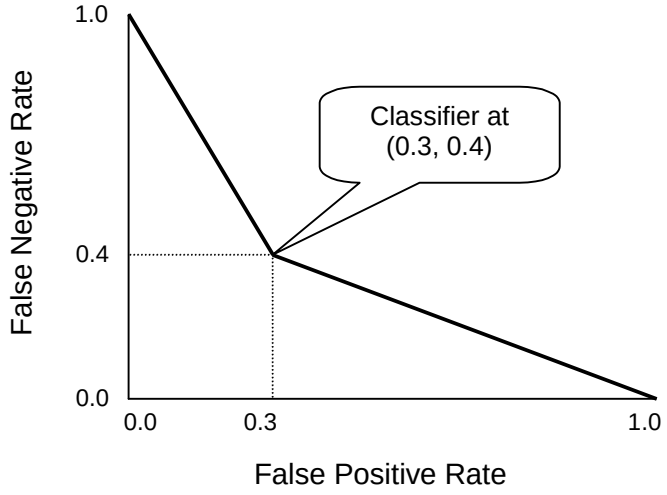


Figure 3. Example of inverse ROC diagram

Now, the points (0,1) and (1,0) represent, respectively, the classifier that classifies anything as negative and the classifier that classifies anything as positive. The ROC curve is now computed with points (0,1), (1,0) and (1,1).

Obviously, with this new diagram, instead of looking for the maximisation of the Area Under the ROC Curve (AUC) we have to look for its minimisation.

Another issue that can be raised about the ROC analysis is why it does not take into account matrices where the cost of correct classifications is not zero (it may be negative, showing a benefit, for instance).

Given a cost matrix with a non-zero diagonal like this:

		Actual T	Actual F
Predicted	T	-2	5.3
	F	2.5	7

One may ask whether the previous rationale for the ROC analysis holds. Fortunately, it is the case, as we will show. The result is that working with either the original cost matrix or the normalised cost matrix is the same for the ROC analysis.

Given the actual probabilities of the two classes (p(T) and p(F)) and the cost matrix $C(\cdot,\cdot)$ the cost of a classifier is obtained as:

$$\begin{aligned}\text{Cost} = {} & p(T)\cdot(FNR)\cdot C(F,T) + p(T)\cdot(TPR)\cdot C(T,T) + p(F)\cdot(FPR)\cdot C(T,F) + p(F)\cdot(TNR)\cdot C(F,F) = \\ & p(T)\cdot(FNR)\cdot C(F,T) + p(T)\cdot(1-FNR)\cdot C(T,T) + p(F)\cdot(FPR)\cdot C(T,F) + p(F)\cdot(1-FPR)\cdot C(F,F) = \\ & p(T)\cdot(FNR)\cdot(C(F,T) - C(T,T)) + p(T)\cdot C(T,T) + p(F)\cdot(FPR)\cdot(C(T,F) - C(F,F) + p(F)\cdot C(F,F)\end{aligned}$$

Two classifiers will have the same cost if:

$p(T) \cdot (FNR_1) \cdot (C(F,T) - C(T,T)) + p(T) \cdot C(T,T) + p(F) \cdot (FPR_1) \cdot (C(T,F) - C(F,F) + p(F) \cdot C(F,F) = p(T) \cdot (FNR_2) \cdot (C(F,T) - C(T,T)) + p(T) \cdot C(T,T) + p(F) \cdot (FPR_2) \cdot (C(T,F) - C(F,F) + p(F) \cdot C(F,F)$

namely,

$p(T) \cdot (FNR_1) \cdot (C(F,T) - C(T,T)) + p(F) \cdot (FPR_1) \cdot (C(T,F) - C(F,F)) =$
$p(T) \cdot (FNR_2) \cdot (C(F,T) - C(T,T)) + p(F) \cdot (FPR_2) \cdot (C(T,F) - C(F,F))$

namely,

$p(T) \cdot (FNR_1 - FNR_2) \cdot (C(F,T) - C(T,T)) = p(F) \cdot (FPR_2 - FPR_1) \cdot (C(T,F) - C(F,F))$

namely,

$$\frac{FNR_1 - FNR_2}{FPR_2 - FPR_1} = \frac{p(F)\cdot[C(T,F) - C(F,F)]}{p(T)\cdot[C(F,T) - C(T,T)]} = m$$

The line is again given by the same slope m:

$$-mx + y = c$$

varying c as desired. This shows that the result is the same as using the normalised cost matrix, which is obtained by subtracting C(F,F) to the false column and C(T,T) to the true column. For the previous example, it would be:

		Actual T	Actual F
Predicted	T	0	12.3
	F	4.5	0

Finally, apart from these extensions, one main concern of this paper is whether (and how) the ROC analysis can be effectively extended to an arbitrary number of dimensions.

3 ROC Analysis for more than two dimensions

Srinivasan has shown [25] that, theoretically, the ROC analysis extends to more than two classes directly. For c classes, and assuming a normalised cost matrix, we have to construct a vector of $d = c(c\text{-}1)$ dimensions for each classifier. In general the cost of a classifier for c classes is:

$$Cost = \sum_{i,j,i\neq j} p(i)\cdot C(i,j)\overline{M}(i,j)$$

where $\overline{M}$ is the confusion ratio matrix and p(i) is the absolute frequency of class i. From the previous formula, two classifiers 1 and 2 will have the same cost when they are on the same iso-performance hyperplane. However, the d-1 values of the hyperplane are not so straightforward and easy to obtain and understand as the slope value of the bi-dimensional case.

The hyperplane is defined by:

$$\sum_{1\leq k\leq d} m_k x_k = c$$

where one of the m_k is arbitrary set to 1.

In the same way as the bi-dimensional case, the convex hull can be constructed, forming a polytope. To know if a classifier can be rejected, it has to be seen if the intersection of the current polytope with the polytope of the new classifier give the new polytope, i.e., the new polytope is included in the first polytope [15].

Provided this direct theoretical extension, there are some problems.

- In two dimensions, doubling $p(T)$ (i.e. doubling the probability of one class) has the same effect on performance as doubling the cost $C(F,T)$ or halving the cost $C(T,F)$. Is this still true for more than two dimensions? The answer is *no* in one direction, because for a change in class distributions, there are infinite many corresponding cost matrices due to many more degrees of freedom.
- The best algorithm for the convex hull generation is $O(N \log N + N^{d/2})$ [15][2].
- In the 2-d case, it is relatively straightforward how to generate a spectrum of classifiers that cover the ROC space. It is not clear how to explore a multi-dimensional ROC space [15].

However, not only there are computational limitations but representational ones. ROC analysis in two dimensions has a very nice and understandable representation, but this cannot be directly extended to more than two classes, because even for 3 classes we have a 6-dimensional space, quite difficult to represent on a paper or a screen.

Let us first study some graphical alternatives for the classical ROC representation and then let us try to address approximations or special cases that could make it possible in practice.

3.1 Cobweb Representation

Given several *d*-dimensional points on the ROC space corresponding to classifiers, we can draw them in a cobweb representation.

For instance, given three classifiers with these confusion ratio matrices:

Point 1

		Actual A	Actual B	Actual C
Predicted	A	0.3	0.1	0.2
	B	0.2	0.6	0.1
	C	0.5	0.3	0.7

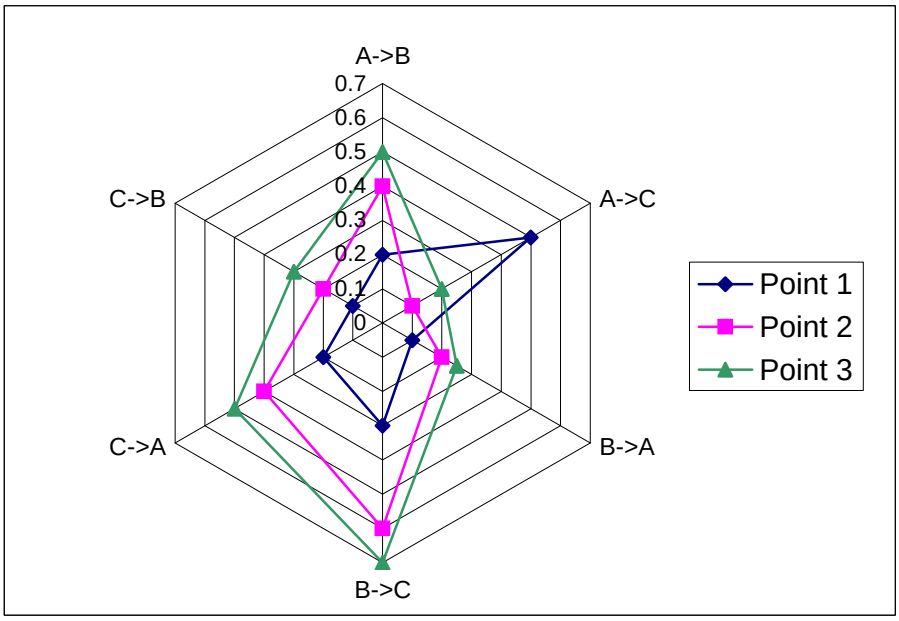
Point 2

		Actual A	Actual B	Actual C
Predicted	A	0.5	0.2	0.4
	B	0.4	0.2	0.2
	C	0.1	0.6	0.4

Point 3

		Actual A	Actual B	Actual C
Predicted	A	0.2	0.25	0.5
	B	0.5	0.05	0.3
	C	0.2	0.7	0.2

We can obtain d= (c·(c-1)) values from each classifier, in this case 6= 3·2, i.e. a point in a 6D space. The values of the different dimensions of a point are formed by the values in the confusion ratio matrix, ignoring the diagonal. This can be represented graphically using a cobweb representation as follows.



Note that point 3 can be discarded, because point 2 has all the coordinates below it.

The previous representation allows determining where each classifier has strong and weak points and, in simple cases, some of them can be discarded.

The previous representation, though, has two disadvantages:

- For many classifiers the representation gets highly confusing because the classifiers get very close.
- The convex hull cannot be drawn or recognised and consequently only classifiers where all the misclassification coordinates are below all the coordinates of another classifier can be discarded.

Moreover, it (graphically) ignores that each pair of classifiers can be combined to produce infinitely many classifiers stochastically combining their predictions. This is precisely why the convex hull has to be computed.

3.2 Lattice Representation

Another partially graphical representation, quite related to the previous one, is based on defining a partial order between classifiers, defined in the following way:

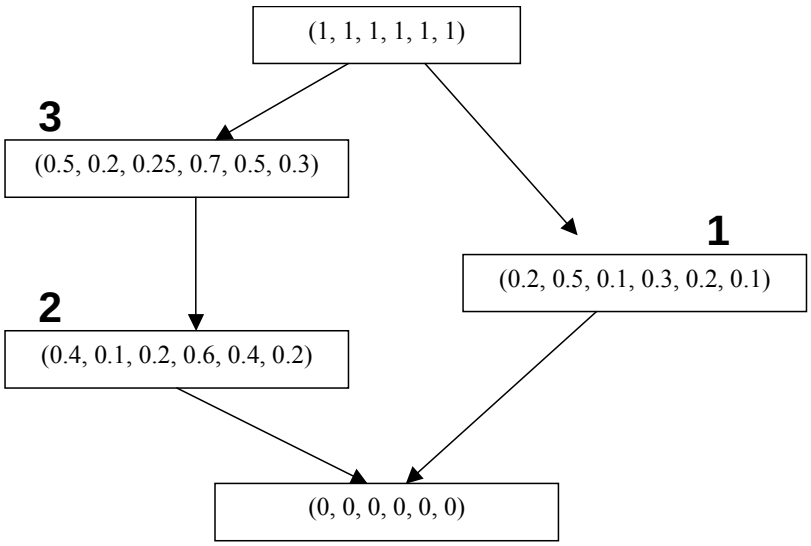
Given two d-dimensional points p and q,

$$p \leq q \text{ iff } \forall i: 1 \leq i \leq d \, , \, p_i \leq q_i$$

If we just consider points from the confusion ratio matrices, then we have that for any point p, $\forall i: 1 \leq i \leq d \, , \, 0 \leq p_i \leq 1$.

Consequently, given a set of classifiers, if we always add the points (1,1,...,1) and (0,0,...,0) then we can construct a lattice.

For instance, for the previous example, we would have the following lattice:



The point C can be discarded because has other point below in the lattice (except 0,0,0,0,0,0)

This representation, as well as the cobweb representation, has two disadvantages:

- For many points the representation gets very large.
- The convex hull cannot be recognised and consequently only points where all the misclassification coordinates are below all the coordinates of another point can be discarded.

This representation and the cobweb representation are equivalent.

3.3 Class Assignment

If there is a cost matrix (derived from weights or provided by the user), the class of a node or rule should not be computed as the majority class, but as the class that minimises the cost of the examples that fall under that node using the current cost matrix.

More concretely, if we have $Cost_i$ as the cost if class i would be selected:

$$Cost_i = \sum_j n(j) \cdot C(i, j)$$

Where $n(j)$ is the number of training examples of class j.

Then the assigned class will be:

$$AssignedClass = \arg\min_i Cost_i$$

Apparently, there is no difference between two or more than two dimensions. However, some "strange assignments" can be given for more than two classes.

For instance, for two classes, if one node or rule has no elements of one class, it is supposed that it will not be selected, supposing that the cost matrix is reasonable, in the sense described in section 2.2.1. Let us see that this is not the case for more than two dimensions.

Let us just study the behaviour of a node/rule in a 3-class problem depending on a cost matrix. In this context we have a cost matrix:

		Actual		
		a	b	c
Predicted	a	0	A_2	A_3
	b	B_1	0	B_3
	c	C_1	C_2	0

Where all the costs are greater than 0. And the leaf is :

E_A	E_B	E_C

The leaf will be assigned A iff:

$$E_B \cdot A_2 + E_C \cdot A_3 \le E_A \cdot B_1 + E_C \cdot B_3$$
$$E_B \cdot A_2 + E_C \cdot A_3 \le E_A \cdot C_1 + E_B \cdot C_2 \text{ , and}$$

The leaf will be assigned B iff:

$$E_A \cdot B_1 + E_C \cdot B_3 \le E_B \cdot A_2 + E_C \cdot A_3$$
$$E_A \cdot B_1 + E_C \cdot B_3 \le E_A \cdot C_1 + E_B \cdot C_2 \text{ , and}$$

The leaf will be assigned C iff:

$$E_B \cdot C_2 + E_C \cdot C_3 \le E_B \cdot A_2 + E_C \cdot A_3$$
$$E_B \cdot C_2 + E_C \cdot C_3 \le E_A \cdot B_1 + E_C \cdot B_3 \text{ , and}$$

We can express this problem more shortly, by defining the following parameters:

$$X_A = E_B \cdot A_2 + E_C \cdot A_3$$
$$X_B = E_A \cdot B_1 + E_C \cdot B_3$$
$$X_C = E_A \cdot C_1 + E_B \cdot C_2$$

As the following easy choice:

If $X_A \le X_B$ then
  If $X_A \le X_C$ then Class=A
  else Class=C
else
  If $X_B \le X_C$ then Class=B
  else Class=C

Let us suppose that there are no elements of one class, let's say A, then E_A=0. Now, we can show that a node will be assigned A iff:

$$E_B \cdot A_2 + E_C \cdot A_3 \le E_C \cdot B_3$$
$$E_B \cdot A_2 + E_C \cdot A_3 \le E_B \cdot C_2 \text{ , and}$$

i.e.

$$E_C \cdot A_3 - E_C \cdot B_3 \le -E_B \cdot A_2$$
$$E_B \cdot A_2 - E_B \cdot C_2 \le -E_C \cdot A_3 \text{ , and}$$

i.e.

$$\frac{A_3 - B_3}{A_2} \leq \frac{-E_B}{E_C}$$
$$\frac{A_2 - C_2}{A_3} \leq \frac{-E_C}{E_B}$$,and

Consequently, the node *will* be assigned A iff

$$\frac{B_3 - A_3}{A_2} \geq \frac{E_B}{E_C}$$
$$\frac{C_2 - A_2}{A_3} \geq \frac{E_C}{E_B}$$,and

Imagine we have this situation: $E_C=E_B=A_2=A_3=1$, and iff $C_2 \geq 2$ and $B_3 \geq 2$, then the node will be assigned A.

So it can happen that even without having no element of class A in one node, that node could be assigned class A.

The issue is that just in the case where $E_B=0$, $E_C=0$, then we can say that the leaf will be always assigned A and never to B or C. The proof is obvious, in this case it will be assigned to A iff:

$$E_B \cdot A_2 + E_C \cdot A_3 \leq E_A \cdot B_1 + E_C \cdot B_3$$
$$E_B \cdot A_2 + E_C \cdot A_3 \leq E_A \cdot C_1 + E_B \cdot C_2$$, and

i.e.

$$0 \leq E_A \cdot B_1$$
$$0 \leq E_A \cdot C_1$$, and

Since $E_A \geq 0$, $B_1 \geq 0$, $C_1 \geq 0$ it is true and that means that it is always assigned to A.

3.4 Area Under the ROC Curve for more than two classes

Although we have shown how difficult it seems to extend ROC analysis to more dimensions, there have been extensions of some measures.

For instance, Hand and Till present a generalisation of a particular AUC measure [11]. It has been shown that for two dimensions the AUC measure is equivalent to the GINI measure (not that the GINI measure is not the GINI splitting criterion used in the CART algorithm).

The idea is that in the AUC measure for 2 dimensions, they use the estimated probabilities of an example x_i pertaining to the class 0, denoted by $p_0(\cdot)$, (estimated from the training set), to rank the pairs $\{g_k, f_k\}$ where g_k and f_k are defined as $g_i = p_0(x1_i)$ and $f_j = p_0(x0_j)$ where $x1_i$ are the examples from the test set of class 1 and $x0_j$ are the examples from the test set of class 0. Note that instead of ordering nodes they order examples.

For instance consider the following two nodes for the training set:

Node 1: (4,1) --> class 0 with prob= 4/5 = 0.8

Node 2: (2,3) --> class 1 with prob= 2/5 = 0.4

And now consider that the test set is distributed in the following way over the decision tree:

Node 1: (6,4)

Node 2: (4,11)

with n_0= 10 elements of class 0 and n_1= 15 elements of class 1. Then we have:

- 6 of class 0 with $p_0(\cdot) = 0.8$
- 4 of class 1 with $p_0(\cdot) = 0.8$
- 4 of class 0 with $p_0(\cdot) = 0.4$
- 11 of class 0 with $p_0(\cdot) = 0.4$

From here, we can rank them as described in [11]. Let us denote with r_i the rank of the i^{th} class 0 test set point. Let us denote $S_0 = \Sigma r_i$. The they derive the area as:

$$\hat{A} = \frac{S_0 - n_0(n_0+1)/2}{n_0 n_1}$$

This area, although is an AUC (and it has the equivalence Gini + 1 = 2 × A), has two main differences with respect to usual ROC curves and also to our proposal in [9]:

- It is a step-like (or a stairs-like) area (no diagonals between the points are computed)).
- It is not convex, because the order is given by the training set and the examples are given by the test set.

Apart from this, the most relevant novelty of Hand and Till paper is that they understand *A* as "an overall measure of how well separated are the estimated distributions of $p_0(\cdot)$ for class 0 and class 1", i.e., A(i,j) could be computed for whatever pair of classes *i* and *j*.

This interpretation allows what they call "a simple generalisation of the AUC for multiple class classification problems". They define a new measure M as:

$$M = \frac{1}{c(c-1)} \sum_{i \neq j} \hat{A}(i,j) = \frac{2}{c(c-1)} \sum_{i<j} \hat{A}(i,j)$$

However, we have shown [9] that this result does not give better results that just the AUC for one point.

4 Using Cost Information for Learning

There has been some work on considering the costs of classification errors in learning systems. Some of this work has been produced within the context of decision tree learners [4][13][20][27]. However, most of them only introduce some small changes in a classical decision tree learning algorithm but do not modify coherently all the criteria used in a decision tree learner. Other systems use the information a posteriori and hence are independent of the learning method that is actually used.

In what follows we discuss the methods that have been essayed to treat cost information in a convenient way, during and after learning.

4.1 Using Cost Information for Generating/Guiding Hypotheses

Most of the decision tree systems that use cost information are based on changing the class distribution. A few of them do introduce small changes in a decision tree, such as changing the way in which classes are assigned to nodes or how the tree is pre-pruned or post-pruned. Let us extend these changes and make a thorough study of what has to be modified to make a decision tree learner cost-sensitive.

4.1.1 Computing Expected Cost

The expected error can be used in the selection criteria or in pruning. Some different modifications can be performed to the expected error in order to compute the expected cost instead.

The first way to compute the expected cost of a node is considering the class c_i which minimises the cost of a node. If we define $Cost_i$ as the cost if class i would be selected:

$$Cost_i = \sum_j n(j) \cdot C(i,j)$$

Where $n(j)$ is the number of training examples of class j. Then we have that a first simple way to compute the expected cost:

$$ExpCost_1 = \min_i Cost_i$$

This way of computing the expected cost does not take into account that many times most of the children of a node with class c_m could be assigned to other classes, so giving completely different cost.

A more coherent way of computing the expected cost is by considering the probability of each class c_i to be assigned as the class of a node.

Let us compute the probability of class i as:

$$p_1(i) = n(i) \,/\, n$$

where n is the total number of examples under a node (cardinality of the node). We can use any smoothing to $p(i)$ to obtain a better estimate of the class probability, e.g. Laplace smoothing:

$$p_2(i) = (n(i) + 1) \,/\, (n + \text{NumClasses})$$

Then, we can use this probability to estimate the expected cost as:

$$ExpCost_2 = \sum_i p_2(i) \cdot Cost_i$$

This, with Laplace smoothing, is the way that Bradford et al. claim to be the best way for pruning with costs [3].

However, this latter estimation of $p(i)$ disregards that the class will be assigned using cost and not majority. In our opinion, a better estimation of $p(i)$ would be to computed based on the costs.

We can obtain a probability by the following formula:

$$p_3(i) = \frac{1 \,/\, Cost_i}{\sum_j 1 \,/\, Cost_j}$$

If some value in the cost matrix is negative, these costs have to be computed by "positiving" the matrix, in order to avoid negative values.

From here, we can use this probability in the expected cost formula:

$$ExpCost_3 = \sum_i p_3(i) \cdot Cost_i$$

We also can apply smoothing to the previous formulae by using a modifying formula for Cost, for example a Laplace smoothing:

$$Cost'_i = n \cdot \sum_j p_2(j) \cdot C(i,j)$$

Example:

Let us illustrate it with an example. Consider the following cost matrix $C(\cdot,\cdot)$:

		Actual a	Actual b	Actual c
Predicted	a	-1	10	2
	b	3	0	5
	c	2	4	-2

and consider a node with 15 examples, with distribution vector $n(\cdot)$ = (3,2,10).

By using ExpCost1, we first compute the cost for each class:

$$Cost_a = 37$$
$$Cost_b = 59$$
$$Cost_c = -6$$

Obviously, using $ExpCost_1$, we would have:

$$ExpCost_1 = -6$$

For using $ExpCost_2$, we need to compute the probabilities (using Laplace correction):

$$p_a = 4/18$$
$$p_b = 3/18$$
$$p_c = 11/18$$

and now we obtain the $ExpCost_2$:

$$ExpCost_2 = (4/18) \cdot 37 + (3/18) \cdot 59 + (11/18) \cdot (-6) = 14.39$$

Finally, for computing the $ExpCost_3$ we have to compute the probabilities in a different way. First we have to compute the costs using the "positivated" cost matrix:

		Actual a	Actual b	Actual c
Predicted	a	1	12	4
	b	5	2	7
	c	4	6	0

Using this matrix the costs would be for each class: (3,2,10).

$$Cost_a = 67$$
$$Cost_b = 89$$
$$Cost_c = 24$$

and the probabilities derived from the costs:

$$p_a = (1/67) / ((1/67) + (1/89) + (1/24)) = 0.22$$
$$p_b = (1/89) / ((1/67) + (1/89) + (1/24)) = 0.17$$
$$p_c = (1/24) / ((1/67) + (1/89) + (1/24)) = 0.61$$

Finally, the expected cost using $ExpCost_3$ would be:

$$ExpCost_3 = (0.22) \cdot 37 + (0.17) \cdot 59 + (0.61) \cdot (-6) = 14.51$$

If we use smoothing to the costs, we would have:

$$Cost'_a = 15 \cdot [\, (4/18) \cdot (-1) + (3/18) \cdot 10 + (11/18) \cdot 2\,] = 40$$
$$Cost'_b = 15 \cdot [\, (4/18) \cdot 3 + (3/18) \cdot 0 + (11/18) \cdot 5\,] = 55.83$$

$$Cost'_c = 15 \cdot [\ (4/18)\ \cdot 2 + (3/18)\ \cdot 4 + (11/18)\ \cdot (-2)\ ] = -1.66$$

and now, the $ExpCost'_3$ would be:

$$ExpCost'_3 = (0.22)\ \cdot 40 + (0.17)\ \cdot 55.83 + (0.61)\ \cdot (-1.66) = 17.23$$

We could also use smoothing in computing p_a, p_b, p_c by using the vector (4,3,11) instead of (3,2,10).

4.1.2 Splitting criteria

The sensitivity of split criteria in decision tree learning has recently been studied experimentally and theoretically [27][6][7]. Although some experimental studies have shown that changing the class distribution can be relatively effective [30], some others have shown that usual splitting criteria (gain, gain ratio, gini and DKM) are insensitive in general to a distribution change [6][7].

The general formula of all these splitting criteria is to find the split with lower I(s), where I(s) is defined as:

$$I(s) = \sum_j p_j \cdot f(p^+_j, p^-_j)$$

where p_j is the probability of falling under that node in the split (cardinality of child node / cardinality of parent node). Using this general formula, each splitting criterion implements a different function *f*, as is shown in the following table:

Criterion	**f(a,b)**
Accuracy	min(a,b)
Gini	2ab
Entropy (Gain)	$a \cdot \log(a) + b \cdot \log(b)$
DKM	$2(a \cdot b)^{1/2}$

According to the experimental analysis of [6], the Accuracy (Expected Error) Splitting Criterion is the most sensitive one to distribution changes, while the others are relative insensitive, and DKM is the most insensitive of all. The theoretical study performed by Elkan [7] reinforces and even sharpens these experimental results: DKM is proven to be completely insensitive and effectiveness on C4.5 is usually small. In any case, the general conclusion is that changing the class distribution (even in the optimal way derived by [7]) when any of these discrimination criteria is used "is not likely to have much influence on the growing phase of decision tree learning" [7].

What Elkan recommends is to leave the training set untouched, use an insensitive learner (using a decision tree with DKM and no pruning for instance) and then apply a correction to the predictions according to the following formula:

$$\text{predict 1 iff } P(c=1 \mid x) \geq p^*$$
$$\text{otherwise predict 0}$$

where p^* is obtained in the following way, by using the cost matrix:

$$p^* = \frac{c(1,0) - c(0,0)}{c(1,0) - c(0,0) + c(0,1) - c(1,1)}$$

The relevance of this formula is that Elkan shows that it is the optimal way to correct the prediction.

Consequently he recommends the previous method instead of adjusting the training set in the following way:

$$\frac{p^*}{1-p^*}\frac{1-p_0}{p_0}$$

where p_0 is the proportion of the minority class expected in the test set (if known, if not it can be assumed 0.5 or assumed to be equal to the proportion in the training set). As we have said, this method would only work optimally if the learning algorithm were fully sensitive.

However, Elkan proposal for adjusting the predictions a posteriori has two drawbacks:

- it cannot be extended to more than two classes (due to the different degrees of freedom).
- the obtained model requires the correction, which can undermine its comprehensibility.

To overcome these problems, we are going to modify the classical splitting criterion. It is not the first time that it is done. Brieman et al. have presented two ways [4]: variable misclassification costs and altered prior method, the latter further studied in [19]. Both of them have provided relative unsuccessful results.

Our first method is based on using probabilities derived from costs, instead of using probabilities derived from frequencies.

In particular, we will use p_3, as was defined:

$$p_3(i) = \frac{1/Cost_i}{\sum_j 1/Cost_j}$$

From here, we could obtain the following cost-sensitive (and multidimensional) splitting criteria:

Cost-sensitive Criterion	**f(p_3(1), p_3(2), ...)**
Cost-sensitive Accuracy	$\min(p_3(i))$
Cost-sensitive Gini	$2 \cdot \prod p_3(i)$
Cost-sensitive Entropy (Gain)	$\Sigma\, p_3(i) \cdot \log(p_3(i))$
Cost-sensitive DKM	$\text{NumClasses} \cdot [\prod p_3(i)]^{1/\text{NumClasses}}$

However, these measures only take cost into account to reassess the probabilities but do not select in terms of cost. A more cost-sensitive criterion could be to use the expected cost as computed before:

Cost-sensitive Criterion	**f(p_3(1), p_3(2), ...)**
Expected Cost (3)	$\sum_i p_3(i) \cdot Cost_i$

Finally, a way on how to modify the MDL principle has not been explored, because it seems difficult to combine bits (as measured by the MDL principle) and costs.

A last idea would be to combine (i.e. to weight) the expected cost criterion with any of the cost-insensitive criterion (MDL, Gain, Gini, DKM, ...) by using just a formula of this kind:

$$I(s) = I_{crit1}(S)^{\alpha} \cdot I_{crit2}(S)^{\beta}$$

Alternatively, we have proposed splitting criteria based on ROC curves [9] but they do not use cost, they just try to open the tree in a way that it behaves better for a wide range of cost matrices.

5 Experimental Study on the Effectiveness of Minimising the Prediction Cost of a Decision Tree

5.1 Cost Matrix Choice

The first thing to consider when an experimental study has to be done is how to select the sample. The problem here is not the selection of several learning problems, which can give a biased result but it is the usual practice in ML, but the choice of the *cost matrices* for each problem.

Obviously, a very regular cost matrix will show a low difference between using or not cost information. However, very irregular matrices may have the tendency to give very extreme classifiers, or may show small differences between using the cost matrices or using only the weight vector (which changes the relevance of each class such as stratification). Moreover, the irregularity can be distributed horizontally, vertically or both ways. In section 2.2.8 several measures of matrix irregularity were introduced.

A random generation of matrices according to a uniform or a normal distribution will usually give very regular matrices. In order to make them more irregular, a very easy way seems to elevate the matrices cells to numbers greater than 1, because this exaggerates the differences.

Another way to generate cost matrices is by generating a very regular matrix (maybe a uniform one) and then changing some few values of it randomly. However, this method can be parametrised in many different ways and seems little reliable.

As a result, we have generated cost matrices in three different ways:

Name	*Method*	*Method Description*
40EX	40 runs, exaggerating by powering.	A random matrix is generated, according to a uniform distribution. The cells are powered by i, with i being the run (iteration).
50E&S	50 runs, exaggerating and softening by powering.	A random matrix is generated, according to a uniform distribution. The cells are powered by i or 1/i, depending on whether i is even or odd, with i being the iteration.
20IRR	20 runs, very irregular.	A uniform matrix is generated. A single very different value is placed in a random position of the matrix.

With these methods, the cost matrices generated have a different degree of irregularity according to the irregularity measures introduced in section 2.2.8. Obviously, there is no knowledge about how the irregularity distribution must be in real applications, but, fortunately, as we will see, the results are relatively independent of the method.

Let us see which are the differences between the three previous methods. We first use the "cars" example (with 4 classes), which is included in the UCI repository [29], and we use six of the learning methods that will be explained later. We represent the accuracy of the classifier (wrt. the test set) and the cost per example (also wrt. the test set).

```
                        (40 runs, exaggerating by powering)
                    Accuracy, CostperEx,
ECOST(MAJORITY) : 0.748437, 0.0182527,
ECOST(WO SMOOTH): 0.737269, 0.0149593,
ECOST(WI SMOOTH): 0.300405, 0.00291589,
C45  (MAJORITY) : 0.893519, 0.00828945,
C45  (WO SMOOTH): 0.888628, 0.00610563,
C45  (WI SMOOTH): 0.334693, 0.00192771,

                       (50 runs, exaggerating and softening by powering)
                    Accuracy, CostperEx,
```

```
ECOST(MAJORITY) : 0.735694, 0.0357393,
ECOST(WO SMOOTH): 0.730069, 0.0324213,
ECOST(WI SMOOTH): 0.419699, 0.0252315,
C45  (MAJORITY) : 0.893519, 0.0260252,
C45  (WO SMOOTH): 0.892662, 0.0234663,
C45  (WI SMOOTH): 0.574213, 0.0210357,

                      (20 runs, very irregular)
                   Accuracy, CostperEx,
ECOST(MAJORITY) : 0.785301, 0.00885417,
ECOST(WO SMOOTH): 0.800637, 0.00376157,
ECOST(WI SMOOTH): 0.633218, 0,
C45  (MAJORITY) : 0.893519, 0.0087963,
C45  (WO SMOOTH): 0.906481, 0.00416667,
C45  (WI SMOOTH): 0.67419,  0,
```

Although the absolute values differ considerably between the three different cost matrix sample methods, the comparison between the six learning methods gives similar results for the three sample methods, with the first method (40EX) being the one with more standard results. Consequently, we will use this first sample method for the following problems.

5.2 Evaluating Learning Methods

Now let us turn to the learning methods we are going to evaluate.

Method Name	Method Description
ECOST(MAJORITY) :	Expected Cost Splitting Criterion (using Cost Matrix) and using Majority to Assign the Class of leaf nodes..
ECOST(WO SMOOTH):	Same as before but using Cost matrix to assign class (with Smoothing)
ECOST(WI SMOOTH):	Same as before but using Cost matrix to assign class (without Smoothing)
C45 (MAJORITY) :	C45 Splitting Criterion and using Majority to Assign the Class of leaf nodes.
C45(STTCLASS WO):	C45 Splitting Criterion and using Weight vector to assign class (without Smoothing)
C45(STTCLASS WI):	C45 Splitting Criterion and using Weight vector to assign class (with Smoothing)
C45(STTALL WO) :	C45 Splitting Criterion (with probability modified by Weight Vector and smoothed) and using Weight vector to assign class (without Smoothing)
C45(STTALL WI) :	C45 Splitting Criterion (with probability modified by Weight Vector and smoothed) and using Weight vector to assign class (with Smoothing)
C45(STTALL WONS):	C45 Splitting Criterion (with probability modified by Weight Vector not smoothed) and using Weight vector to assign class (without Smoothing)
C45(STTALL WINS):	C45 Splitting Criterion (with probability modified by Weight Vector not smoothed) and using Weight vector to assign class (with Smoothing)
C45 (WO SMOOTH):	C45 Splitting Criterion and using Cost matrix to assign class (without Smoothing)
C45 (WI SMOOTH):	C45 Splitting Criterion and using Cost matrix to assign class (with Smoothing)

And the datasets we are using are:

Datasets:	No. Classes	No. Attributes	No. Examples Training	No. Examples Test
monks2	2	6	169	432
tae	3	5	76	75
cars	4	6	864	864
drugs	5	6	1100	1100

And the results are shown in the following tables:

	Monks2		Tae		cars		drugs	
	Accuracy	CostperEx	Accuracy	CostperEx	Accuracy	CostperEx	Accuracy	CostperEx
ECOST(MAJORITY)	0.678067	0.168822	0.583333	0.0695538	0.748437	0.0182527	0.748437	0.0182527
ECOST(WO SMOOTH)	0.675463	0.163502	0.584	0.0664391	0.737269	0.0149593	0.737269	0.0149593
ECOST(WI SMOOTH)	0.507986	0.0379743	0.354333	0.00880325	0.300405	0.00291589	0.300405	0.00291589
C45 (MAJORITY)	**0.740741**	0.134859	**0.6**	0.0634649	**0.893519**	0.00828945	**0.893519**	0.00828945
C45(STTCLASS WO)	0.731944	0.124859			0.890133	0.00648385	0.890133	0.00648385
C45(STTCLASS WI)	0.533218	0.0347342	0.371	0.0094319				
C45(STTALL WO)	0.741204	0.117495			0.725	0.0152896	0.725	0.0152896
C45(STTALL WI)	0.529282	0.0349776	0.367	0.00939658				
C45(STTALL WONS)	0.737153	0.137659			0.778704	0.0172536	0.778704	0.0172536
C45(STTALL WINS)	0.527025	0.0373807	0.364	0.0101613				
C45 (WO SMOOTH)	0.730556	0.125227	0.599	0.0608008	0.888628	0.00610563	0.888628	0.00610563
C45 (WI SMOOTH)	0.521181	**0.0342052**	0.358333	**0.00834345**	0.334693	**0.00192771**	0.334693	**0.00192771**

If we concentrate on accuracy, we see that C45 splitting criterion using majority class assignment has the best accuracy, as expected, because it does not use cost information. If we focus on cost per example, we see that C45 with smoothing and using cost minimisation assignment has the best cost per example.

It is important to note that Expected Cost and the use of Stratification are, surprisingly, worse than no use at all of cost information in the splitting criterion. Finally, if we see how smoothing affects the selection of the class of a node, we realise that it has a quite remarkable effect; each method with smoothing has considerably better results than without it.

One final question is whether pruning has an important effect. In the next table we see that the comparison of methods is similar when pruning is activated.

PRUNING (EXPECTED COST) (1.05)		MONKS2
Accuracy	CostperEx	StdDev
ECOST(MAJORITY)	0.674491	0.189081
ECOST(WO SMOOTH)	0.651806	0.167432
ECOST(WI SMOOTH)	0.559074	0.115993
C45 (MAJORITY)	**0.726481**	0.171049
C45 (WO SMOOTH)	0.694537	0.145864
C45 (WI SMOOTH)	0.596019	**0.110106**

As a result of these and other experiments, we can sum up the following conclusions:

- Changing the splitting criterion in a cost-sensitive way seems to have no effect (or bad effect) in minimising the cost of a classifier. Although experiments on expected error should be done.
- The most important thing for making a decision tree cost-sensitive is how the class is assigned to each leaf node.
- Pruning or not is not relevant in terms of the difference between methods.

- Smoothing (in the moment of assigning the class) is very *relevant*.
- The use of a cost matrix is slightly better than the use of a weight vector (stratification).
- The exact deviation of the position where a classifier has gone wrt. the corresponding position given by the cost matrix is very difficult to assess, especially in multidimensional cases, because a single different value makes angle change dramatically.

The main consequence of the previous results is that:

It seems that *covering the ROC space can be done by just reassigning the classes of leaf nodes in different ways*.

In fact, some more restricted (but consistent) results have been obtained by [3], in particular the irrelevance of the pruning method, the relevance of smoothing, and the final conclusion "it will usually suffice to induce a single probability tree and use it with different loss matrices, especially in the same area of the ROC curve" [3].

6 Covering the ROC Space

One of the problems observed from the previous results is that the assignment of classes in a local way for each node makes it very difficult for guiding a tree to a specific ROC point. One approach to handle this is just forget about ROC analysis and just provide the user with the tree with the highest accuracy, and then let the user (automatically) reassign the classes whenever a new cost matrix is available (note that reassigning the cost represents just a few arithmetical operations and can be done in real time even for very huge trees). This turns out to be somehow similar to constructing a set of class assignments for the same tree that could cover the whole ROC space, and let the user (automatically) select the optimal one whenever the cost matrix is available. A second approach is a more thoughtful way to re-assign the nodes.

Next, in the first subsection we undertake the first approach, and its problems to be extended to more than two classes and then we undertake the second approach in the second subsection. In the third subsection we try to guide the search for one tree placed in one location and seek subsequent trees that could cover the ROC space more effectively.

Before, let state some features formally. Given a set of tree leaf nodes n_k, $1 \le k \le p$ and a training set S with possible classes c_i, $1 \le i \le nc$, we denote by $n(\mathrm{i})$ the number of examples of S that fall under node n, and we denote by $n_k(i)$ the number of examples of S that fall under node n_k and are classified under class i.

If the goal is to minimise the cost for the training set given a cost matrix, it is clear that the best way to assign the class locally is given by the following formula:

$$BestClass_k = \arg\min_i \sum_j n_k(j) \cdot C(i,j)$$

In the case that two or more classes have the same cost, the most common class is selected. In the case there is still a tie among two or more classes, the most common class of the test set is chosen from these remaining classes.

However, although this is the best way for the training set, this might not be the best way for a test set, and smoothing methods can be applied to $n_k(i)$ in order to obtain less overfitted estimations.

6.1 Covering the ROC Space by Changing Class Assignments of One Tree

The idea is that the ROC curve of a decision tree can be defined by all the possible assignments to the class nodes. It may be interesting to obtain the ROC curve corresponding to several trees and select the tree that has more ROC surface. This could provide the user with just one solution (the best ROC solution), which is comprehensible and can be parametrised for different cost matrices, likely giving low costs.

In [9] we have studied how to select a good tree for which these future assignments could be suitable for a wide range of cost matrices. For this it is useful to draw the ROC curve of a decision tree (with all the possible assignments) and select the tree with higher area under the ROC curve. This would be infeasible for a tree of thousands of leaves, because the number of assignments will be c^n, where c is the number of classes and n the number of leaf nodes. Even for two classes, the convex hull over these points would have cost in $O(n \cdot 2^n)$.

Fortunately, for two classes, we have proven in [9] that only n+1 points are necessary for computing the convex hull and we have identified these points, just the points given by ordering the nodes according to their class ratio. This turns to be a specialisation of a very similar method presented in [11]. The cost of this operation is just $O(n \cdot \log n)$, for ordering the leaves of a tree according to the leaf ratio. From the previous result now it is very easy to draw the curve of a simple tree and compare different curves from different trees. This even allows combining them, detecting the parts where each tree is dominant.

The previous result is quite nice when the number of classes is 2. However, it seems difficult to be generalised to more than two classes. The major problem is that in order to do it is dimensionality; for 3 classes we have a 6 dimensional problem. For three classes, the convex hull must now be generated from all the 6-dimensional points corresponding to all the 3^n assignments and the origin (0,0,0,0,0,0).

Experimentally, it seems that these c^n assignments are also reduced, but we do not know the formula or the subset of assignments that can be discarded. The following results show that it could be very significant if the result given for 2 classes could be generalised to more than 2 classes, because the complexity reduction is very important:

n	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
2 classes real + origin: n+1	3	4	5	6	7	8	9	10	11
2 classes all + origin: 2^n+1	3	5	9	17	33	65	129	257	513
3 classes real (experimental taking the maximum):	4	10	28	78	201	444	923	1664	2653
3 classes all + origin: 3^n+1	4	10	28	82	244	730	2188	6562	19684

The open question is to find the formula that follows the highlighted series. We only know that for n=1, the formula is clearly c+1, where c is the number of classes. For c=2, the formula is n+1, where n is the number of nodes.

6.2 Changing Class Assignments to Go to a Precise ROC point

We can invent arbitrarily different cost matrices and assigning the classes according to these matrices in order to go to different points in the ROC space. The question now is how we can assign the classes globally to obtain a classifier that adjusts more to a *given* cost matrix, in the sense that it gets closer to the corresponding point. As we have said, it may seem that the answer is to assign the one that minimises cost with that cost matrix. Let study with more detail whether this is so.

Consider the following example, where we have three classes and the following cost matrix C:

Actual

		a	b	c
Predicted	a	1	12	4
	b	5	2	7
	c	4	6	0

Let us suppose that a decision tree has been learned with 10 leaf nodes, with the following class distribution over the training set.

Node	Training Set Distribution a	b	c	Assigned Class
1	5	3	1	b
2	4	10	0	b
3	7	6	1	b
4	1	2	1	b
5	2	2	7	c
6	0	1	1	c
7	1	0	0	a
8	3	3	2	c
9	0	1	0	b
10	2	1	0	a

The last column shows the class assigned according to the previous BestClass criterion and without the use of smoothing. This finally would give the following confusion matrix:

		Actual a	b	c
Predicted	a	3	1	0
	b	17	22	3
	c	5	6	10

with accuracy 35/67 = 0.52.

As we can see in the previous matrix, if we ignore the central diagonal (which is not used in the ROC analysis) and separate each column, we have a:(17,5), b:(1,6), c:(0,3) which is not very complementary to the corresponding columns of the cost matrix a:(5,4), b:(12,6), c:(4,7), in the sense that one would expect low values for high values and vice versa.

The issue is that the use of a cost matrix for locally assigning the classes minimises the cost but does not ensure a good point in the ROC space. But which confusion vectors are we willing to obtain? A first idea is just to inverse the cost vectors a:(1/5, 1/4), b:(1/6, 1/12), c:(1/7, 1/4) and normalise to 1, i.e. a:(0.444, 0.555), b:(0.666, 0.333), c:(0.364, 0.636). A "good" confusion matrix would be any one such that its column vectors, once normalised, are similar to the previous ones.

But how can we obtain this kind of vector and what can be lost in accuracy?

For instance, for the previous problem a different class assignment could go like this:

Node	Training Set Distribution a	b	c	Assigned Class
1	5	3	1	b->a
2	4	10	0	b
3	7	6	1	b
4	1	2	1	b->c
5	2	2	7	c
6	0	1	1	b->c

7	1	0	0	a->c
8	3	3	2	c
9	0	1	0	b
10	2	1	0	a->c

that now gives the following confusion matrix:

		Actual		
		a	b	c
Predicted	a	5	4	2
	b	11	17	1
	c	9	8	10

with accuracy 32/67= 0.478.

In this case the vectors are a:(5,9), b:(4,8), c:(2,1) which are much more complementary to the corresponding columns of the cost matrix a:(5,4), b:(12,6), c:(4,7). In normalised terms, the vectors a:(0.357, 0.642), b:(0.666, 0.333), c:(0.333, 0.666) are much more similar to the desired vectors: a:(0.444, 0.555), b:(0.666, 0.333), c:(0.364, 0.636).

Some questions arise now:

- Is it justified to get closer to the desired vectors at the cost of losing some accuracy? (in the previous example, from 0.52 to 0.478). It may be the case that the second assignment is always under the ROC surface.
- How can we measure the deviation from the desired vectors? And, finally,
- How can we compute an assignment that minimises the deviation with limited accuracy lost?

Before answering the previous questions, let us have some insight. Why do we want to guide by using cost matrices? If the goal is to cover a confusion ROC space, why do not we guide by using confusion matrices?

If we are talking about a six-dimensional space (for three classes), so we can want to cover different vectors in this six-dimensional space. For instance, given the following point a:(0.2, 0.2), b:(0.6, 0.1), c:(0.35, 0.15), we could define a vector from that point and search for an assignment that yields a confusion matrix onto that vector, for instance:

		Actual		
		a	b	c
Predicted	a	20	6	7
	b	4	7	3
	c	4	1	18

Whose confusion *Ratio* Matrix $\bar{M}$ is:

		Actual		
		a	b	c
Predicted	a	0.714	0.429	0.25
	b	0.143	0.5	0.107
	c	0.143	0.071	0.643

whose corresponding point a:(0.143, 0.143), b:(0.429, 0.071), c:(0.250, 0.107) is on the vector a:(0.2, 0.2), b:(0.6, 0.1), c:(0.35, 0.15).

However, this is the ideal, but rare, situation, i.e., one and only one assignment can obtain a confusion matrix whose corresponding point in the ROC space is on the same vector than the desired vector. In general, other two situations might appear:

1. More than one assignment is on the desired vector. The assignment with the highest accuracy is selected.
2. No assignment can be done onto the desired vector. Since there can be many "close" assignments, a good idea is to select the closest one.

The last case is even extensible to all the cases, because a slight deviation on direction could be accompanied with a high increase in accuracy and this would usually result in a better ROC surface (it may even under the ROC surface). Consequently, a measure has to be defined in order to obtain the "best assignment". Several approaches for this are:

- A combination (a weighting) of angle deviation and distance to the centre (distance to the centre as an inverse measure to accuracy). A good weighting could be to compute the area between the point, the axis centre and the normal line from the point to the vector.

An exhaustive search is not feasible because given k nodes and nc classes, we have nc^k assignments. Alternatives to an exhaustive search could be to order the nodes by cardinality, or by accuracy (supposing majority class), or by estimated cost, or by stratification (using the formula from [7]) and by the use of dynamic programming.

Note that the method could be different if pruning is used or not or if smoothing is used or not. For the special case without pruning and without smoothing there is no point in doing the assignments different from majority assignment.

6.3 Covering the ROC Space by Using Several Trees

Provided that as discussed in the previous section there is a way to place a decision tree in a concrete space, it would be interesting to look for several trees at different points.

For instance, Figure 4 shows the situation after one classifier has been found at point (0.3, 0.4). Note that the vertical axis has been drawn inversely as usual, using False Negatives instead of True Positives, according to the equation FN= 1–TP. This representation is the one we described in section 2.4. One might want to look for the next classifier to cover all the space, so it seems reasonable to look for the largest space where no improvement to the default curve has been done yet. A first and simple way to do that is to find the mid point of the largest segment, as is shown in the figure.

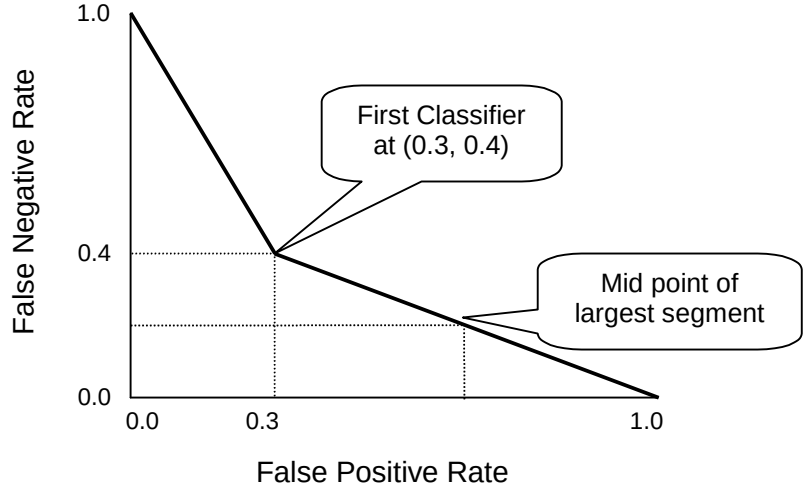


Figure 4. Example of inverse ROC diagram that shows the choice of next target.

This middle point gives a FN rate of 0.2 and a FP rate of 0.65. A very straightforward way to direct the new second classifier towards this zone is to select the cost matrix as follows:

		Actual T	Actual F
Predicted	T	0	1/0.65
	F	1/0.2	0

The question arises once again for more dimensions.

An idea could be to move a beam in different angles but again, for two dimensions it looks easy, given a number *r* of assignments, just generate at different (π/2r) angle intervals or by different 1/r one axis intervals. But this is not clear for more dimensions.

7 Conclusions

In this paper we have reviewed the main issues of cost-sensitive learning, and ROC analysis, highlighting the limitations of current approaches for problems with more than two classes and proposing some extensions and alternatives. Several new transformations and equivalences have been introduced and new measures for evaluating cost matrix discrepancy have been presented. With respect to representation, we have presented two alternative representations for ROC curves that can be used for multidimensional problems.

We have shown experimentally that cost information is almost useless in the moment of learning, since the differences are reflected mostly on the assignment of the tree leaves rather than its structure (topmost splits). Consequently, it seems a better way to assign the classes a posteriori, depending on the cost matrix in the moment of application.

Although there exists a very positive result in [9] on the way of how to efficiently construct a ROC curve from a single tree, this result is not straightforwardly extended to more than two classes and it is left as a open question. In the same way, how to generate different models that make a good curve is left open especially in the case of more than two classes.

Other future direction not considered here that could allow to tackle multidimensionality, is the study of ways to reduce dimensionality (which have only be shown feasible in some specific cases). Other ideas that could be investigated are:

- The use of an approximate convex hull algorithm of cost O($n \cdot d$) which computes whether a point is over or under the convex hull formed by a set of points [24].
- Adaptive methods to try to correct the deviation of previous guesses when covering the ROC space.

References

1. Blockeel, H.; Struyf, J. "Frankestein classifiers: Some experiments on the Sisyphus data set", ECML/PKDD2001 Workshop on *Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, 2001.
2. Boissonat, J.D.; Yvinec, M. *Algorithmic Geometry*. Cambridge University Press, 1998.
3. Bradford, J.; Kunz, C.; Kohavi, R.; Brunk, C; and Brodley, C. "Pruning decision trees with misclassification costs" in *Proc. of the European Conference on Machine Learning*, pp. 131-136, 1998.
4. Breiman, L.; Friedman, J.H.; Olshen, R.A. and Stone, C.J. *Classification and regression trees*, Belmont, CA, Wadsworth, 1984.
5. Domingos, P. "Metacost: A general method for making classifiers cost-sensitive" *Proc. of the Fifth International Conference on Knowledge Discovery and Data Mining*, pp. 155-164, New York, ACM, 1999.
6. Drummond, C.; Holte, R.C. "Exploiting the cost (in)sensitivity of decision tree splitting criteria", *Proc. of the Seventeenth International Conference on Machine Learning, pp. 239-246,* 2000.

7. Elkan, C. "The Foundations of Cost-Sensitive Learning", *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence*, IJCAI'01, 2001.
8. Fan, W.; Stolfo, S.; Zhang, J. and Chan, P.H. "AdaCost: Misclassification Cost-sensitive Learning" Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99), pp.97-105, Bled, Slovenia, June 1999.
9. Ferri-Ramírez, C.; Flach, P. Hernández-Orallo "Rocking the ROC Analysis with Decision Trees" Technical Report, Dep. of Computer Science, University of Bristol, 2001.
10. Hand, D.J. *Construction and assessment of classification rules*. Chichester: Wiley, 1997.
11. Hand, D.J.; Till, R.J. "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems" *Machine Learning*, 45, 171-186, 2001.
12. Kearns, M. and Mansour, Y. "On the boosting ability of top-down decision tree learning algorithms" *Proceedings of the Twenty-Eighth ACM Symposium on the Theory of Computing*, pp. 459-468, New York, ACM Press, 1996.
13. Knoll, U.; Nakhaeizadeh, G.; Tausend, B. "Cost-sensitive pruning of decision trees" *Proc. of the Eight European Conference on Machine Learning*, ECML-94, pp. 383-386, Berlin, Germany, Springer-Verlag, 1994.
14. Kukar, M.; Kononenko, I. "Cost-sensitive learning with neural networks" *Proc. of the Thirteenth Conference on Artificial Intelligence*, Chichester, NY, Wiley, 1998.
15. Lane, T. "Extensions of ROC Analysis to Multi-Class Domains", ICML-2000 Workshop on cost-sensitive learning, 2000.
16. Lavrac, N., Gamberger, D. and Turney, P. "Cost-sensitive feature reduction applied to a hybrid genetic algorithm." In Proc. Seventh International Workshop on Algorithmic Learning Theory, pp. 127-134, Springer, Berlin, 1996.
17. Margineantu, D.; Dietterich, T.G. "Learning Decision Trees for Loss Minimization in Multi-Class Problems", Technical Report 99-30-03, Department of Computer Science, Oregon State University, 1999.
18. Margineantu, D.; Dietterich, T.G. "Bootstrap Methods for the Cost-Sensitive Evaluation of Classifiers", Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000), pp.583-590, Morgan Kaufmann, San Francisco, CA, 2000.
19. Pazzani, M. J., Merz, C. J., Murphy, P., Ali, K., Hume, T., and Brunk, C. "Reducing Misclassification Costs" In Proceedings of the 11th International Conference of Machine Learning, Morgan Kaufmann, 217-225, 1994.
20. Provost, F.J. "Goal-directed inductive learning: Trading off accuracy for reduced error cost" *AAAI Spring Symposium on Goal-Driven Learning*, 1994.
21. Provost, F. and Fawcett, T. "Analysis and visualization of classifier performance: Comparison under imprecise class and cost distribution" in *Proc. of The Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pp. 43-48, Menlo Park, CA: AAAI Press, 1997.
22. Quinlan, J.R. *C4.5. Programs for Machine Learning*, San Francisco, Morgan Kaufmann, 1993.
23. Quinlan, J.R. "Improved use of continuous attributes in C4.5" *Journal of Artificial Intelligence Research*, 4, 77-90, 1996.
24. Smith, S. P., and Jain, A.K. "Testing for Uniformity in Multidimensional Data" IEEE Trans. on Pattern Analysis and Machine Intelligence, 6 (1984), pp. 73-81.
25. Srinivasan, A. "Note on the Location of Optimal Classifiers in N-dimensional ROC Space" Technical Report PRG-TR-2-99, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford.
26. Swets, J., Dawes, R., and Monahan, J. "Better decisions through science" *Scientific American*, October 2000, 82-87.
27. Turney, P.D. "Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm", *Journal of Artificial Intelligence Research* 2, pp. 369-409, 1995.
28. Turney, P. "Types of Cost in Inductive Concept Learning" Proceedings Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning (WCSL at ICML-2000), 15-21,2000.
29. University of California, UCI Machine Learning Repository Content Summary, http://www.ics.uci.edu/~mlearn/MLSummary.html.

30. Weiss, G. and Provost, F. "The Effect of Class Distribution on Classifier Learning: An Empirical Study" Technical Report ML-TR-44, Department of Computer Science, Rutgers University, 2001.