2. Classification Evaluation

Consider a typical classification problem
$f: X \to Y$

- D: probability distribution over X
- S: Sample of n instances drawn from X (according to distribution D) and for which we know $f(x)$

Performance evaluation in classification is based on ACCURACY or ERROR RATE.

Let's define two errors:

- the TRUE ERROR of hypothesis h wrt target function f and distribution D is the probability that I will miss classify an instance drawn at random according to D:

$$\boxed{error_D(h) = \Pr_{x \in D}[f(x) \neq h(x)]}$$

Make at run time

- The SAMPLE ERROR of h wrt target function and sample S is the proportion of examples h miss classifies:

$$\boxed{error_S(h) = \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))}$$

Computed from the datasets

$$\delta(f(x)/h(x)) = \begin{cases} 1 & \text{if } f(x)/h(x) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The true error is more important, is what we want to evaluate, because we are interested in classify a new instance.

The TRUE ERROR cannot be computed since for a new instance we do not know $f(x)$.

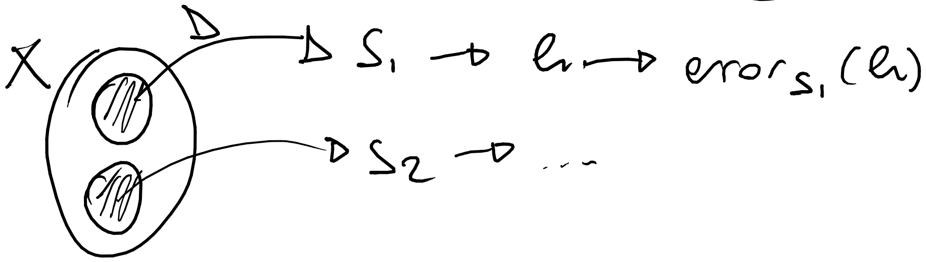
<u>Note</u>: the goal of a learning system is to be accurate in $h(x)$ $\forall x \notin S$.

$$ACCURACY(h) = 1 - ERROR(h)$$

If $ACCURACY_S(h)$ is very high and $ACCURACY_D(h)$ is very low, then our system would not be useful.

We now understand how to compute the sample error.

Let's understand the following:



generate a dataset by the set X and compute a solution h, for which we compute the error IS LIKE ROLLING A DICE, because we chose a particular set of instances, with a particular algorithm. IF WE ROLL THE DICE AGAIN we have a new solution and a NEW ERROR.

In general, given a random variable the expected value is defined as the infinite average of possible outcomes

$$\boxed{BIAS = E[error_S(h)] - error_D(h)}$$

We want that the bias is zero and if it is $error_D(h) = E[error_S(h)]$.

How to compute $error_S(h)$:

- Partition the dataset D ($D = T \cup S$ and $T \cap S = \emptyset$)

  We use a training set to learn h and a set to compute the hypothesis evaluation, this is an unbiased estimation

  To be UNBIASED the set for which we compute the error should be independent from the one used to compute the hypothesis.

- Compute h by using T
- Evaluate $error_S(h) = \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))$

This process should be repeated many times (we do not rolling a dice once), in principle you should repeat it infinite times.

Confidence intervals

If : - S contains n sample
- $n \geq 30$

z_N increases while percentage increases

Then

with approximately $N\%$ possibility, $error_D(h)$ lies in interval:

$$error_S(h) \pm \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

where

$N\%$	50%	68%	80%	...	99%
z_N	0.67	1.28	1.64	...	2.58

Given two hypothesis we may want to compare them. We have no GUARANTEES that if one h is better on S is also better on D.

$d = error_D(h_1) - error_D(h_2)$

$\hat{d} = error_S(h_1) - error_S(h_2)$

$E[\hat{d}] = d$ ← $\hat{d}$ is an unbiased estimator for d iff h_1, h_2, S_1 and S_2 are INDEPENDENT from each other

Hypothesis $h \in H$ OVERFITS training data if there is an alternative hypothesis $h' \in H$ such that $error_S(h) < error_S(h')$ AND $error_D(h) > error_D(h')$

EVALUATION of the output of LEARNING ALG.

The best way to evaluate the output of a learning algorithm is to use a technique called k-fold cross validation

K-FOLD CROSS VALIDATION

- Partition dataset D into k DISJOINT set $S_1 \dots S_k$ $(|S_i| > 30)$
- For $i = 1 \dots k$

solve k times our learning problem

use S_i as test set, and the remaining data as training set T_i

$T_i \leftarrow \{D - S_i\}$

$h_i \leftarrow L(T_i)$ [solution of the learning algorithm]

$\delta_i \leftarrow error_{S_i}(h_i)$

- Return: $error_{L,D} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$, $accuracy_{L,D} = 1 - error_{L,D}$

We can use it also for comparing two different learning algorithm L_A and L_B, if the value returned $\bar{\delta} < 0$ then L_A is better than L_B.

Note: the size of the training set $\frac{k-1}{k}|D|$ (most of the size)

OTHER PERFORMANCE METRICS

"Is accuracy always a good performance metric?"

Let's suppose to have a binary classification $f: X \to \{-, +\}$ with training set D containing 90 % of positive samples.

I have two hypothesis h_1 and h_2, h_1 has accuracy of the 90 % and h_2 has accuracy 85 %, which is the best?

Well, in general h_1, but h_1 may be a solution that always return + and h_2 a result of a classification algorithm.

In some cases, accuracy is not enough to assess the performance of a classification method.

Unbalanced datasets are very common in problems related to anomaly detection.
(e.g. malware analysis, fraud detection, medical tests, etc.)

PREDICTED CLASS

TRUE CLASS	YES	NO
YES	TP	FN
NO	FP	TN

$$\text{Error rate} = |\text{error}| / |\text{instances}|$$

$$= \frac{(FN + FP)}{(FP + FN + TP + TN)}$$

$$\text{accuracy} = 1 - \text{Error rate} = \frac{TP + TN}{FP + FN + TP + TN}$$

$$P = \frac{TP}{(TP + FP)}$$

predicted positive

PRECISION: ability to avoid FP

$$R = \frac{TP}{TP + FN}$$

real positive

RECALL: ability of the sys to avoid FN

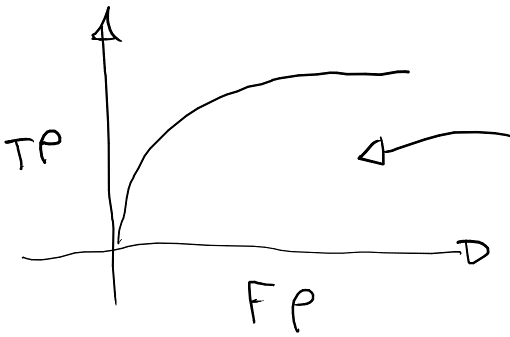
$$F_1 - \text{score} = \frac{2PR}{P + R}$$

Let's consider the system of autonomous car that detects pedestrian and brakes!

- FP: there is no person, but the system sees a person
- FN: there is a person but the system does not see it.

False positive and false negative have different relevance, YOU WANT HIGH RECALL!

ROC CURVE: you can plot the relation between TP and FP, varying some parameter of the algorithm.



ROC Area: Area under the curve

CONFUSION MATRIX:

In a classification problem with many classes, we can compute how many times an instance of C_i is classified as C_j.

MAIN DIAGONAL CONTAINS ACCURACY FOR EACH CLASS.

By CM you can understand which classes are usually missclassified.