AdaBoost and the Super Bowl of Classifiers
A Tutorial Introduction to Adaptive Boosting

Raúl Rojas
Computer Science Department
Freie Universität Berlin
Christmas 2009

Abstract

This note provides a gentle introduction to the AdaBoost algorithm used for generating strong classifiers out of weak classifiers. The mathematical derivation of the algorithm has been reduced to the bare essentials.

1 Motivation

Assume that you are working on a two-class pattern recognition problem for which you are given a large pool of classifiers (which we call experts). You want to submit a still better classifier for a pattern recognition competition, the Super Bowl of Classifiers. Therefore, you want to put together a "dream team" of experts – let's say a combination of eleven classifiers extracted from the pool. For a given pattern x_i each expert classifier k_j can emit an opinion $k_j(x_i) \in \{-1, 1\}$ and the final decision of the committee K of experts is $\text{sign}(C(x_i))$, the sign of the weighted sum of expert opinions, where

$$C(x_i) = \alpha_1 k_1(x_i) + \alpha_2 k_2(x_i) + \cdots + \alpha_{11} k_{11}(x_i),$$

and $k_1, k_2, \ldots, k_{11}$ denote the eleven experts selected from the pool of classifiers. The constants $\alpha_1, \alpha_2, \ldots, \alpha_{11}$ are the weights we assign to the opinion of each expert in the committee. Remember that every expert k_j just answers "yes" $(+1)$ or "no" (-1) to a classification problem. The arrangement discussed here is a linear combination of classifiers followed by a nonlinear decision (the sign function).

The AdaBoost (*adaptive boosting*) algorithm was proposed in 1995 by Yoav Freund and Robert Shapire as a general method for generating a strong classifier out of a set of weak classifiers [1], [3]. AdaBoost works even when the classifiers come from a continuum of potential classifiers (such as neural networks, linear discriminants, etc.) However, for the sake of simplicity, let us assume that the pool of experts is finite, that it contains L classifiers, and that it is provided as input to AdaBoost. This is the case, for example, with the well-known method of face recognition introduced by Viola and Jones [2].

2 Scouting

If we want to go to the classifiers competition we have to take care of: a) scouting prospective team members, b) drafting them, and c) assigning a weight to their contribution to the team.

Scouting is done by testing the classifiers in the pool using a training set T of N multidimensional data points x_i. For each point x_i we have a label $y_i = 1$ or $y_i = -1$. We test and rank all classifiers in the expert pool by charging a cost e^{β} any time a classifier fails (a miss), and a cost $\mathrm{e}^{-\beta}$ every time a classifier provides the right label (a success or "hit"). We require $\beta > 0$ so that misses are penalized more heavily than hits. It might seem strange to penalize a hit with non-zero cost, but as long as the penalty of success is smaller than the penalty for a miss ($\mathrm{e}^{-\beta} < \mathrm{e}^{\beta}$) everything is fine (see Problem 1). This kind of error function different from the usual squared Euclidian distance to the classification target is called an exponential loss function. AdaBoost uses exponential error loss as error criterion.

When we test the L classifiers in the pool, we build a matrix S (S for scouting) in which we record the misses (with a 1) and hits (with a zero) of each classifier. Row i in the matrix is reserved for the data point x_i. Column j is reserved for the j-th classifier in the pool:

Classifiers

	1	2	$\cdots$	L
x_1	0	1	$\cdots$	1
x_2	0	0	$\cdots$	1
x_3	1	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$		$\vdots$
x_N	0	0	$\cdots$	0

In the example above, the first classifier succeeds with the data points x_1, x_2 and x_N. It fails with the data point x_3. It is easy to read from the table for which data points the other classifiers fail or succeed.

The main idea of AdaBoost is to proceed systematically by extracting one classifier from the pool in each of M iterations. The elements in the data set are weighted according to their current relevance (or urgency) at each iteration. At the beginning, all elements are assigned the same weight (just 1, or $1/N$ if we want to have a total sum of 1 for all weights). As the drafting progresses, the more difficult examples, that is, those where the committee still performs badly, are assigned larger and larger weights. The drafting process concentrates in selecting new classifiers for the committee focusing on those which can help with the still misclassified examples. It would be superfluous to draft a classifier which always, or almost always, produces the same opinion as a classifier already drafted. If we wanted to draft a classifier twice, we could just as well duplicate its weight. The best "team players" are those which can provide new insights to the committee. Classifiers being drafted should complement each other in an optimal way. Not everybody can be the quarterback.

3 Drafting

In each iteration we need to rank all classifiers, so that we can select the current best out of the pool. At the m-th iteration we have already included $m-1$ classifiers in the committee and we want to draft the next one. The current linear combination of classifiers is

$$C_{(m-1)}(x_i) = \alpha_1 k_1(x_i) + \alpha_2 k_2(x_i) + \cdots + \alpha_{m-1} k_{m-1}(x_i)$$

and we want to extend it to

$$C_m(x_i) = C_{(m-1)}(x_i) + \alpha_m k_m(x_i).$$

At the first iteration ($m = 1$), $C_{(m-1)}$ is the zero function. We define the total cost, or total error, of the extended classifier as the exponential loss

$$E = \sum_{i=1}^{N} \mathrm{e}^{-y_i(C_{(m-1)}(x_i) + \alpha_m k_m(x_i))}$$

where α_m and k_m are yet to be determined in an optimal way. Since our intention is to draft k_m we rewrite the above expression as

$$E = \sum_{i=1}^{N} w_i^{(m)} \mathrm{e}^{-y_i \alpha_m k_m(x_i))} \tag{1}$$

where

$$w_i^{(m)} = \mathrm{e}^{-y_i C_{(m-1)}(x_i)} \tag{2}$$

for $i = 1, \ldots, N$. In the first iteration $w_i^{(1)} = 1$ for $i = 1, \ldots, N$. During later iterations, the vector $w^{(m)}$ represents the weight assigned to each data point in the training set at iteration m. We can split the sum in Eq. 1 into two sums

$$E = \sum_{y_i = k_m(x_i)} w_i^{(m)} \mathrm{e}^{-\alpha_m} + \sum_{y_i \neq k_m(x_i)} w_i^{(m)} \mathrm{e}^{\alpha_m}$$

This means that the total cost is the weighted cost of all hits plus the weighted cost of all misses. Writing the first summand as $W_c \mathrm{e}^{-\alpha_m}$ and the second as $W_e \mathrm{e}^{\alpha_m}$, we simplify the notation to

$$E = W_c \mathrm{e}^{-\alpha_m} + W_e \mathrm{e}^{\alpha_m} \tag{3}$$

For the selection of k_m the exact value of $\alpha_m > 0$ is irrelevant, since for a fixed α_m minimizing E is equivalent to minimizing $\mathrm{e}^{\alpha_m} E$ and because

$$\mathrm{e}^{\alpha_m} E = W_c + W_e \mathrm{e}^{2\alpha_m}$$

Since $\mathrm{e}^{2\alpha_m} > 1$, we can rewrite the above expression as

$$\mathrm{e}^{\alpha_m} E = (W_c + W_e) + W_e(\mathrm{e}^{2\alpha_m} - 1)$$

Now, $(W_c + W_e)$ is the total sum W of the weights of all data points, that is, a constant in the current iteration. The right hand side of the equation is minimized when at the m-th iteration we pick the classifier with the lowest total cost W_e (that is the lowest rate of weighted error). Intuitively this makes sense, the next draftee, k_m, should be the one with the lowest penalty given the current set of weights.

4 Weighting

Having picked the m-th member of the committee we need to determine its weight α_m. From Eq. 3 we immediately see that

$$\frac{\mathrm{d}E}{\mathrm{d}\alpha_m} = -W_c \mathrm{e}^{-\alpha_m} + W_e \mathrm{e}^{\alpha_m}$$

Equating this expression to zero and multiplying by e^{α_m} we obtain

$$-W_c + W_e \mathrm{e}^{2\alpha_m} = 0$$

The optimal α_m is thus:

$$\alpha_m = \frac{1}{2} \ln \left(\frac{\mathrm{W_c}}{\mathrm{W_e}} \right)$$

Remembering that W is the total sum of weights, we can rewrite the expression above as

$$\alpha_m = \frac{1}{2} \ln \left(\frac{\mathrm{W} - \mathrm{W_e}}{\mathrm{W_e}} \right) = \frac{1}{2} \ln \left(\frac{1 - \mathrm{e_m}}{\mathrm{e_m}} \right)$$

Where $e_m = W_e/W$, is the percentage rate of error given the weights of the data points.

5 Pseudocode

Given a training set T of data points x_i and their labels y_i in a two class problem (+1,-1), we assign initial weights $w_i^{(1)} = 1$ to all data points x_i. We want to draft a committee of M classifiers from a pool of classifiers. We perform M iterations. At each iteration we call W the sum of weights of all data points, and W_e the sum of weights of those data points where the classifier being considered yields the wrong label.

AdaBoost
For $m = 1$ to M

1. Select and extract from the pool of classifiers the classifier k_m which minimizes
$$W_e = \sum_{y_i \neq k_m(x_i)} w_i^{(m)}$$
2. Set the weight α_m of the classifier to
$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \mathrm{e_m}}{\mathrm{e_m}} \right)$$
where $e_m = W_e / W$
3. Update the weights of the data points for the next iteration. If $k_m(x_i)$ is a miss, set
$$w_i^{(m+1)} = w_i^{(m)} \mathrm{e}^{\alpha_m} = w_i^{(m)} \sqrt{\frac{1 - e_m}{e_m}}$$
otherwise
$$w_i^{(m+1)} = w_i^{(m)} \mathrm{e}^{-\alpha_m} = w_i^{(m)} \sqrt{\frac{e_m}{1 - e_m}}$$

Some comments about this pseudocode formulation of AdaBoost are pertinent. The pool of classifiers in Step 1 can be substituted by a family of classifiers, one of whose members is trained to minimize the error function given the current weights. That is, the pool of classifiers does not need to be given in advance, it only needs to ideally exist. If indeed a finite set of classifiers is given, we only need to test the classifiers once for each data point. The scouting matrix S can be reused at each iteration, multiplying the transposed vector of weights $w^{(m)}$ with S in order to determine W_e for each classifier in the pool.

Regarding the weights, it is possible to reformulate the weight update step so that only misses lead to a weight modification (see Problem 3).

Note that the weight vector $w^{(m)}$ is constructed iteratively. It could be recomputed completely at every iteration according to Eq. 2, but the iterative construction is more efficient and simple to implement.

Note also, that a classifier which does not perform better than chance (i.e., for which $e_m = 1/2$) receives a weight of zero. A perfect classifier ($e_m = 0$) would receive an infinite weight, since it would be the only committee member that we need. A perfect liar ($e_m = 1$) would receive a negative infinite weight. We just reverse its decision every time and draft it as the only committee member.

6 Problems

1. Show that if we assign cost a to misses and cost b to hits, where $a > b > 0$, we can rewrite such costs as $a = c^d$ and $b = c^{-d}$ for constants c and d. That is, exponential loss costs of the type e^{α_m} and $e^{-\alpha_m}$ do not compromise generality.

2. Show that normalizing the weights of the data points at each iteration is irrelevant

3. If we leave the weights of hits unchanged, show how to correctly update the weights of misclassified data points.

4. Define a two-class problem in the plane (for example, a circle of points inside a square) and build a strong classifier out of a pool of randomly generated linear discriminants of the type $\text{sign}(ax_1 + bx_2 + c)$.

5. Read the code of the implementation of the Viola-Jones algorithm in the OpenCV library.

References

[1] Y. Freund, and R. Shapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Proceedings of the Second European Conference on Computational Learning Theory*, 1995, pp. 23–37.

[2] Paul A. Viola, Michael J. Jones, "Robust Real-Time Face Detection", ICCV 2001, Vol. 2, pp. 747.

[3] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer-Verlag, New York, 2001.