Performance evaluation

Defining the error

In order to see if our hypothesis h^* is good we want to find a way to compute its error rate and accuracy. In doing so, we must take in account that our hypothesis is trained on an input space which generally follows a certain distribution. A distribution plots every value of our sample together with its probability of finding it inside the population.

That said we define two kinds of error:

- **True error** is the probability that when extracting a sample from the distribution our hypothesis will give an incorrect prediction: $error_D(h) = P_{x \in D}[f(x) \neq h(x)]$
- **Sample error** is the error defined on the testing dataset for which we already know correct solutions. This is not a probability and is computed as: $error_S(h) = \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))$

Accuracy is defined as:

$$accuracy(h) = 1 - error(h)$$

In order to have a good estimate of the error, we want the sample error to be as close as possible to the true error. This concept is expressed as the *estimation bias*, which shows the difference between the two errors:

$$estimation\ bias = E[error_S(h)] - error_D(h)$$

Note that the sample error is expressed as an expected value, since it is a random variable depending on the sample that has been extracted. We always have to remember that the true expected value of sample error can't be computed, since we would need infinite samples, that corresponds to already having a complete knwoledge of the problem.

The solution in order to have zero estimation bias is making shure that our test and train set will be disjoint. Since we have to work with randomness, the best we can do is keeping a low variance of the estimation of the error. In order to do this we use the concept of confidence intervals:

With approximately $N\%$ probability the $error_D(h)$ falls in to the interval:

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

Where z_N is the coefficient which describes the grade of precision we want.

The only way to keep a good confidence while having a small interval is by increasing n, the number of samples.

The steps of the error computation are:

1. Partition the dataset in train test T and test set S such that: $D = T \cup S, T \cap S = \emptyset, |T| = \frac{2}{3}|S|$
2. Compute an hypothesis using T
3. Evaluate $error_S(h) = \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))$

Notice that $error_S(h)$ is a random variable since it depends on the split. This error is defined as an unbiased estimator for $error_D(h)$

For a medium sized dataset the size of the split is: $\frac{2}{3}\ train, \frac{1}{3} split$.

Evaluating our solution

Once we have an hypothesis, we want to test it against the possible problems that we can have during evaluation.

A major one can be *Overfitting*. This error happens when our solution gets too specialized on predicting the input space, but results weak when testing. Formally we say that an hypothesis $h \in H$ overfits training data if exists another hypothesis $h' \in H$ such that:

$$error_S(h) < error_S(h') \ \wedge \ error_D(h) > error_D(h')$$

In order to improve our confidence on the avaluation of error we use a technique calles *K-fold cross validation.* It consists on repeating the process of error evaluation multiple times, each time on a different split.

In practice we divide our dataset in k partitions $S_1, S_2, ..., S_k$. At each iteration we use S_i for training and every other for testing, obtaining k different errors. At the end we compute:

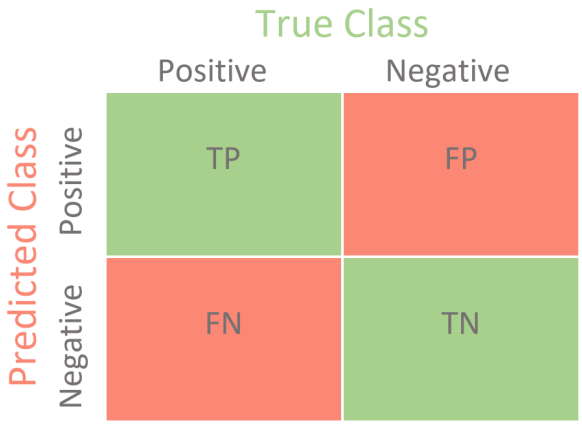
$$error_{L,D} = \frac{1}{K} \sum_{i=1}^{K} \delta_i$$

Notice with K-fold we are evaluating the the learning method. Similarly we can compute accuracy of the learning method as. $accuracy_{L,D} = 1 - error_{L,D}$.

But accuracy not always tells us the whole story. In fact having an unbalanced dataset could still cause us trouble. Take this example:

We have a two classes dataset $D = \{+, -\}$ where negative samples make the 90% of it. From this we train two hypothesis h_1, h_2 obtaining that h_1 is 90% accurate and h_2 is 85% accurate. We might say that the first one is always better, and we'd be wrong. For example h_1 might just be guessing that every sample is negative.

We need to introduce a more accurate metric: *the confusion matrix*. This matrix shows us the number of true guesses vs the number of false guesses. For example if we have two classes, the corresponding confusion matrix is:



The values on the main diagonal represent the correct guesses. From this matrix we can extract other important informations, namely:

$Recall = \frac{|true\ positives|}{|real\ positives|} = \frac{TP}{TP+FN}$

$Precision = \frac{|true\ positives|}{|predicted\ positives|} = \frac{TP}{TP+FP}$

Recall tells us how good is our system in avoiding false negatives, while precision does the same for false positives. Depending on the context, we might be more interested in having a high precision rather than recall and vice versa. In general we want them to be balanced, so we use the F1-score, which accounts for both:

$$F1 - score = \frac{2(Precision \times Recall)}{Precision + Recall}$$