Measuring classification performance: the `hmeasure` package.

C. Anagnostopoulos, `canagnos@imperial.ac.uk`,
D.J. Hand, `d.j.hand@imperial.ac.uk`,
N.M. Adams `n.adams@imperial.ac.uk`
Department of Mathematics, South Kensington Campus,
Imperial College London, London SW7 2AZ

January 2, 2019

Abstract

The ubiquity of binary classification problems has given rise to a prolific literature dedicated to the proposal of novel classification methods, as well as the incremental improvement of existing ones, that largely relies on effective performance metrics. The inherent trade-off between false positives and false negatives, however, complicates any attempt at providing a scalar summary of classification performance. Several measures have been proposed in the literature to overcome this difficulty, prominent among which is the Area Under the ROC Curve (AUC), implementations of which are widely available. The AUC has recently come under criticism for handling the aforementioned tradeoff in a fundamentally incoherent manner, in the sense that it treats the relative severities of misclassifications differently when different classifiers are used. A coherent alternative was recently proposed, known as the H measure, that can optionally accommodate expert knowledge regarding misclassification costs, whenever that is available. The `hmeasure` package computes and reports the H measure alongside most commonly used alternatives, including the AUC. It also provides convenient plotting routines that yield insights into the differences and similarities between the various metrics.

1 Introduction

A binary classification rule is a method that assigns a class $y \in \{0, 1\}$ (or, respectively, $y \in \{\text{negative}, \text{positive}\}$) to an object, on the basis of its description $\mathbf{x} \in \mathcal{X}$. This extremely flexible formulation means that a wide variety of learning tasks encountered in practice are possible to express as a classification problem. Accordingly, this has led to a vast literature on the topic. Given the exponential growth of classification methods, and the growing scientific and economic impact of classification technology (ranging from fraud detection to medical diagnosis),

it is hard to understate the key role of appropriate measures of performance to enable easy, but statistically sound, comparisons across methods and datasets.

Measures of performance need to satisfy several criteria: first, they must coherently capture the aspect of performance of interest; second, they must be intuitive enough to become widely used, so that the same measures are consistently reported by researchers, enabling community-wide conclusions to be drawn; third, they must be computationally tractable, to match the rapid growth in the scale of modern data collection; finally, they must be simple to report as a single number for each method-dataset combination, so that the practical need to perform comparisons across various classifiers and datasets may be addressed.

In the field of classification, the study of measures of performance has matured over the past two decades in response to these, and other concerns, and has mainly converged to the use of any of a small selection of inter-related measures, usually expressed as weighted combinations of False Positive and False Negative Rates, or averaged versions thereof. This reflects the fact that binary classification performance inherently involves two largely antagonistic objectives: minimising the number of False Positives (e.g., occurrences where a disease is mistakenly diagnosed) as well as that of False Negatives (e.g., occurrences where the presence of a disease is missed). Classifiers are in fact typically able to trade-off one type of misclassification for another, depending on the preferences of the end user. This trade-off is often represented by a one-dimensional curve known as the Receiver Operating Characteristic (ROC) curve (see Section 2). Significant effort has been put into reducing this curve into a single number that would lend itself more easily to performance comparisons across classifiers. The most popular such scalar summary of the ROC curve is the Area Under the ROC Curve (AUC), currently used in over 6000 journal papers per year (source: Web of Science).

Nevertheless, the AUC has recently come under criticism [6, 7] for handling the two types of misclassifications in an incoherent fashion.

In particular, if one believes that a classifier's performance should be based on some combination of the relative losses due to the two types of misclassification, even when one does not know the relative severity of the two types of misclassification, then the AUC has been proven to be incoherent. (Of course, if one does not believe that the relative misclassification severities are relevant to performance, then the AUC is coherent). As a consequence, parts of the literature have responded by gradually shifting towards a recently proposed coherent, uncontroversial alternative known as the H measure.

The **hmeasure** package reports most popular measures of classification performance, including the AUC and H measure, for one or more binary classifiers. This paper illustrates the use of the package, by way of a tutorial centered around a worked example, also employed in the `hmeasure` package, which involves the Pima dataset from the MASS library.

2 Misclassification counts

In binary classification, the objective is to design a rule that assigns objects to one of 2 classes, often referred to as *positive* (i.e., *cases*, often encoded as 1s) and *negative* (*non-cases*, or 0s), on the basis of descriptive vectors of those objects. For instance, we may attempt to detect the presence of a certain medical condition on the basis of a set of medical tests. Rules are sometimes wholly designed on the basis of human expertise (e.g., by a doctor), but in many situations a typical scenario of interest is that of *supervised* classification, where rules are automatically *trained* to recognise objects of each class using a training set of data which includes both descriptive vectors and true classes for a sample of objects. The training algorithm itself may consist in heuristics, statistical estimation techniques, machine learning principles, or a combination thereof. The classifier is then deployed on a *test* dataset of n descriptive vectors, where the real classes of the objects in question are concealed from the classifier and used only to measure performance. In this work we will only be concerned with this latter issue of assessing performance using a test dataset.

2.1 Misclassification counts

The simplest performance metrics rely on comparing the classifier's predicted classes $\hat{y}_1, \ldots, \hat{y}_n$ for the n test datapoints with their true labels, $y_1, \ldots, y_n$. The primary statistics of interest are the so-called *misclassification counts*, i.e., the number of False Negatives (FN) and False Positives (FP). These are often reported alongside their complements, the number of True Negatives (TN) and True Positives (TP), in a *confusion matrix*:

		Predicted	
		$\hat{y}_i = 1$	$\hat{y}_i = 0$
Actual	$y_i = 1$	**TP**	**FN**
	$y_i = 0$	**FP**	**TN**

A scalar summary of the above is often needed, for instance, to enable comparisons across classifiers. In machine learning and statistics, the most widely used such summary is the *Error Rate* (ER), which is simply the total misclassification count divided by the number of examples, i.e., $\mathrm{ER} = \frac{\mathrm{FN}+\mathrm{FP}}{n} = \frac{\mathrm{FN}+\mathrm{FP}}{\mathrm{FN}+\mathrm{FP}+\mathrm{TN}+\mathrm{TP}}$. There are however several others which are widely used in other domains, as we now proceed to illustrate. Consider the Pima dataset, which we split into a training and a test dataset:

```
> require(MASS); require(class); data(Pima.te);
> library(hmeasure)
> n <- dim(Pima.te)[1]; ntrain <- floor(2*n/3); ntest <- n-ntrain
> pima.train <- Pima.te[seq(1,n,3),]
> pima.test <- Pima.te[-seq(1,n,3),]
```

This stores the training and test data in a matrix format, where the last, 8th column, called "type", is the class label, given as a factor with two levels:

```
> true.labels <- pima.test[,8]
> str(true.labels)

 Factor w/ 2 levels "No","Yes": 1 1 2 2 1 1 1 1 1 1 ...
```

We then consider the Linear Discriminant Analysis (LDA) classifier implemented in R as part of the `class` package. We first train the classifier using the training data:

```
> lda.pima <- lda(formula=type~., data=pima.train)
```

We then apply it to obtain predicted classes for the test dataset. We relabel to 0/1:

```
> out.lda = predict(lda.pima, newdata=pima.test)
> true.labels.01 <- relabel(true.labels)
> lda.labels.01 <- relabel(out.lda$class)
```

We may obtain the confusion matrix for these predictions using the function `misclassCounts`, which is part of the `hmeasure` package:

```
> lda.counts <- misclassCounts(true.labels.01, lda.labels.01); lda.counts$conf.matrix

         pred.1 pred.0
actual.1     50     25
actual.0     24    122
```

The function additionally computes several other misclassification-based statistics:

```
> print(lda.counts$metrics,digits=3)

     ER  Sens  Spec Precision Recall   TPR   FPR     F Youden
1 0.222 0.667 0.836     0.676  0.667 0.667 0.164 0.671  0.502
```

ER is the Error Rate described earlier. The True Positive Rate (TPR) and the False Positive Rate (FPR) constitute an alternative representation of the confusion matrix, and are given respectively by the ratio of TPs to the total number of positive examples, and the ratio of FPs to the total number of negative examples, $\mathrm{TPR} = \mathrm{TP}/(\mathrm{TP}+\mathrm{FN}), \mathrm{FPR} = \mathrm{FP}/(\mathrm{TN}+\mathrm{FP})$. An analogous pair of summary statistics is Sensitivity (Sens, same as TPR) versus Specificity (Spec, given by $1-\mathrm{FPR}$), and yet another such pair is Precision (TP divided by the total number of predicted positives, $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$) versus Recall (same as Sensitivity, or TPR). Finally, the F measure and the Youden index are scalar measures that attempt to take a more balanced view of the two different objectives than ER does. The former is given by the harmonic mean of Precision and Recall, and the latter by Sensitivity + Specificity $- 1$.

Note that `misclassCounts()` is flexible with regards to the data type of the predicted class label. The default input is a numeric array with levels 0 and 1, but it also accepts numeric, logicals, and factors, with the following conventions:

- numeric vector with values a and b: minimum$(a, b) \rightarrow$ "0", maximum$(a, b) \rightarrow$ "1"
- logical vector: `TRUE` $\rightarrow$ "1",`FALSE` $\rightarrow$ "0",
- factor levels: alphabetically, e.g., "**m**ale" $\rightarrow$ "1", "**f**emale" $\rightarrow$ "0",

The latter convention ensures that `"Yes"/"No"` are replaced with 1s and 0s respectively:

```
> relabel(c("Yes","No","No"))

[1] 1 0 0
```

An exception to this alphabetical ordering convention is made for the labels `"case/non-case"` which `relabel()` will map to `"1/0"` respectively, rather than to `"0/1"`:

```
> relabel(c("case","noncase","case"))

[1] 1 0 1
```

This is to ensure agreement with the usage of these terms in medicine, where a "case" represents the presence of a disease, and hence most naturally corresponds to a positive label. In all cases, `relabel()` outputs the following warning message for the user to inspect and ensure the relabelling agrees with their intuition about the meaning of the labels:

```
Class labels have been switched from (No,Yes) to (0,1)
Class labels have been switched from (case,noncase) to (1,0)
```

2.2 Scores and thresholds

The above statistics are based exclusively on a set of predicted labels. However, in most classifiers, predicted labels are in fact obtained by a two-stage process. Initially, the descriptive vector $\mathbf{x}$ is mapped to a scalar *score* $s(\mathbf{x}) \in \mathbb{R}$. This score is then compared to a *threshold*, t, to obtain the class label:

$$\text{Classifier}(t): \quad \hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } s(\mathbf{x}) > t \\ 0, & \text{if } s(\mathbf{x}) \leq t. \end{cases} \tag{1}$$

The threshold allows the end user to 'tune' a classifier in order to trade-off FPs for FNs, or vice versa. An extreme example is where one classifies all objects as positive ($t = -\infty$) regardless of their description, enabling one to never "miss a case" (FN=0), at the cost of a large number of "false alarms" (high FP). Conversely, for $t = \infty$, no objects will ever be classified as positive, forcing FP=0 at the cost of incurring a maximum number of FNs.

Such classifiers are known as *scoring* classifiers, and they form the prevalent type in the supervised classification literature. A particular special case

thereof are *probabilistic classifiers*, where the score is an estimate of the posterior probability of class 1 membership, $s(\mathbf{x}) = \hat{p}(\text{`Yes'} \mid \mathbf{x})$. The `lda` function is an implementation of a probabilistic classifier, and reports the scores in its output:

```
> out.lda$posterior[1:3,]

         No        Yes
2 0.9780078 0.02199222
3 0.9738716 0.02612837
5 0.3824495 0.61755047
```

Here the score may be found in the second column, which corresponds to the probability of class 1. The LDA classifier outputs the class label with maximum probability. In the framework above, this corresponds to a threshold of 0.5:

```
> scores.lda <- out.lda$posterior[,2];
> all((scores.lda > 0.5) == lda.labels.01)

[1] TRUE
```

We may alter the choice of threshold to assess the effect this has on the trade-off between FPs and FNs. Setting $t = 0.3$ produces an improvement in FNs (from 11 to 3) at the expense of a deterioration in FPs (from 7 to 12), or, as more commonly reported, an improvement in sensitivity (from 0.667 to 0.909), for a deterioration in specificity (from 0.91 to 0.846):

```
> lda.counts.T03 <- misclassCounts(scores.lda>0.3, true.labels.01)
> lda.counts.T03$conf.matrix

         pred.1 pred.0
actual.1     64     10
actual.0     36    111

> lda.counts.T03$metrics[c('Sens','Spec')]

       Sens     Spec
1 0.8648649 0.755102
```

2.3 ROC curves

The above discussion establishes that the performance of any given scoring classifier in terms of misclassification counts (and hence also in terms of ER) *varies* with t, and consequently, to evaluate such a classifier, or compare two or more such classifiers, one needs to either appropriately fix t, or somehow consider a suitable range of its values. Several solutions have been produced in the literature to address this difficulty (see [11] for a review). Some of these propose a certain setting for t and then employ misclassification counts as the measure of

performance. For instance, as explained earlier, what is commonly reported as 'Error Rate', is in fact just $\mathrm{ER}(\tilde{t})$ where $\tilde{t}$ is whatever default value is employed by the classifier implementation in question. More sophisticated approaches attempt to take a balanced view across several, or even all possible settings for t. We review both types of metrics in this section.

The most common graphical representation of this trade-off is the Receiver Operating Characteristic (ROC) curve, which is a plot of the Sensitivity (i.e., TPR) against 1- Specificity (i.e., FPR), as the threshold is varied from $-\infty$ to ∞. Computing the ROC curve only requires the scores, and does not actually involve iterating over different values of t (see [6] for details). This is implemented in the `HMeasure` function, and displayed by calling `plotROC` on its output. To make the discussion more interesting we consider a second classifier, the k-Nearest-Neighbours (k-NN) classifier, available in the `class` package:

```
> class.knn <- knn(train=pima.train[,-8], test=pima.test[,-8],
+   cl=pima.train$type, k=9, prob=TRUE, use.all=TRUE)
> scores.knn <- attr(class.knn,"prob")
> scores.knn[class.knn=="No"] <- 1-scores.knn[class.knn=="No"]
```

The last line of code is necessary as k-NN utilises a different convention for scores, outputing by default the score of the most probable class, rather than the score of class 1. The `HMeasure` function takes as input the set of true labels (following the same conventions as `misclassCounts()`, see Section 2.1), as well as a data frame of scores, where each column is a classifier:

```
> scores <- data.frame(LDA=scores.lda,kNN=scores.knn)
> results <- HMeasure(true.labels,scores)
> class(results)

[1] "hmeasure"
```

The output is an object of class `hmeasure`, whose main fields are a set of summary metrics including statistics based on misclassification counts (see Section 2.1), as well as aggregate metrics (see Section 2.4). It additionally contains information necessary for plotting purposes. In particular, calling the function `plotROC()` on an `hmeasure` object will produce, by default, the ROC curves of the classifiers in question. Figure 1 is an example of this for the LDA and k-NN classifiers on the Pima data.

We note in Figure 1 that the ROC curve of LDA lies mostly above that of k-NN. Had it lied *strictly* above it, then we would have been able to conclude that *for any value of the threshold*, LDA fared better for this dataset in terms of both FNs and FPs than k-NN. This result is also called *strict dominance*, and it is the strongest possible result of a classifier comparison. For instance, the ideal classifier would produce a ROC curve lying above all possbile ROC curves, and consisting in the line segments $(0,0) \to (0,1) \to (1,1)$, as the $(0,1)$ point represents perfect classification. As things stand, LDA is seen to fare better than k-NN for most, but not all values of the threshold. The diagonal

```
> plotROC(results)
```

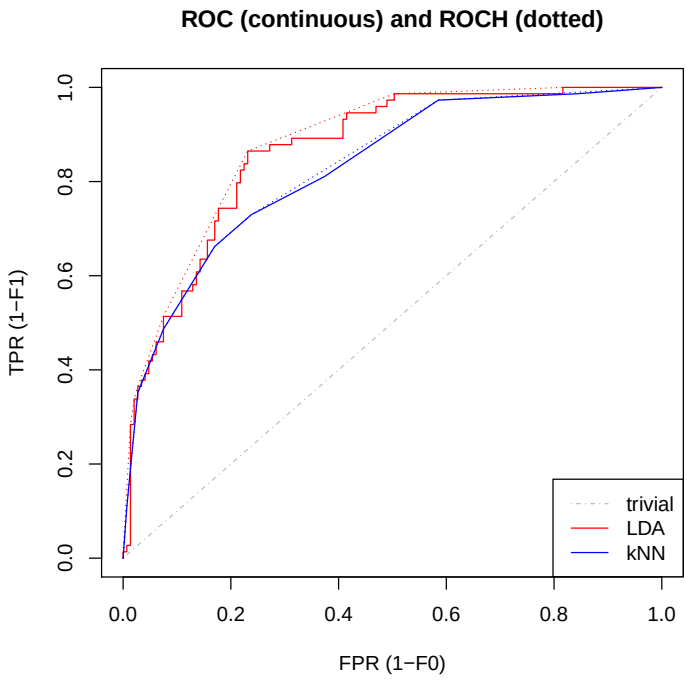

Figure 1: Code fragment producing an ROC curve for the LDA classifier. Providing the scores as a data frame allows us to name the method in question.

represents the performance of a trivial classifier, that randomly allocates classes to objects. Although it is possible in practice for a classifier to underperform the trivial classifier, it is customary in such cases to switch the sign of the scores and therefore 'flip' the ROC curve about the diagonal to produce a superior classifier. This is done automatically in the `HMeasure` function, and the following warning is produced:

```
> ROC curve of kNN mostly lying under the diagonal. Switching scores.
```

Finally, the dotted lines in Figure 1 represent the *convex hull* of the ROC curve (ROCH) for each classifier, i.e., the least convex curve that lies above its ROC curve. This representation is motivated by the fact that it is possible to modify any given classifier in a way that 'convexifies' its ROC curve [13].

2.4 Aggregate measures

We now turn our attention to aggregate measures, defined as functions of the ROC curve's values, at single or multiple locations. These are compactly re-

ported as follows:

```
> summary(results)

        Length Class      Mode
metrics 22     data.frame list
```

We define the abbreviations here, and give more detail in the rest of the section:

- H: the H-measure (see Section 2.6)
- Gini: the Gini coefficient
- AUC: the Area Under the ROC Curve
- AUCH: the Area Under the convex Hull of the ROC Curve
- KS: the Kolmogorov-Smirnoff statistic
- MER: the Minimum Error Rate
- MWL: the Minimum cost-Weighted Error Rate
- Spec.Sens95: Specificity when Sensitivity is held fixed at 95%
- Spec.Sens95: Sensitivity when Specificity is held fixed at 95%

In this section we explain how each of these metrics is computed. Note that overriding the default value of `show.all` in the `summary()` method additionally reports the metrics computed by `misclasscounts()` which we described earlier in Section 2.1:

```
> summary(results,show.all=TRUE)

        Length Class      Mode
metrics 22     data.frame list
```

These latter metrics all rely on a single pre-specified threshold value. The default value is $t = 0.5$, but the user may easily specify a different threshold as follows:

```
> HMeasure(true.labels,scores,threshold=0.3)$metrics[c('Sens','Spec')]

         Sens      Spec
LDA 0.8648649 0.7551020
kNN 0.8108108 0.6258503
```

Separate thresholds per classifier may be provided instead. E.g., the above agrees with

```
> HMeasure(true.labels,scores,threshold=c(0.3,0.3))$metrics[c('Sens','Spec')]
```

```
         Sens      Spec
LDA 0.8648649 0.7551020
kNN 0.8108108 0.6258503
```

but not with

```
> HMeasure(true.labels,scores,threshold=c(0.5,0.3))$metrics[c('Sens','Spec')]

         Sens      Spec
LDA 0.6756757 0.8299320
kNN 0.8108108 0.6258503
```

It should be noted that this feature of user-specified thresholds is introduced for reasons of completeness, transparency and experimentation only. It is generally not advised to directly select values for the threshold – this is discussed in Section 2.6. To relieve the user entirely of having to worry about whether the default choice of threshold $\tilde{t} = 0.5$ is suitable, we additionally report the *Minimum Error Rate* (MER), as an alternative to $\mathrm{ER} = \mathrm{ER}(\tilde{t})$, which corresponds to the value of t that achieves the *minimum* $\mathrm{ER}(t)$ over the test dataset. If $s(x)$ is a probability then the minimum error rate is generally achieved when $t = 0.5$. See Section 2.6 for further details on this measure.

Error rate considers FPs as important as FNs. This may not be suitable in certain application areas, such as medical diagnosis, or fraud detection. A popular alternative in such domains is to fix specificity (resp., sensitivity) at a desired level, often set at 0.95, and only report the achievable sensitivity (resp. specificity) at that level. These measures can be read off the ROC curve directly, and are also conveniently reported as `Sens.Spec95` and `Spec.Sens95` respectively in the `Hmeasure` output, for a default setting of 0.95. The user may easily specify one or more such levels as follows:

```
> summary(HMeasure(true.labels,scores,level=c(0.95,0.99)))

        Length Class      Mode
metrics 24     data.frame list
```

An alternative metric that seeks to jointly consider specificity and sensitivity is the Kolmogorov-Smirnoff (KS) statistic, which corresponds to the maximum value their sum takes as the threshold is varied. This also attains an intuitive graphical interpretation as the maximum vertical distance between the ROC and the diagonal.

The above metrics all focus on single locations on the ROC curve, i.e., single choices of thresholds. Other metrics attempt to aggregate over several values of the threshold instead. The most widespread such measure is the Area Under the ROC Curve (AUC), or its variation the Area Under the ROCH Curve (AUCH). The AUC has the intuitive property that if the ROC of one classifier lies strictly above that of another, the rankings of their respective AUCs will reflect this. However, the converse does not hold, as ROC curves will more often than not cross. In such cases, AUC has been criticised for considering

all values of sensitivity as 'equally likely', an assumption that is untenable on several grounds [6]. The AUC normally takes values between 0.5 and 1, although its chance-standardised version, the Gini coefficient, given by $2\text{AUC} - 1$, is sometimes preferred.

2.5 Score distributions

It is instructive to recast the above discussion in terms of the *class-conditional score distributions*, denoted by F_0 and F_1, and the respective *class priors* π_0 and π_1, given by

$$F_1(t) = P(s(\mathbf{x}) < t \mid 1),\ F_0(t) = P(s(\mathbf{x}) < t \mid 0),\ \pi_0 = P(0),\ \pi_1 = P(1)$$

If we employ the empirical score distributions (see Figure 2) as proxies for the unknown true distributions, it may be seen that $F_0(t) = 1 - \text{FPR}(t)$, $F_1(t) = 1 - \text{TPR}(t)$, $\pi_1 = \frac{TP+FN}{n}$ and $\pi_0 = \frac{TN+FP}{n}$. In particular, the ROC curve is a plot of $1 - F_1$ on the vertical axis against $1 - F_0$ on the horizontal axis, and:

$$\text{ER}(t) = \{\pi_0(1 - F_0(t)) + \pi_1 F_1(t)\}$$

Finally, the AUC may now be thought of as the following integral:

$$AUC = \int F_0(t) dF_1(t) = \int F_0(t) f_1(t) dt$$

The rightmost equality assumes the distribution is differentiable, which will not hold for an empirical distribution. To simplify the exposition, for the remainder of this discussion we assume that F_0 and F_1 are the true distributions.

2.6 Costs and the H-measure

Most of the above metrics attempt to take a balanced view of the trade-off between FPs and FNs. A principled way to achieve this is to introduce the notion of *misclassification costs*, which seek to quantify the relative severity of one type of error over the other. Let c in $[0, 1]$ denote the 'cost' of misclassifying a class 0 object as class 1 (FP), and $1 - c$ the cost of misclassifying a class 1 object as class 0 (FN). Consider then the total cost:

$$L(c; t) = 2(c\pi_0(1 - F_0(t)) + (1 - c)\pi_1 F_1(t))$$

where we have multiplied by 2 in order to make the numbers comparable to $\text{ER}(t)$, which is now a special case of $L(c; t)$ for equal misclassification costs, i.e., $c = 0.5$. It is worth noting that since each FN carries a cost of $1 - c$, it is implicitly assumed that the two misclassification costs sum to 1. This assumption in fact involves no loss of generality, as discussed in [6].

We may similarly generalise MER into the Minimum Weighted Loss (MWL) instead, by choosing the threshold that minimises $L(c; t)$ for each value of c:

$$\text{MWL}(c) = L(c; T_c), \quad \text{where } T_c = \operatorname*{argmin}_t L(c; t)$$

```
> plotROC(results,which=4)
```

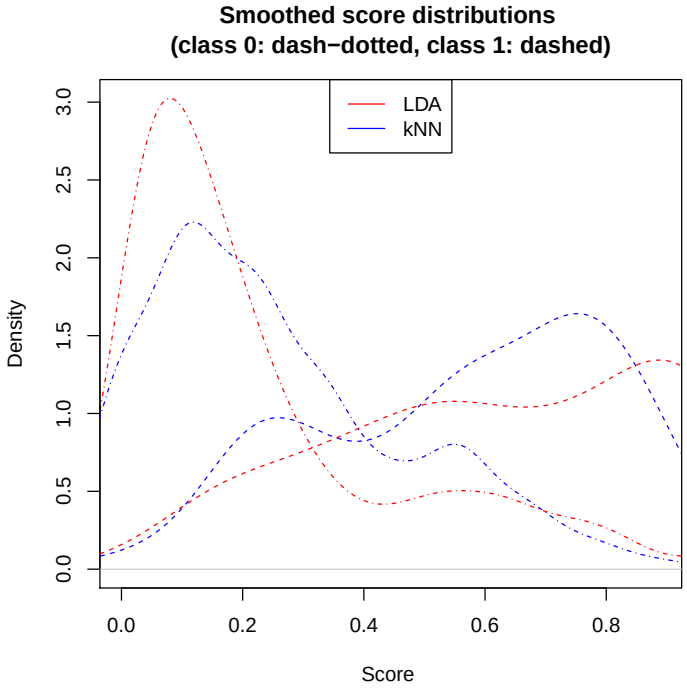

Figure 2: Empirical scoring densities for the LDA and k-NN classifiers on the Pima dataset.

Again, MER is now a special case of MWL for $c = 0.5$. Interestingly, the KS statistic is also related to a special case of MWL for $c = \pi_1$, a choice we motivate in the next section:

$$\text{KS} = \max_t\{\text{Sens} + \text{Spec}\} - 1 = \max_t\{F_0(t) - F_1(t)\} = 1 - \frac{\text{MWL}(\pi_1)}{2\pi_0\pi_1}$$

In the MWL setup, the threshold is no longer a free parameter of the classifier, but rather determined by the cost, c, and, under sensible conditions, uniquely (see [7]). The cost itself on the other hand is clearly a property of the application domain, not of the classifier. For instance, cost-sensitive classification is particularly popular in medical diagnosis of life-threatening diseases, where it must be taken into account that a false alarm generally costs less than a missed case, so $1 - c >> 0.5$. In practice however it is very difficult to ask the end user to specify a particular value for c. Moreover, most methodological improvements in classification occur without a specific application in mind, where, again, it is difficult to set a reasonable cost. It is more realistic to specify a *distribution* instead, $w(c)$, over different values of c, capturing the end user's uncertainty about the exact values of the costs:

$$L_w = \int_c L(c; T_c)w(c)dc$$

This notion of *averaged minimum cost-weighted loss* allows to formulate a criticism of the AUC which in turn motivates the H measure. It is shown in [6] that the AUC may be written in terms of L_w, albeit with a choice of prior $w(c)$ that depends on the classifier. In effect, when interpreted in terms of costs, the AUC evaluates different classifiers using different metrics. In contrast, employing a common prior for all classifiers (see next section for a default choice) yields the H measure:

$$H = 1 - \frac{L_w}{L_w^{\max}}$$

where we have normalised by the maximum value L_w can take (i.e.,$L_w^{\max}$ is the averaged MWL of the trivial classifier), and subtracted from 1 so that higher values indicate better performance. The suggested choice of prior is discussed in the following section.

2.7 Choosing the prior

Although for the H measure the distribution $w(c)$ is fixed – in the sense that any given researcher should choose a distribution and use that for all classifiers being applied on the given problem – this distribution is meant to reflect the structure of each given application domain, and hence may very well differ across different research areas. However, for purposes of global comparison, [2] proposed a certain universal standard which we now describe.

In many problems the class sizes are extremely unbalanced. In such cases, it would be rare that one would want to treat the two classes symmetrically.

Instead, one would probably want to treat misclassifications of the smaller class as more serious than those of the larger class, since if they are treated as of equal severity then very little loss would be made by assigning everything to the larger class. This asymmetry can be seen to underlie the KS statistic, which is a simple linear transformation of the MWL when $c = \pi_1$ and $1 - c = \pi_0$. This latter choice precisely has the effect of turning a missed instance of the rare class into a graver type of error. The proposed default prior for the H measure relies on the same intuition, but rather than committing to a single fixed value of c, it considers the entire range of $c \in [0, 1]$, but places maximum weight on the choice $c = \pi_1$. In mathematical terms, $w(c)$ is a Beta distribution whose mode is at $c = \pi_1 \approx 0.3$ for the Pima dataset. We may plot the prior by the `plotROC` command (see Figure 3(a)). Note that the `hmeasure` package internally employs an alternative parameterisation of the cost, known as the *severity ratio*, which considers the ratio between the two costs, $\mathrm{SR} = \frac{c}{1-c}$, so that $\mathrm{SR} = 1$ represents symmetric costs. The default severity ratio may of course be changed to a user-specified setting.

Returning to our example, we observe that, using the default $c = \pi_1$, we obtain:

```
> results$metrics[c('H','KS','ER','FP','FN')]

            H        KS        ER FP FN
LDA 0.4802968 0.6335723 0.2217195 25 24
kNN 0.3707186 0.4920941 0.2262443 25 25
```

The two classifiers have very similar error rates, as they differ on a single error. However, this error is of the graver type, since the positive (disease) class is less prevalent:

```
> summary(pima.test[,8])

 No Yes
147  74
```

This is reflected in the KS statistic and the H measure which both clearly favour the LDA classifier. An interesting point of comparison is the behaviour of the H measure for $c = 0.5$:

```
> results.SR1 <- HMeasure(
+   true.labels, data.frame(LDA=scores.lda,kNN=scores.knn),severity.ratio=1)
> results.SR1$metrics[c('H','KS','ER','FP','FN')]

            H        KS        ER FP FN
LDA 0.4401067 0.6335723 0.2217195 25 24
kNN 0.3463211 0.4920941 0.2262443 25 25
```

Despite the fact that both the H measure and the ER encode the assumption of symmetric costs, the H measure continues to clearly favour the LDA to the

```
> par(mfrow=c(2,1))
> plotROC(results,which=2)
> plotROC(results.SR1,which=2)
```

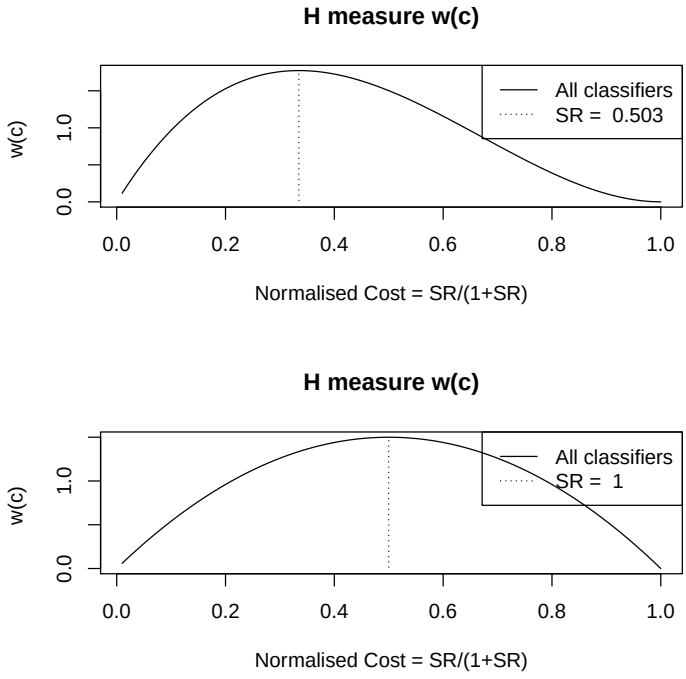

Figure 3: The prior employed by the H measure for $c = \pi_1$ (left) and $c = 0.5$ (right).

k-NN classifier. It is immediately evident from Figure 1 that indeed LDA is equivalent to k-NN in the early part of the ROC, but clearly outperforms it in the latter part. Presumably, the choice of threshold corresponding to $c = 0.5$ happens to be in a region where the two perform equally well. Nevertheless, the uncertainty of the H-measure allows it to cast a 'wider net' (see Figure 3(b)), taking into account parts of the ROC space either side of this specific choice of threshold, and hence correctly concludes that LDA should be preferred in this instance. To conclude this discussion, we now use the `plotROC` function to graphically display the incoherency of the AUC when interpreted in terms of costs. Recall that the AUC averages over thresholds under the assumption that all values of the TPR (sensitivity) are equally likely. This uniform prior over the sensitivity induces, for each dataset and classifier, a prior over costs, which we display in Figure 4. The prior can be seen to differ across the two classifiers [6]. This is incoherent under the cost interpretation.

```
> plotROC(results,which=3)
```

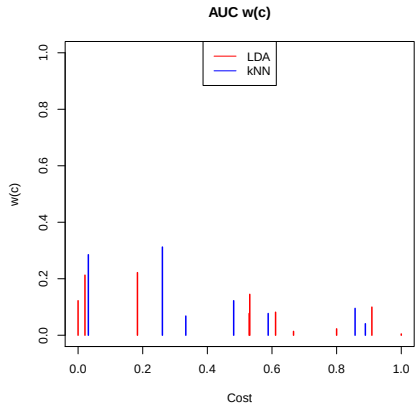

Figure 4: Cost distribution implicitly employed by the AUC for the k-NN and LDA classifier against the Pima dataset.

3 Conclusion

The trade-off between false positives and false negatives inherent in binary classification tasks complicates any attempt at providing a scalar summary of classification performance. Several measures have been proposed in the literature to overcome this difficulty, prominent among which is the Area Under the ROC Curve (AUC), implementations of which exist in almost all statistical programming languages. The AUC has recently come under criticism for handling the aforementioned tradeoff in a fundamentally incoherent manner. In this report we focus in one particular criticism that appeared in [6], whereby the AUC is shown to treat the relative severities of misclassifications differently when different classifiers are used, but note here that more recently other related criticisms have appeared (e.g., [9]). A coherent alternative was proposed in [6], known as the H measure, which can accommodate expert knowledge regarding misclassification costs, whenever that is available. The `{hmeasure}` package computes and reports the H measure of classification performance, alongside most commonly used alternatives, including the AUC. The package also provides convenient plotting routines that yield insights into the differences and similarities between the various metrics.

References

[1] M. Gönen. Analyzing receiver operating characteristic curves with sas. Technical report, SAS Institute: Cary, NC., 2007.

[2] D.J. Hand and C. Anagnostopoulos. A better Beta for the H measure of classification performance Preprint, arXiv:1202.2564v1

[3] D.J. Hand. *Construction and assessment of classification rules.* Wiley: Chichester., 1997.

[4] D.J. Hand. Measuring diagnostic accuracy of statistical prediction rules. *Statistica Neerlandica*, 53:3–16., 2001.

[5] D.J. Hand. Good practice in retail credit scorecard assessment. *Journal of the Operational Research Society*, 56:1109–1117, 2005.

[6] D.J. Hand. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning*, 77:103–123, 2009.

[7] D.J. Hand. Evaluating diagnostic tests: the area under the ROC curve and the balance of errors. *Statistics in Medicine*, 29:1502–1510, 2010.

[8] D.J. Hand. Assessing the performance of classification, signal detection, and diagnostic methods. Technical report, Department of Mathematics, Imperial College, London, 2011.

[9] D.J. Hand and C. Anagnostopoulos. When is the area under the roc curve an appropriate measure of classifier performance? Technical report, Department of Mathematics, Imperial College, London, 2011.

[10] D.J. Hand, C. Whitrow, N.M. Adams, P. Juszczak, and D. Weston. Performance criteria for plastic card fraud detection tools. *Journal of the Operational Research Society*, 59:956–962, 2008.

[11] W.J. Krzanowski and D.J. Hand. *ROC curves for continuous data.* Chapman and Hall, 2009.

[12] M.S. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction.* Oxford University Press: Oxford, 2003.

[13] F. Provost and T. Fawcett Robust classification systems for imprecise environments. In: *Proceedings of the THird International Conference on Knowledge Discovery and Data Mining*, 43–48 AAAI Press, Menlo Park, CA

[14] X-H. Zhou, N.A. Obuchowski, and D.K. McClish. *Statistical Methods in Diagnostic Medicine.* Wiley: New York, 2002.