Boosting algorithms for Supervised Learning

A practical study of *AdaBoost*

Alexandre Combessie   Ismail Machraoui

ENSAE ParisTech

Project for the "Aggregation in Statistical Learning" course by Pierre Alquier, 2016

Introduction

- Freund and Schapire (1996) start with a generalized algorithm for the on-line allocation model
  - Example: allocating bets among horse-racing "experts"
  - Adaptation of Littlestone and Warmuth (1994) multiplicative weight-update rule for majority voting
- From this on-line setting, they derive a Boosting algorithm for supervised, batch learning
  - Idea: convert a family of "Weak Learner" algorithms into a single "Strong Learner"
  - Non-obvious reversal of the online-to-batch framework
- Starting with a simple algorithm for binary classification, they extend it to multi-class classification and regression

Outline

1. Algorithm description
   - On-line allocation origin
   - Batch binary classification
   - Multi-class extensions
2. Mathematical results
   - Training
   - Generalization
3. Empirical study
   - Chosen implementations
   - Performance analysis

General on-line allocation framework

- Starting example: How to allocate money among gamblers on horse races? (a bit like bandits, but with agents)
- Mathematical framework:
  - Agent A with $\{1, ..., N\}$ *strategies* to choose from
  - At each *trial* $t = \{1, ..., T\}$, agent A decides on the distribution $\mathbf{p}^t$ over the strategies
  - Each strategy i incurs loss l_i^t (possibly in adversarial environment) so that the loss of A is $\mathbf{p}^t \cdot \mathbf{l}^t$
  - The goal of agent A is to minimize its cumulative loss compared to the loss of the best strategy:

$$\min_{\mathbf{p}^t} \left\{ \sum_{t=1}^{T} \mathbf{p}^t \cdot \mathbf{l}^t - \min_i \sum_{t=1}^{T} l_i^t \right\}$$

Proposed on-line allocation algorithm

Hedge(β)

Initialization:

Weight multiple parameter $\beta \in [0, 1]$

Number of trials $T \in \mathbb{N}^*$

Initial weight vector $\mathbf{w}^1 \in [0, 1]^N$ with $\sum_{i=1}^{N} w_i^1 = 1$

for $t = 1$ to T number of trial **do**

  Set strategy allocation $\mathbf{p}^t \leftarrow \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$

  Observe realization with loss vector $\mathbf{l}^t$

  Incur loss of $\mathbf{p}^t \cdot \mathbf{l}^t$

  for $i = 1$ to N **do**

    Update weight vector $w_i^{t+1} \leftarrow w_i^t \times \beta^{l_i^t}$

  end for

end for

Preliminary framework to define the concepts of Boosting, Weak Learners and Strong Learners

- Boosting is the transformation of a "Weak Learner" algorithm into a "Strong Learner" one
- The notion of "Weak Learner" is defined in the Probably Approximately Correct (PAC) learning framework
  - A "Strong Learner" is an algorithm that given $\varepsilon, \delta > 0$ outputs a hypothesis with error $< \varepsilon$ with probability $1 - \delta$
  - A "Weak Learner" only verifies it for $\varepsilon \geq 1/2 - \gamma$ where $\gamma > 0$ or decreases as $1/p$ with p polynomial
  - For simplification, the PAC learning framework is not used in the rest of the paper, in favor of a more general one where examples (x_i, y_i) are chosen randomly according to a fixed but unknown distribution $\mathfrak{P}$ over $X \times Y$
- In the context of *batch* learning we will focus on boosting by *sampling* over the examples

How to adapt this on-line allocation algorithm to boosting problems in batch settings? (1/2)

- The original boosting algorithm was developed by Schapire, and improved by Freund as the "boost-by-majority" algorithm
- The issue of these algorithms is that they require to know the bias of the Weak Learner in advance, and does not make us of all the Weak Learner hypothesis.
- **To solve these problems, Freund and Schapire decide to adapt their online allocation algorithm in the context of boosting**

How to adapt this on-line allocation algorithm to boosting problems in batch settings? (2/2)

- There are correspondences between the on-line allocation model and the problem of boosting
- The authors actually choose the less obvious reverse correspondence:

	Boosting problem	
On-line allocation	*Natural*	*Reversed*
Strategy	Weak Learner	Training example
Trial	Training example	Weak Learner

Input: Labeled examples $(x_1, y_1)...(x_N, y_N)$ with distribution D, algorithm **WeakLearn**, number of iterations T

Initialization: Weight vector $w_i^1 \leftarrow D(i)$ **for** $i = 1$ to N

for $t = 1$ to T number of iterations **do**

Set example distribution $\mathbf{p}^t \leftarrow \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$

Call **WeakLearn** with distribution $\mathbf{p}^t$, get hypothesis h_t

Compute h_t loss $\varepsilon_t \leftarrow \sum_{i=1}^N p_i^t [\![h_t(x_i) \neq y_i]\!]$

Set $\beta_t \leftarrow \varepsilon_t/(1 - \varepsilon_t)$

Update weight $w_i^{t+1} \leftarrow w_i^t \times \beta^{1-[\![h_t(x_i) \neq y_i]\!]}$ **for** $i = 1$ to N

end for

return Final hypothesis

$$h_f(x) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t} \right) [\![h_t(x) \neq y]\!]$$

Multi-class extensions $Y = \{1, ..., k\}$

There are several ways to extend the previous algorithm in multi-class contexts:

1. **AdaBoost.M1**: Naive extension of binary AdaBoost by replacing $Y = \{0, 1\}$ by $Y = \{1, ..., k\}$ and adding a rule to abort the main loop if $\varepsilon_t > 1/2$
2. **Adaboost.M2**: Advanced extension of AdaBoost with more communication between the boosting and Weak Learner algorithm: probabilities and class weights
3. **Binarization**: For k classes, perform boosting separately on the k binarized problems and aggregate them according to rules like One-vs-Rest or Error-Correcting Output

AdaBoost.M1

- The algorithm is very close to the binary AdaBoost, but it adds an important **loop breaking rule**:

$$\varepsilon_t = \sum_{i=1}^{N} p_i^t [\![h_t(x_i) \neq y_i]\!] > 1/2$$

- Without this rule, we could have a weight multiplier $\beta_t > 1$ and the algorithm would diverge
- Inherent problem with this approach: The more k classes, the harder it is for a Weak Learner to ensure $\varepsilon_t \leq 1/2$

AdaBoost.M2

Modifications to AdaBoost.M1 are highlighted in red

Input: Labeled examples $(x_1, y_1)...(x_N, y_N)$ with distribution D, algorithm **WeakLearn**, number of iterations T

Initialization: $w^1_{i,y} \leftarrow D(i)/(k-1)$ **for** $i = 1$ to N and $y \in Y - \{y_i\}$

for $t = 1$ to T number of iterations **do**

Set example distribution $D_t(i) \leftarrow \frac{\sum_{y \neq y_i} w^t_{i,y}}{\sum_{i=1}^N \sum_{y \neq y_i} w^t_{i,y}}$

Set label weights $q_t(i, y) \leftarrow \frac{w^t_{i,y}}{\sum_{y \neq y_i} w^t_{i,y}}$

Call **WeakLearn** with example distribution $\mathbf{D}_t$ and label weights $\mathbf{q}_t$, get hypothesis h_t with probability values in $[0, 1]$

Compute h_t pseudo-loss

$\varepsilon_t \leftarrow \frac{1}{2} \sum_{i=1}^N D_t(i)\Big(1 - h_t(x_i, y_i) + \sum_{y \neq y_i} q_t(i, y) h_t(x_i, y)\Big)$

Set $\beta_t \leftarrow \varepsilon_t/(1 - \varepsilon_t)$

$w^{t+1}_{i,y} \leftarrow w^t_{i,y} \times \beta^{\frac{1}{2}(1 + h_t(x_i, y_i) - h_t(x_i, y))}$ **for** $i = 1$ to N and $y \in Y - \{y_i\}$

end for

return Final hypothesis $h_f(x) = \text{argmax}_{y \in Y} \sum_{t=1}^T \left(\log \frac{1}{\beta_t}\right) h_t(x, y)$

Binarization - One vs all

Input: Labeled examples $(x_1, y_1)...(x_N, y_N)$ with $y_i \in \{1, ..., k\}$ and with distribution D, algorithm **AdaBoost**, number of iterations T

for $j = 1$ to k number of classes **do**

Transform $(x_1, y_1)...(x_N, y_N)$ into a binary dataset $(x_1, y_1^{(j)})...(x_N, y_N^{(j)})$ where $y_i^{(j)} = [\![y_i = j]\!]$

Call **AdaBoost** on $(x_1, y_1^{(j)})...(x_N, y_N^{(j)})$ with distribution D and number of iterations T

Get hypothesis $h_f^{(j)}$ with probability values in $[0, 1]$

end for

return Final hypothesis

$$h_f(x) = \underset{j \in \{1,...,k\}}{\operatorname{argmax}} \, h_f^{(j)}(x)$$

Training error : Adaboost.M1

- Let $\varepsilon_1 \dots \varepsilon_t$ be the **Weak Learner** algorithm generated errors : $\varepsilon_t = \sum_{i=1}^{N} p_i^t [\![h_t(x_i) \neq y_i]\!]$
- Let ε the error of h_f (the final hypothesis) :

$$\varepsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\varepsilon_t(1-\varepsilon_t)} \text{(similar to } \textbf{Adaboost}\text{)}$$

Or if $\varepsilon = \frac{1}{2} - \gamma$:

$$\varepsilon \leq \prod_{t=1}^{T} \sqrt{1-4\gamma^2}$$

If all the errors of all the hypotheses are equal, the inequality can be simplified to :

$$\varepsilon \leq exp(-2T\gamma^2)$$

Training error : Adaboost.M2

- Let $\varepsilon_1 \ldots \varepsilon_t$ be the **Weak Learner** algorithm generated errors. As a reminder, ε_t stands here for the p-loss.
- Let ε the error of h_f (the final hypothesis) :

$$\varepsilon \leq 2^T (k-1) \prod_{t=1}^{T} \sqrt{\varepsilon_t (1 - \varepsilon_t)}$$

- It can be explained by a reduction to a binary **Adaboost** and then apply **Adaboost** upper-bound to get to the stated result.

Generalization error

- Provided that hypotheses of WeakLearn are from a class of functions which VC-dimension d ≥ 2 then :
  For any $\delta > 0$

$$\Pr[|\varepsilon_f - \hat{\varepsilon}| > 2\sqrt{\frac{d_f(ln(\frac{2N}{d_f}) + ln(\frac{9}{\delta}))}{N}}] \leq \delta$$

  where $d_f \leq 2(d+1)(T+1)log_2[e(T+1)]$, ε_f stands for the generalization error of the final hypothesis
- The cross-validation technique might be used in order to get an optimal performance (i.e smallest error of the final hypothesis h_f^T).
- Making the parameter T vary, the best T kept is the one minimizing the error of the final hypothesis on the validation set.

Chosen dataset
Heart Disease UCI

- Heart disease UCI dataset gathers data about 303 patients in Hungary, Switzerland and the USA. It is available on archive.ics.uci.edu/ml/datasets/Heart+Disease.
- The dataset contains 14 features either catagorical or numerical, with few missing values.
- The target variable refers to the existence of heart disease for a given patient. It takes values in $\{0,1,2,3,4\}$ where 0 stands for pathology absence and the other remaining values points out some level degree of the disease.

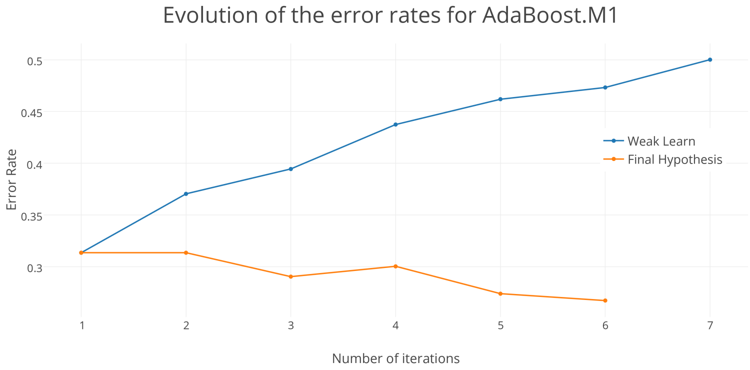
Algorithm implementations

The algorithms we've chosen to implement are :

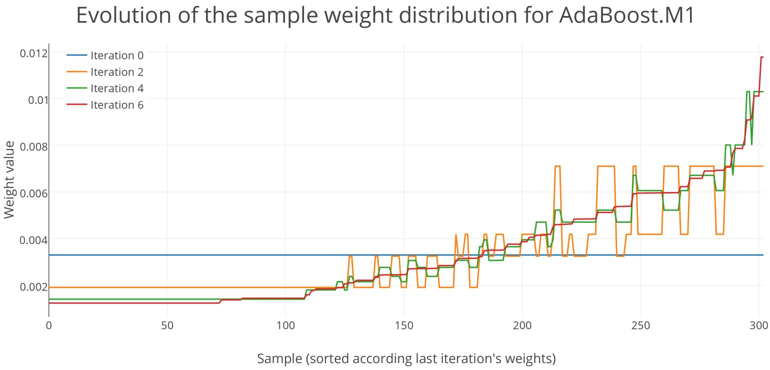
1. Adaboost.M1
2. Adaboost.M2
3. Binarization - one vs all

We set max iterations to $T = 100$

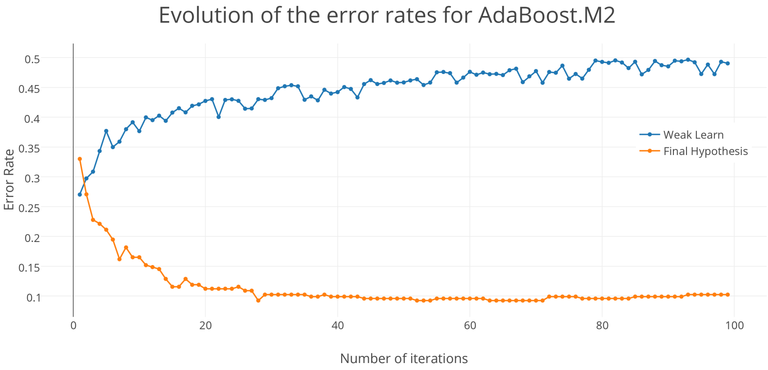
Performance analysis : Adaboost.M1 (1/2)



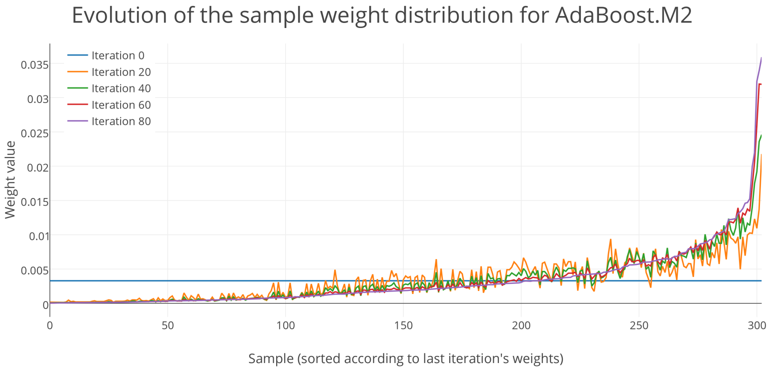
Performance analysis : Adaboost.M1 (2/2)



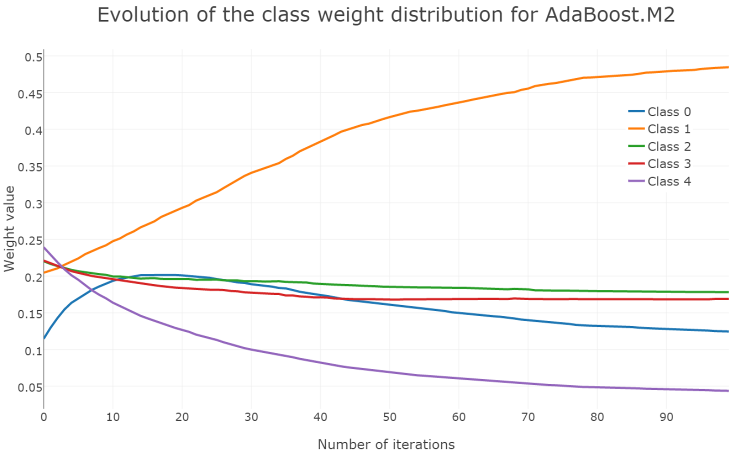
Performance analysis : Adaboost.M2 (1/3)



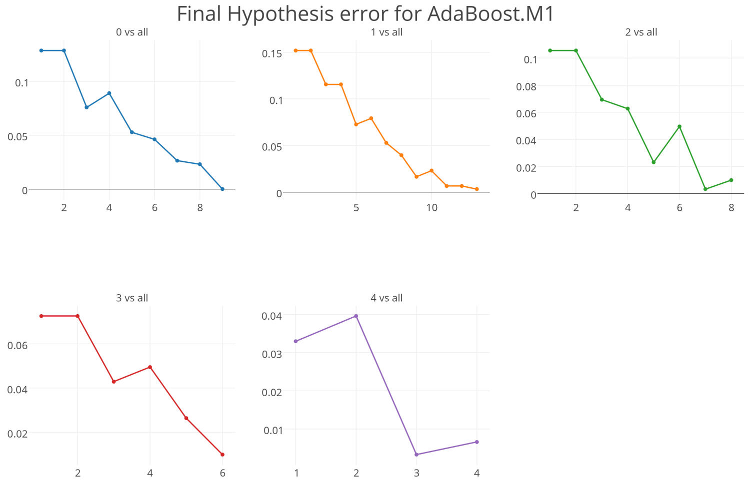
Performance analysis : Adaboost.M2 (2/3)



Performance analysis : Adaboost.M2 (3/3)

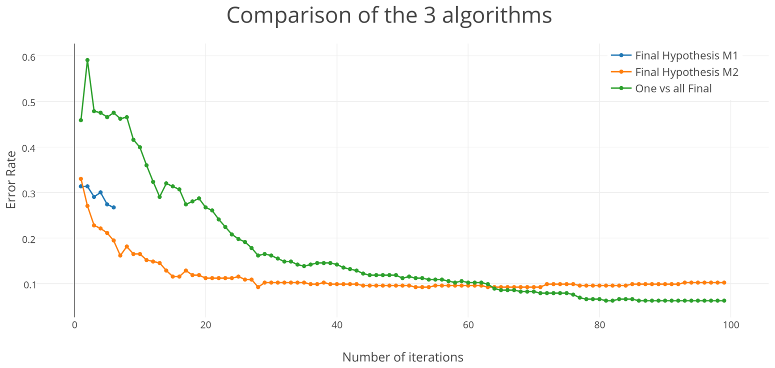


Performance analysis : One vs all



Performance analysis

Summary



For Further Reading

Y. Freund and R. Schapire.
A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting
Journal of Computer and System Sciences, 55:119–139, 1997.

Y. Freund and R. Schapire.
A Short Introduction to Boosting
In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1401–1406, 1999.