Artificial Intelligence

academic year 2025/2026

Giorgio Fumera

Pattern Recognition and Applications Lab
Department of Electrical and Electronic Engineering
University of Cagliari (Italy)

Machine Learning

Performance evaluation

How to evaluate generalisation capability?

The **learning from examples** approach to supervised classification aims at finding a classifier exhibiting a good **generalisation capability**, i.e., capable of correctly predicting the label of "most" instances (e.g., e-mails or character images) **after deployment**, **during operation**, through the analysis of a **limited** set of examples collected **during design**.

Two issues emerge (among others):

- how can the **effectiveness** of a classifier in predicting the label of any instance be formally defined, i.e., what **performance measure** can be used?
- how can one estimate generalisation capability (using any given measure) **before** a classifier is deployed, considering that it refers to **any** possible instance, including **unseen** ones, i.e., instances not part of the available training set?

Measures of classification performance

Different performance measures can be used, depending on the application.

- The simplest one is the **error rate**, defined as the fraction of **misclassified** instances out of a given set
- An equivalent measure is the **classification accuracy**, the fraction of **correctly classified** instances
- More complex measures can be defined based on the **confusion matrix** of a classifier. For an m-class problem it is defined as a $m \times m$ matrix C whose elements c_{ij} are the number of instances of class j, out of a given set, labelled as belonging to class i

The confusion matrix of a classifier

Example of confusion matrix for a handwritten digit classifier ($m = 10$ classes, labelled as `0`, `1`, ..., `9`), computed on a hypothetical set of 1000 instances (100 for each class) whose correct class label is **known**:

		true class									
		0	1	2	3	4	5	6	7	8	9
predicted class	0	93	1	0	0	0	1	0	0	1	0
	1	0	95	4	1	0	1	1	0	0	1
	2	2	0	91	0	0	0	0	1	0	0
	3	0	1	0	98	0	0	1	0	0	1
	4	1	0	1	0	100	0	0	0	0	0
	5	0	0	0	0	0	94	0	0	1	0
	6	2	1	3	0	0	1	97	1	0	2
	7	0	0	0	0	0	0	0	96	0	0
	8	1	2	1	0	0	2	1	1	97	4
	9	1	0	0	1	0	1	0	1	1	92

Performance measures for two-class problems

Many two-class problems are related to **detection** tasks in input data, e.g.:

- spam filtering: detecting spam e-mails
- medical diagnosis: detecting a given disease (e.g., from blood tests, X-ray, body scan or MRI)
- malware detection (e.g., in PDF documents or mobile applications)

In this kind of problem the class of "objects" to be detected (e.g., spam e-mails, body scans of patients affected by a given disease, or PFD documents containing malware) is usually named **positive**, whereas the other (e.g., legitimate e-mails) is named **negative**.

Performance measures for two-class problems

The entries of the confusion matrix of “positive” vs “negative” classification problems are named:

- **true positives** (TP): number of positive instances (out of a given set) correctly labelled by a given classifier as positive
- **false positives** (FP): number of negative instances misclassified as positive
- **true negatives** (TN): number of negative instances correctly labelled as negative
- **false negatives** (FN): number of positive instances misclassified as negative

		true class	
		`positive`	`negative`
predicted class	`positive`	TP	FP
	`negative`	FN	TN

Performance measures for two-class problems

From the confusion matrix, the **error rate** is defined as:

$$\frac{FP + FN}{TP + TN + FP + FN}$$

Note that FP and FN errors are equally weighted.
However, depending on the application they may have different consequences, e.g.:

- misclassifying a spam e-mail as legitimate (FN) can be annoying for the user (who will have to manually remove them from the inbox), but misclassifying a legitimate e-mail as spam (FP) may result in missing important communications
- diagnosing a healthy patient as affected by a serious disease (FP) can cause troubles, but can be repaired through further tests; diagnosing a sick patient as healthy (FN) may instead prevent him or her to promptly receive the necessary treatment

Performance measures for two-class problems

Classifiers like Decision Trees output a class label. Many other kinds of classifiers (such as artificial neural networks) output instead a **score**, e.g., a real number $s \in [0, 1]$.

In the latter case, a **decision threshold** t has to be set to turn the score s into a predicted class label y, e.g.:

$$\begin{array}{lll} \text{if } s \geq t & \text{then} & y = \text{positive} \\ \text{if } s < t & \text{then} & y = \text{negative} \end{array}$$

It is easy to see that, by increasing the value of t, the number of FP errors **decreases**, whereas the number of FN errors **increases**.

The value of the threshold t can be chosen to attain a **trade-off** between FP and FN errors, depending on the application at hand.

Performance measures for two-class problems

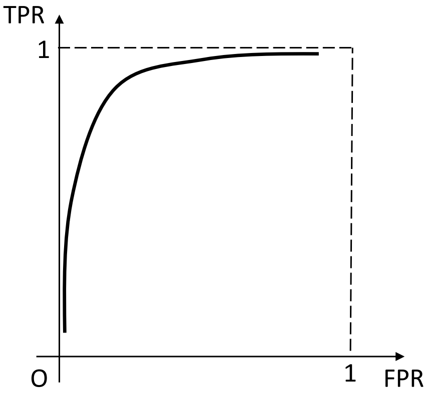
A widely used tool for evaluating classifier performance in detection tasks is the **Receiver Operating Characteristic (ROC) curve**, which depicts the behaviour of the **true positive rate** (TPR) vs the **false positive rate** (FPR), e.g., as a function of the decision threshold:

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{TN + FP}$$

- TPR: fraction of **positive** instances **correctly** labelled as **positive** (also known as **sensitivity** or **recall**)
- FPR: fraction of **negative** instances **wrongly** labelled as **positive** (the value $1 - FPR$, is also known as **specificity**)

Performance measures for two-class problems

An example of ROC curve:



Expected classification cost

In some applications it may be possible to **quantify** the consequence of misclassifications in some unit of measurement (i.e., a monetary unit).

In this case each entry c_{ij} of the confusion matrix can be associated with a **classification cost** λ_{ij}:

- the cost of correct classifications λ_{ii} $(i = 1, \ldots, m)$ is usually 0
- the cost of misclassifications λ_{ij} $(i, j = 1, \ldots, m,\ i \neq j)$ can be different for different pairs of classes, if errors on different classes have different consequences

Expected classification cost

The performance measure that can be used in this kind of application is the **expected classification cost**, which can be estimated from the confusion matrix on a set of n labelled instances as:

$$\frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_{ij} c_{ij}}{n}$$

Note that, if all misclassifications incur the **same** cost, $\lambda_{ij} = \lambda,\ i \neq j$, and the cost of correct classifications is **zero** $(\lambda_{ii} = 0,\ i = 1, \ldots, m)$, then the expected cost is proportional to the **error rate**:

$$\frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_{ij} c_{ij}}{n} = \lambda \frac{\sum_{i=1}^{m} \sum_{j=1, j\neq i}^{n} c_{ij}}{n} = \lambda \times (\text{error rate})$$

How to estimate generalisation capability?

Any performance measure can be estimated from a set of **labelled** examples.

A straightforward solution is to use the same **training set** which was previously used to train the classifier: the corresponding estimate is called **resubstitution error**.

However, the resubstitution error **overestimates** the true performance measure, since learning algorithms are designed to **minimise** the error on training examples: in fact, this can even cause **over-fitting** (very few or no errors on training examples, but low generalisation capability).

A more accurate estimate of generalisation capability is therefore necessary.

The hold-out technique

A simple solution is to **randomly** split the set of labelled examples collected for classifier design into two subsets:

- one subset (e.g., 70% of the available examples) will be used by the learning algorithm as the **training set**
- the remaining examples, called **testing set**, will be used to estimate the generalisation capability of the trained classifier

This technique, named **hold-out**, provides a more reliable estimate of generalisation capability to instances **unseen** by the learning algorithm. A more reliable estimate can be obtained by averaging the ones obtained from multiple random splittings of the available examples.

However, a classifier trained on a **smaller** training set tends to exhibit a **lower** generalisation capability. The classifier to be deployed can therefore be re-trained on the **whole** set of labelled examples, but this means that the hold-out technique is likely to provide an **underestimate** of the corresponding generalisation capability.

The cross-validation technique

To mitigate the drawbacks of the hold-out technique, the ***k*-fold cross-validation** procedure can be used:

1. the available examples $\mathcal{T}$ are randomly split into k (typically, $k = 5$ or 10) **disjoint** and **equally sized** subsets $\mathcal{T}_1, \ldots, \mathcal{T}_k$, called **folds**
2. for $i = 1, \ldots, k$:
   2.1 train a classifier using the examples in $\mathcal{T} - \mathcal{T}_i$
   2.2 estimate its performance measure e_i on $\mathcal{T}_i$
3. estimate the generalisation capability as the average performance across the k folds, $\left(\sum_{i=1}^{k} e_i\right)/k$

The classifier to be deployed is then trained on the **whole** $\mathcal{T}$.

The larger k, the more accurate the estimate of its generalisation capability. However, also the processing cost increases, due to k executions of the learning algorithm.

The leave-one-out technique

The limit case of k-fold cross-validation is when $k = n$, being n the size of $\mathcal{T}$.

This technique is known as **leave-one-out**, since at each iteration just **one** example of $\mathcal{T}$ is left out the training set, to be used as the testing set.

Leave-one-out provides the most accurate estimate of generalisation capability among all possible values of k in k-fold cross-validation, but also exhibits the highest processing cost.

Probabilistic view of supervised classification

During classifier **design**, instances that will be processed by the trained classifier are **unknown**, and can be viewed as joint random variables (X, Y) (where X denotes the attribute vector and Y the class label) from an **unknown** probability density function $P(X, Y)$.

Also the label that will be predicted by the trained classifier $h(\cdot)$ on a random instance can be viewed as a random variable $\hat{Y} = h(X)$.

Accordingly, the performance measure of a trained classifier is defined as the **expected value** of a **loss function** $\ell(\hat{Y}, Y)$ defined over the predicted and true class label of a random instance, with respect to the probability density $P(X, Y)$.

Probabilistic view of supervised classification

For instance, the simplest loss function is the **0–1 loss**:

$$\ell(\hat{Y}, Y) = \begin{cases} 0, & \text{if } \hat{Y} = Y, \\ 1, & \text{if } \hat{Y} \neq Y \end{cases}$$

The corresponding expected value is the **misclassification probability** $P(h(X) \neq Y)$.

Probabilistic view of supervised classification

Since $P(X, Y)$ is unknown, in practice any performance measure can only be **estimated**, using the **frequentist** definition of probability, from a given set of n labelled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$.

For instance, the **error rate** mentioned above is a frequentist estimate of the error probability:

$$P(h(X) \neq Y) \approx \frac{\sum_{i=1}^n \ell(h(\mathbf{x}_i), y_i)}{n} \ ,$$

where:

$$\ell(h(\mathbf{x}_i), y_i) = \begin{cases} 0, & \text{if } h(\mathbf{x}_i) = y_i, \\ 1, & \text{if } h(\mathbf{x}_i) \neq y_i \end{cases}$$