

Ensembles - Bagging and Boosting

JIA Pei

Email: jp4work@gmail.com

Vision Open Working Group

First Edition

January 3, 2011

Contents

0.1	Supervised Classification	1
0.2	Additive Models	1
0.2.1	Base Learner	1
0.2.2	Additive Models	2
0.3	Bagging	2
0.4	Boosting	3
0.4.1	Introduction	3
0.4.2	Optimization	4
0.4.3	Margin Based Cost Function for Boosting	5
0.4.4	A Summary of Boosting Variants	6

Abstract

Adaboost is one of the most successful machine learning achievements, which has been widely applied in many research frontiers, particularly, in the area of face detection for computer vision. This technical report thoroughly reviews the theoretical background of boosting based classification methods, and carefully categorizes these methods by adopted cost functions and optimization methods.

0.1 Supervised Classification

A supervised classification problem can be abstracted as: Given the sample space X , and the label space Y , with a “supervised” probability distribution D on $X \rightarrow Y$, how to design a classifier to afford satisfactory classification performance? Namely, we expect the probability that h_f wrongly classifies a random testing sample $x, x \in X$ could be as small as possible. Formularily, we expect

$$P(h_f(x) \neq y | D : X \rightarrow Y) \rightarrow \mathbf{min} \quad (1)$$

, where y is the correct class that x should belong to, while $h_f(x)$ is the classification result of x given by the classifier h_f .

However, the distribution D in (1) is normally unknown. Then, N given samples $(x_1, y_1), (x_2, y_2) \cdots (x_N, y_N), x_i \in X, y_i \in Y, i = 1, 2 \cdots N$ are used instead to train a reasonable classifier. A cost function C is then introduced to evaluate the classifiability of this classifier for each training sample. The generalized form of the classifier’s cost function is a function of the classifier’s output $h_f(x)$ and the expected output y , which can be denoted as $C(y, h_f(x))$.

Clearly, instead of solving (1), our goal now changes to minimize

$$C_f(h_f | x_i \rightarrow y_i, i = 1, 2 \cdots N) = \frac{1}{N} \sum_{i=1}^N C(y_i, h_f(x_i)) \quad (2)$$

, where C_f is the final averaged total cost of all training samples $(x_i, y_i), i = 1, 2 \cdots N$ using classifier h_f . Due to various choices of C , C_f can be looked on as the functional of the classifier h_f .

0.2 Additive Models

0.2.1 Base Learner

In real applications, a number of features are extracted to represent every training sample. For every extracted feature, a very simple base classifier – stump (including naive stump, SVM-stump, etc.) can be used to evaluate the classifiability of this certain feature. A stump function is a thresholding method for binary classification. A SVM-stump function is a specific stump function which thresholds two classes by SVM criterion. In other words, stump functions and their variants can be looked on as just a single level decision tree. Formularily, a stump can be denoted as

$$h : X \rightarrow \{\pm 1\} \quad (3)$$

, where X is the sample space, and $\{\pm 1\}$ is the binary mapping space, h is the stump classifier which defines a simple classification rule. **Note:** These base learners must perform at least slightly better than random classifiers.

0.2.2 Additive Models

Additive models for supervised classification is to build a “stronger classifier” based upon multiple weighted base classifiers. Formularily,

$$h_f(x) = F\left(\sum_{t=1}^T w_t h_t(x)\right) \quad (4)$$

, where T is the total number of base classifiers, $h_t(x)$ is the output of the t th base classifier with respect to sample x (stump defined in (3) is one of the choices as base classifiers), w_t is the additive weights for respective base classifiers while $F()$ is the fusion function to construct the final “stronger classifier” h_f (sign function defined in (5) can be chosen as one of the fusion functions).

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases} \quad (5)$$

Considering the complexity of (4), the base learners h_t are normally selected as simple as possible due to the following two reasons:

1. The number of extracted features is large. If every feature is corresponding to one base classifier, then there will a large number of base classifiers.
2. Some applications such as face detection require real-time performance.

Therefore, naive stumps are preferred to SVM-stumps.

0.3 Bagging

Bagging stands for bootstrap aggregating. It is a kind of ensemble methods that combines multiple predictors. In bagging, one predictor is a classifier built from one bootstrap set sampled from all training samples, with displacement. Bagging algorithms have a uniform procedure:

1. Resample B bootstrap sets from X with displacement to obtain X_1, X_2, \dots, X_B . It's inferable that around 37% training samples will be left out from a single bootstrap set.

2. For each bootstrap set X_i , train a predictor h_i , where $i = 1, 2, \dots, B$. Clearly, if the bootstrap set X_i doesn't contain one sample x , the predictor h_i may not be able to predict a classification for it.
3. For an arbitrary sample x in original set X , supposing there are K bootstrap sets containing the same sample (Clearly, $K \leq B$), make K predictions by corresponding predictors. These K prediction results are denoted as $h'_k(x)$, where $k = 1, 2, \dots, K$.
4. Average these K predictions as the final classification result. This is how bootstrap aggregating (bagging) comes and how additive model is used in bagging.

0.4 Boosting

0.4.1 Introduction

Unlike bagging takes the strategy to bootstrap a single training sample set to multiple ones, boosting confronts the training sample set directly. That is to say, boosting is just working on the original training sample set and its purpose is just to minimize the total cost (2) directly. All boosting methods, including AdaBoost variants like discrete AdaBoost, real AdaBoost, gentle AdaBoost, and CGBoost, LogitBoost, BrownBoost, LPBoost, TotalBoost, MadaBoost etc., are of a unique boosting framework, viewed from functional analysis point.

A monograph about boosting can be retrieved in ICCV 2009's tutorial "Boosting and Random Forest for Visual Recognition" [1]. However, this tutorial only mentioned boosting variants using gradient descent search strategy but overlooked other search strategies, such as conjugate gradient based methods, Newton-Raphson based methods, etc. In the next subsection 0.4.2, we will cover all three typical search strategies mentioned above for boosting,

Let's have a look at our current objective – lowering the total cost of (2). This can be regarded as equivalent to optimizing the "stronger" classifier h_f . Clearly, the optimization process may variate in three aspects:

- How to construct the "stronger" classifier h_f . Additive models as in (4),
- What is the search strategy to optimize h_f . Refer to 0.4.2.
- How to define the cost function C . Refer to 0.4.3.

In fact, by looking on a final “stronger” classifier as a function, all such possible “stronger” classifiers will compose a function space. Ideally, this function space is a Hilbert space [5]. The inner product of two functions h_f^j and h_f^k in this function space is defined as:

$$\langle h_f^j, h_f^k \rangle \equiv h_f^{jT} \cdot h_f^k = \sum_{i=1}^N h_f^j(x_i) h_f^k(x_i) \quad (6)$$

, where h_f^{jT} is the transpose of function h_f^j . More specifically, if we sequentially list all outputs of function h_f^j as a column vector, then its transpose h_f^{jT} is the corresponding row vector.

0.4.2 Optimization

Classical optimization methods include gradient descent method with line search, conjugate gradient (CG) based methods, Newton-Raphson’s method, etc.

Gradient Descent Optimization for Boosting

Supposing we have already estimated a “stronger” classifier $h_f = h_f^0$ with a comparatively low cost $C_f(h_f)$ for the first round, and we wish to optimize it using gradient descent algorithm. That is to say, we are looking for a small enough positive scalar ϵ and a direction h_f^1 , and we wish the final cost $C_f(h_f + \epsilon h_f^1)$ decreases as rapidly as possible. Simply take one order Taylor expansion of the final cost (2)

$$C_f(h_f + \epsilon h_f^1) \approx C_f(h_f) + \epsilon \langle \nabla C_f(h_f), h_f^1 \rangle \quad (7)$$

, where $\nabla C_f(h_f)$ is the first order functional derivative of C_f at function h_f , $\langle \nabla C_f(h_f), h_f^1 \rangle$ is the inner product of two functions. Our objective now is deduced to minimizing $\langle \nabla C_f(h_f), h_f^1 \rangle$. After choosing h_f^1 , the final “stronger” classifier will be updated to $h_{f+} = \epsilon h_f^1$. We may continue the iterative process to select the next update function h_f^2 , and then h_f^3 , etc. This iteration continues until $\langle \nabla C_f(h_f), h_f^i \rangle \geq 0, 1 \leq i \leq T$ or maximum number of iterations arrives.

In [3,4], the above iterative process based on gradient descent method is named as *AnyBoost*, and a brief generalized AnyBoost algorithm is given in algorithm 1.

In fact, the name “AnyBoost” used in [3,4] has been abused and falls short of the reality. Instead of using gradient descent method, boosting can be

realized by other search strategies such as CG method and Newton-Raphson method in the following.

Conjugate Gradient Optimization for Boosting

Compared to 0.4.2, instead of searching in the selected gradient descent direction, conjugate gradient optimization searches in a modified direction, which is a linear combination of the gradient descent (can be steepest descent direction) and the search direction of the former step. CGBoost proposed in [2] views CG based boosting from the functional analysis. A brief generalized conjugate gradient method based boosting algorithm is given in algorithm 2.

Newton-Raphson Optimization for Boosting

Taking two order Taylor expansion of the final cost (2), we may have

$$C_f(h_f + \epsilon h_f^1) \approx C_f(h_f) + \epsilon \langle \nabla C_f(h_f), h_f^1 \rangle + \frac{1}{2} \epsilon^2 h_f^{1T} \nabla^2 C_f(h_f) h_f^1 \quad (8)$$

, where $\nabla^2 C_f(h_f)$ is the second order functional derivative of C_f at function h_f ,

By looking on (8) as a function of h_f^1 , in order to minimize it, we may take one order derivative and obtain:

$$\epsilon h_f^1 = -\frac{\nabla C_f(h_f)}{\nabla^2 C_f(h_f)} \quad (9)$$

A brief generalized Newton-Raphson method based boosting algorithm is given in algorithm 3.

Note: Similar to all optimization problems, it is possible that our final “stronger” classifier traps into a local minimum because no convex set can be guaranteed. So, the selection of the very first “stronger” classifier is vital. Normally, h_f^0 is chosen as an equally weighted sum of all base classifiers, namely:

$$h_f^0(x) = \sum_{t=1}^T \frac{1}{T} h_t(x) \quad (10)$$

0.4.3 Margin Based Cost Function for Boosting

Definition *Margin* of an training sample is defined as the product of the classifier’s output $h_f(x)$ and the expected output y , namely, $yh_f(x)$ [3,4].

Algorithm 1 AnyBoost Algorithm

Require: All base classifiers (normally, weak classifiers) $h_t; 1 \leq t \leq T$;
 N training samples $(x_1, y_1), (x_2, y_2) \cdots (x_N, y_N), x_i \in X, y_i \in Y, i = 1, 2 \cdots N$

- 1: Select the initialized “stronger” classifier h_f as $h_f^0(x)$ defined in (10)
- 2: **for** $i = 1$ to T **do**
- 3: Calculate $\nabla C_f(h_f)$, namely, one order derivative of functional C_f at function h_f .
- 4: Carefully select a function h_f^i so that $\langle \nabla C_f(h_f), h_f^i \rangle$ is minimized. Normally, the steepest descent gradient direction is selected.
- 5: **if** $\langle \nabla C_f(h_f), h_f^i \rangle \geq 0$ **then**
- 6: **return** h_f
- 7: **end if**
- 8: Pick up a suitable positive weight w_i by line search
- 9: Update the “stronger” classifier by $h_{f+} = w_i h_f^i$
- 10: **end for**
- 11: The final classifier is h_f

For binary classification problem, $y = \pm 1$, $sign(yh_f(x))$ just indicates whether the sample x is correctly (if $sign(yh_f(x)) = 1$) or wrongly (if $sign(yh_f(x)) = -1$) classified. By taking off $sign()$ function, $yh_f(x)$ itself shows how accurate or inaccurate the sample x has been classified.

Intuitively, margin $yh_f(x)$ can be used to weight the classifiability of the classifier h_f . Practically, it has been adopted as variables of cost functions by many existing boosting variants, such as AdaBoost including discrete AdaBoost, real AdaBoost and gentle AdaBoost, CGBoost, LogitBoost, etc.

Actually, it might be possible to define margin in some other forms rather than here.

0.4.4 A Summary of Boosting Variants

Categorily speaking, without considering the adopted cost functions, and if denoting boosting algorithms based on steepest descent optimization as “SDBoost”, boosting algorithms based on conjugate gradient optimization as “CGBoost”, boosting algorithms based on Newton-Raphson optimization as “NRBoost”, then, boosting should have a lot of variants as summarized in table 1.

However, terms have been occupied (in fact, abused) by some existing popular boosting algorithms, as enumerated in table 2. All these popular boosting algorithms adopt cost functions defined upon variable margin (refer

Algorithm 2 Conjugate Gradient Method Based Boosting Algorithm

Require: All base classifiers (normally, weak classifiers) $h_t; 1 \leq t \leq T$;
 N training samples $(x_1, y_1), (x_2, y_2) \cdots (x_N, y_N), x_i \in X, y_i \in Y, i = 1, 2 \cdots N$

- 1: Select the initialized “stronger” classifier h_f as $h_f^0(x)$ defined in (10)
- 2: **for** $i = 1$ to T **do**
- 3: Calculate $\nabla C_f(h_f)$, namely, one order derivative of functional C_f at function h_f .
- 4: Carefully select a function h_f^i so that $\langle \nabla C_f(h_f), h_f^i \rangle$ is minimized. Normally, the steepest descent gradient direction is selected.
- 5: $g_f^i = h_f + \alpha h_f^i$, α can be reasonably deducted by Fletcher-Reeves formula, or Polak-Ribière formula, or Hestenes-Stiefel formula (Refer to [6]).
- 6: **if** $\langle \nabla C_f(h_f), g_f^i \rangle \geq 0$ **then**
- 7: **if** $\langle \nabla C_f(h_f), h_f^i \rangle \geq 0$ **then**
- 8: **return** h_f
- 9: **end if**
- 10: $g_f^i = h_f^i$
- 11: **end if**
- 12: Pick up a suitable positive weight w_i by line search
- 13: Update the “stronger” classifier by $h_{f+} = w_i g_f^i$
- 14: **end for**
- 15: The final classifier is h_f

Algorithm 3 Newton-Raphson Method Based Boosting Algorithm

Require: All base classifiers (normally, weak classifiers) $h_t; 1 \leq t \leq T$;
 N training samples $(x_1, y_1), (x_2, y_2) \cdots (x_N, y_N), x_i \in X, y_i \in Y, i = 1, 2 \cdots N$

- 1: Select the initialized “stronger” classifier h_f as $h_f^0(x)$ defined in (10)
- 2: $i = 0$
- 3: **while** $i < MAXITER$ **do**
- 4: Calculate one order derivative $\nabla C_f(h_f)$ and two order derivative $\nabla^2 C_f(h_f)$ of functional C_f at function h_f .
- 5: Calculate ϵh_f^i using (9)
- 6: **if** ϵh_f^i is neglectable **then**
- 7: **return** h_f
- 8: **end if**
- 9: Update the “stronger” classifier by $h_{f+} = \epsilon h_f^i$
- 10: **end while**
- 11: The final classifier is h_f

Table 1: Boosting Categories

Boosting Variants	h_f	Optimization Method	Step
Discrete SDBoost	± 1	Steepest Descent	Line Search
Discrete CGBoost		Conjugate Gradient	Line Search
Discrete NRBoost		Newton-Raphson	N/A
Real SDBoost	real	Steepest Descent	Line Search
Real CGBoost		Conjugate Gradient	Line Search
Real NRBoost		Newton-Raphson	N/A

to 0.4.3).

Table 2: Various Boosting Algorithms Based on Margin

Boosting Variants	h_f	Cost Function	Optimization Method	Step
Discrete AdaBoost	± 1	$e^{-yh_f(x)}$	Steepest Descent	Line Search
Real AdaBoost	real	$e^{-yh_f(x)}$	Steepest Descent	Line Search
Gentle AdaBoost	real	$e^{-yh_f(x)}$	Newton-Raphson	N/A
LogitBoost	± 1 or real	$\ln(1 + e^{-yh_f(x)})$	Newton-Raphson	N/A
CGBoost	± 1 or real	$e^{-yh_f(x)}$	Conjugate Gradient	Line Search

We may use different well-designed cost functions rather than those displayed in table 2. However, these cost functions are popular ones that have been widely used in market. Our experiments are just based on these typical cost functions.

Bibliography

- [1] T. Kim, J. Shotton, and B. Stenger. Boosting and random forest for visual recognition. In *IEEE Conference on Computer Vision (ICCV 2009)*, 2009.
- [2] L. Li, Y. S. Abu-Mostafa, and P. Amrit. Cgboost: Conjugate gradient in function space. Technical report, California Institute of Technology, 2003.
- [3] L. Mason, J. Baxter, P. Bartlett, and M. Frean. *Advances in Large Margin Classifiers*, chapter Functional gradient techniques for combining hypotheses, pages 221–246. MIT Press, 1999.
- [4] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. *Advances in Neural Information Processing Systems*, 12, 2000.
- [5] Wikipedia. Hilbert space — wikipedia, the free encyclopedia, 2009.
- [6] Wikipedia. Nonlinear conjugate gradient method — wikipedia, the free encyclopedia, 2009.