Pattern Recognition 45 (2012) 4428–4437

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

# Functional gradient ascent for Probit regression

Songfeng Zheng [a],*, Weixiang Liu [b]

[a] Department of Mathematics, Missouri State University, 901 S. National Ave., Springfield, MO 65897, USA
[b] Biomedical Engineering Lab, School of Medicine, Shenzhen University, Shenzhen, Guangdong 518060, China

ARTICLE INFO

*Article history:*
Received 22 March 2012
Received in revised form
17 May 2012
Accepted 11 June 2012
Available online 19 June 2012

*Keywords:*
Probit regression
Classification
Functional gradient ascent
Boosting

ABSTRACT

This paper proposes two gradient based methods to fit a Probit regression model by maximizing the sample log-likelihood function. Using the property of the Hessian of the objective function, the first method performs weighted least square regression in each iteration of the Newton–Raphson framework, resulting in ProbitBoost, a boosting-like algorithm. Motivated by the gradient boosting algorithm [10], the second proposed approach maximizes the sample log-likelihood function by updating the fitted function a small step in the gradient direction, performing gradient ascent in functional space, resulting in Gradient ProbitBoost. We also generalize the algorithms to multi-class problems by two strategies, one of which is to use the gradient ascent to maximize the multi-class sample log-likelihood function for fitting all the classifiers simultaneously, and the second approach uses the one-versus-all scheme to reduce the multi-class problem to a series of binary classification problems. The proposed algorithms are tested on typical classification problems including face detection, cancer classification, and handwritten digit recognition. The results show that compared to the alternative methods, the proposed algorithms perform similar or better in terms of testing error rates.

## 1. Introduction

In binary classification problems, we have response variable $Y \in \{0,1\}$, and the $d$-dimensional predictor vector $\mathbf{x} \in \mathbf{R}^d$. Probit regression is appropriate for fitting a binary response model, which assumes the posterior probability of $Y$ given $\mathbf{x}$ has the form

$$P(Y=1|\mathbf{x}) = \Phi(f(\mathbf{x})), \tag{1}$$

where $f(\mathbf{x})$ is a function of the input vector $\mathbf{x}$, and $\Phi(\cdot)$ is the standard normal cumulative distribution function, i.e.,

$$\Phi(t) = \int_{-\infty}^{t} \varphi(z)\,dz \quad \text{with} \quad \varphi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}. \tag{2}$$

It follows from Eq. (1) that

$$P(Y=1|\mathbf{x}) \geq 0.5 \Leftrightarrow f(\mathbf{x}) \geq 0, \tag{3}$$

and this enables us to use sign$(f(\mathbf{x}))$ as the classifier in the decision making stage.

Note that Eq. (1) can be rewritten as

$$P(Y|\mathbf{x}) = \Phi(f(\mathbf{x}))^Y (1-\Phi(f(\mathbf{x})))^{1-Y}, \tag{4}$$

therefore the log-likelihood function is

$$\begin{aligned} l(f) &= Y \log \Phi(f(\mathbf{x})) + (1-Y)\log(1-\Phi(f(\mathbf{x}))) \\ &= Y \log \Phi(f) + (1-Y)\log \Phi(-f), \end{aligned} \tag{5}$$

where we slightly abuse the usage of notation by letting $f = f(\mathbf{x})$, which is clear from the context. Eq. (5) is true because $1-\Phi(f) = \Phi(-f)$ from the symmetry of the standard normal distribution.

To fit the Probit regression model, we find $f(\mathbf{x})$ such that the expected log-likelihood function is maximized:

$$\mathrm{E}[l(f)|\mathbf{x}] = \mathrm{E}[Y \log \Phi(f) + (1-Y)\log \Phi(-f)|\mathbf{x}]. \tag{6}$$

Unfortunately, the expected log-likelihood in Eq. (6) is usually uncomputable because the related distributions are unknown. Instead, given a set of data $\{(\mathbf{x}_i, Y_i), i=1,\ldots,n\}$, we usually maximize the sample log-likelihood function in Eq. (7) to find $f(\mathbf{x})$:

$$\hat{l}(f) = \frac{1}{n}\sum_{i=1}^{n} [Y_i \log \Phi(f(\mathbf{x}_i)) + (1-Y_i)\log \Phi(-f(\mathbf{x}_i))], \tag{7}$$

i.e., the fitted function $f^*$ can be obtained by

$$f^* = \arg\max_f \hat{l}(f). \tag{8}$$

In modern era, high dimensional problems are ubiquitous, for example, the problems in computer vision and bioinformatics usually involve thousands of predictors. As such, a model fitting method being able to solve high dimensional problems is highly appreciated. Maximizing the sample log-likelihood function in Eq. (7) can be solved via an iteratively re-weighted least-squares (IRWLS) approach [14], but the convergence of IRWLS is not guaranteed when there are a large number of predictors. Böhning [3] found a lower bound for the log-likelihood function in Eq. (7)

* Corresponding author. Tel.: +1 471 836 6037.
*E-mail addresses:* SongfengZheng@MissouriState.edu (S. Zheng), wxliu@szu.edu.cn (W. Liu).

0031-3203/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2012.06.006

and maximized this lower bound to obtain an approximation of the Probit regression model. However, in high dimensional spaces, this lower bound method needs to calculate the inverse of a large matrix, which is computationally expensive and unstable. The Probit regression model could also be solved by applying the parameter expansion technique to expectation maximization [19] and data augmentation [20]. The methods presented in [19,20] are theoretically sound, but there is no high dimensional example given. In [22,31], Probit regression was employed together with Bayesian inference for cancer classification, which is a difficult problem in very high dimensional space, and [7] integrated Probit regression model and Bayesian inference in reproducing kernel Hilbert space to perform the same task. However, Bayesian inference usually involves Markov chain Monte Carlo algorithms, which are very time consuming, even in the decision making stage, and this is often undesired in applications.

In the additive model framework, this paper introduces two alternative algorithms to maximize the sample log-likelihood function of Probit regression model, which can work in high dimensional spaces. Using the property of the second order derivative of the expected log-likelihood function, the first method performs weighted least square regression in each iteration of the Newton–Raphson framework, resulting in a boosting-like algorithm, which is called ProbitBoost. Motivated by the gradient boosting algorithm [10], the second proposed approach iteratively maximizes the log-likelihood function by updating the fitted model a small step in the gradient direction, performing gradient ascent in functional space, yielding Gradient ProbitBoost. We also generalize the proposed algorithms to multi-class case by using the first order based gradient ascent to maximize the multi-class sample log-likelihood function for fitting the multiple classifiers simultaneously. As the second option for multi-class problem, we use the one-versus-all approach to reduce the problem to multiple binary classification problems.

The proposed ProbitBoost and Gradient ProbitBoost are tested on typical classification problems with very high dimensional features, including face detection, cancer classification based on gene expression data, and handwritten digit recognition. By analyzing detailed performance curves/measures, we observe that the Newton–Raphson based ProbitBoost performs better than the first order based Gradient ProbitBoost, in terms of both convergence speed and classification error rate. The performance of the proposed algorithm with different weak learners is also investigated, and we recommend the regression stump [25] for a general classification problem. We also experimentally investigate the strategies for setting the step size parameter for the Gradient ProbitBoost. On the considered experiments, compared to the alternative methods, the proposed algorithms achieve better or similar performance in terms of classification error rates. To the best of our knowledge, this is the first attempt to apply gradient based optimization method for fitting the Probit regression model in high-dimensional spaces, and the resulting algorithms are easy to implement with performance comparable to state-of-the-art.

The rest of this paper is organized as follows: Section 2 reviews the interpretation of boosting algorithm as functional gradient descent; Section 3 derives the proposed ProbitBoost and Gradient ProbitBoost algorithms for binary classification problems and generalizes the proposed algorithms to multi-class scenario; Section 4 presents the experimental results on the problems of face detection, cancer classification, and handwritten digit recognition, and the comparison to the alternatives is also presented; in Section 5, we briefly discuss the differences among AdaBoost, LogitBoost, and the proposed ProbitBoost; finally, Section 6 summarizes this paper and discusses the future research direction.

## 2. Boosting as functional gradient descent

Boosting [9] is a classic algorithm in pattern classification and is well known for its simplicity and high accuracy. The powerful feature selection mechanism of boosting makes it suitable to work in high dimensional spaces. Friedman et al. [10,11] developed a general statistical framework which yields a direct interpretation of boosting as a method for function estimation, which is a “stagewise, additive model”.

Consider the problem of function estimation

$$f^*(\mathbf{x}) = \arg\min_f \mathrm{E}[\rho(Y,f(\mathbf{x}))|\mathbf{x}], \tag{9}$$

where $\rho(\cdot,\cdot)$ is a loss function which is typically differentiable and convex with respect to the second argument. Estimating $f^*(\cdot)$ from the given data $\{(\mathbf{x}_i,Y_i),\ i=1,\dots,n\}$ can be performed by minimizing the empirical risk $n^{-1}\sum_{i=1}^{n}\rho(Y_i,f(\mathbf{x}_i))$ and pursuing iterative steepest descent in functional space. This leads us to the generic functional gradient descent algorithm [10], as shown in Algorithm 1.

**Algorithm 1.** Generic functional gradient descent.

```
0. Initialize $f^{[0]}(\cdot)$ with $f^{[0]}(\cdot) = \arg\min_c \frac{1}{n}\sum_{i=1}^{n}\rho(Y_i,c)$,
   or set $f^{[0]}(\cdot)=0$, and set iteration number $m=0$.
1. Increase $m$ by 1. Compute the negative gradient $-(\partial/\partial f)\rho(Y,f)$
   and evaluate at $f^{[m-1]}(\mathbf{x}_i)$:
   $U_i = -\frac{\partial\rho(Y_i,f)}{\partial f}\Big|_{f=f^{[m-1]}(\mathbf{x}_i)}, \quad i=1,\dots,n.$
2. Fit the negative gradients $U_1,\dots,U_n$ to $\mathbf{x}_1,\dots,\mathbf{x}_n$ by the base
   procedure (e.g., the weak learner in AdaBoost):
   $\{(\mathbf{x}_i,U_i),\ i=1,\dots,n\} \longrightarrow g^{[m]}(\cdot)$.
3. Update the estimation by $f^{[m]}(\cdot) = f^{[m-1]}(\cdot) + v g^{[m]}(\cdot)$, where $v$
   is a step size factor.
4. Check the stopping criterion, if not satisfied, go to step 1.
```

Many boosting algorithms can be understood as functional gradient descent with appropriate loss function. For example, if we choose $\rho(Y,f) = \exp(-(2Y-1)f)$, we would recover the AdaBoost algorithm [11]; if we choose $\rho(Y,f) = (Y-f)^2/2$, we would result in $L_2$ Boost [5]; this idea was also applied to quantile regression and classification models, see [16,32,33].

## 3. Functional gradient ascent for Probit regression

This section derives the ProbitBoost algorithm in the Newton–Raphson framework by using the properties of the gradient and Hessian of the log-likelihood function, then the first order gradient method is applied to yield the Gradient ProbitBoost. We also generalize the proposed algorithms to multi-class problems by the first order method and the one-versus-all strategy. For completeness, the details of the weak learners is presented which will be used extensively in this paper.

### 3.1. Gradient and Hessian of the expected log-likelihood function

Gradient based optimization methods are well known for their simplicity in implementation and being able to get at least a local optimum [26]. To fit Probit regression model, we will maximize the sample log-likelihood function in Eq. (7) by gradient ascent. Before we propose the algorithm, let us first investigate the ideal case, i.e., maximizing the expected log-likelihood function $\mathrm{E}[l(f)|\mathbf{x}]$ in Eq. (6) by gradient ascent. In the Newton–Raphson framework, we proceed in an iterative fashion with

$$f^{[m+1]}(\mathbf{x}) = f^{[m]}(\mathbf{x}) - H^{-1}(f^{[m]}(\mathbf{x}))D(f^{[m]}(\mathbf{x})), \tag{10}$$

where $f^{[m]}(\cdot)$ is the fitted function at the $m$-th iteration; $D(\cdot)$ and $H(\cdot)$ are the gradient and Hessian of the objective function, i.e., $\mathrm{E}[l(f)|\mathbf{x}]$, respectively.

The gradient is calculated as:

$$D(f) = \frac{\partial \mathrm{E}[l(f)|\mathbf{x}]}{\partial f} = \mathrm{E}\left[\frac{Y\varphi(f)}{\Phi(f)} - \frac{(1-Y)\varphi(-f)}{\Phi(-f)}\middle|\mathbf{x}\right]$$
$$= \mathrm{E}\left[\frac{\varphi(f)(Y-\Phi(f))}{\Phi(f)\Phi(-f)}\middle|\mathbf{x}\right]. \tag{11}$$

Hessian is defined as

$$H(f) = \frac{\partial D(f)}{\partial f} = \mathrm{E}\left\{\frac{\partial}{\partial f}\left[\frac{\varphi(f)(Y-\Phi(f))}{\Phi(f)\Phi(-f)}\right]\middle|\mathbf{x}\right\} = \mathrm{E}[h(f)|\mathbf{x}], \tag{12}$$

where

$$h(f) = \frac{\partial}{\partial f}\left[\frac{\varphi(f)(Y-\Phi(f))}{\Phi(f)\Phi(-f)}\right]. \tag{13}$$

In the following derivations, we will use the relations, $\Phi(-f) = 1-\Phi(f)$, $\varphi(f) = \varphi(-f)$, and $\varphi'(f) = -f\varphi(f)$, which follow directly from the properties of standard normal distribution.

When $Y=0$, Eq. (13) becomes

$$h(f) = \frac{\partial}{\partial f}\left[\frac{\varphi(f)(0-\Phi(f))}{\Phi(f)\Phi(-f)}\right] = -\frac{\partial}{\partial f}\left[\frac{\varphi(f)}{\Phi(-f)}\right]$$
$$= -\frac{-f\varphi(f)\Phi(-f)+\varphi(f)\varphi(-f)}{\Phi^2(-f)}$$
$$= \frac{\varphi(f)[f\Phi(-f)-\varphi(f)]}{\Phi^2(-f)} = \frac{\varphi(f)G_0(f)}{\Phi^2(-f)}, \tag{14}$$

with

$$G_0(f) = f\Phi(-f)-\varphi(f).$$

When $Y=1$, Eq. (13) becomes

$$h(f) = \frac{\partial}{\partial f}\left[\frac{\varphi(f)(1-\Phi(f))}{\Phi(f)\Phi(-f)}\right] = \frac{\partial}{\partial f}\left[\frac{\varphi(f)\Phi(-f)}{\Phi(f)\Phi(-f)}\right]$$
$$= \frac{\partial}{\partial f}\left[\frac{\varphi(f)}{\Phi(f)}\right] = \frac{-f\varphi(f)\Phi(f)-\varphi(f)\varphi(f)}{\Phi^2(f)}$$
$$= \frac{\varphi(f)[-f\Phi(f)-\varphi(f)]}{\Phi^2(f)} = \frac{\varphi(f)G_1(f)}{\Phi^2(f)}, \tag{15}$$

with

$$G_1(f) = -f\Phi(f)-\varphi(f).$$

In summary, we have $\mathrm{E}[h(f)|\mathbf{x}]$ as the Hessian of the expected log-likelihood function $\mathrm{E}[l(f)|\mathbf{x}]$, with

$$h(f) = \begin{cases} \dfrac{\varphi(f)G_0(f)}{\Phi^2(-f)} & \text{if } Y=0 \\ \dfrac{\varphi(f)G_1(f)}{\Phi^2(f)} & \text{if } Y=1 \end{cases} = \frac{\varphi(f)G_Y(f)}{\Phi^2((2Y-1)f)}. \tag{16}$$

where

$$G_Y(f) = -(2Y-1)f\Phi[(2Y-1)f]-\varphi(f).$$

We can further prove the following property of Hessian:

**Lemma 1.** *$h(f)<0$ for any value of $f(\mathbf{x})$, thus the Hessian of $\mathrm{E}[l(f)|\mathbf{x}]$ is negative.*

**Proof.** It is easy to see that when $f<0$,

$$G_0(f) = f\Phi(-f)-\varphi(f) < f\Phi(-f) < 0.$$

If $f \geq 0$, we have

$$G_0'(f) = \Phi(-f)-f\varphi(-f)-\varphi'(f)$$
$$= \Phi(-f)-f\varphi(f)+f\varphi(f) = \Phi(-f) > 0,$$

which implies that $G_0(f)$ is a monotonically increasing function on $[0,\infty)$. We can calculate $G_0(0) = -\varphi(0) < 0$, and it is easy to verify that

$$\lim_{f\to\infty} G_0(f) = 0.$$

By the property of monotonic increasing function, it follows that $G_0(f)<0$ when $f \geq 0$. Thus, $G_0(f)$ always assumes negative values.

From the definition of $G_0(f)$ and $G_1(f)$, they are symmetric about 0, because

$$G_0(-f) = -f\Phi(f)-\varphi(-f) = -f\Phi(f)-\varphi(f) = G_1(f).$$

Since $G_0(f)$ is negative, $G_1(f)$ is negative as well.

From Eq. (16), it immediately follows that $h(f)<0$. □

According to Lemma 1, the Hessian of $\mathrm{E}[l(f)|\mathbf{x}]$ is negative, which means that the function $\mathrm{E}[l(f)|\mathbf{x}]$ is concave in $f$, thus the maximum point could be found by gradient ascent algorithms.

### 3.2. ProbitBoost: fitting Probit model by Newton–Raphson

From Eq. (10), the Newton–Raphson step for maximizing the expected log-likelihood function is

$$f^{[m+1]}(\mathbf{x}) = f^{[m]}(\mathbf{x}) - H^{-1}(f^{[m]}(\mathbf{x}))D(f^{[m]}(\mathbf{x}))$$
$$= f^{[m]}(\mathbf{x}) - \frac{1}{\mathrm{E}[h(f^{[m]})|\mathbf{x}]}\mathrm{E}\left[\frac{\varphi(f^{[m]})(Y-\Phi(f^{[m]}))}{\Phi(f^{[m]})\Phi(-f^{[m]})}\middle|\mathbf{x}\right]$$
$$= f^{[m]}(\mathbf{x}) + \frac{1}{\mathrm{E}[-h(f^{[m]})|\mathbf{x}]}\mathrm{E}\left[\frac{\varphi(f^{[m]})(Y-\Phi(f^{[m]}))}{\Phi(f^{[m]})\Phi(-f^{[m]})}\middle|\mathbf{x}\right]$$
$$= f^{[m]}(\mathbf{x}) + \mathrm{E}_{-h}\left[\frac{\varphi(f^{[m]})(Y-\Phi(f^{[m]}))}{-h(f^{[m]})\Phi(f^{[m]})\Phi(-f^{[m]})}\middle|\mathbf{x}\right]$$
$$= f^{[m]}(\mathbf{x}) + \mathrm{E}_{-h}\left[\frac{(\Phi(f^{[m]})-Y)\Phi^2((2Y-1)f^{[m]})}{G_Y(f^{[m]})\Phi(f^{[m]})\Phi(-f^{[m]})}\middle|\mathbf{x}\right], \tag{17}$$

where we used the notation $\mathrm{E}_w(\cdot|\mathbf{x})$ for the weighted conditional expectation [11] with

$$\mathrm{E}_w[g(\mathbf{x},y)|\mathbf{x}] = \frac{\mathrm{E}[w(\mathbf{x},y)g(\mathbf{x},y)|\mathbf{x}]}{\mathrm{E}[w(\mathbf{x},y)|\mathbf{x}]} \quad \text{with} \quad w(\mathbf{x},y)>0\ \forall\mathbf{x},\forall y. \tag{18}$$

Eq. (17) uses the weight $-h(f)$, which is reasonable for being a weight function since $h(f)<0$ by Lemma 1. We define a transformed response variable as

$$Y^* = \frac{(\Phi(f)-Y)\Phi^2((2Y-1)f)}{G_Y(f)\Phi(f)\Phi(-f)}, \tag{19}$$

then Eq. (17) shows that in each step, for updating the fitted function, we just need to fit a weighted least square regression of the transformed response $Y^*$ on $\mathbf{x}$ with weight $-h(f)$.

The above procedure can be translated from population version to sample version, resulting in Algorithm 2, which is called ProbitBoost because the procedure is like that of AdaBoost [9], $L_2$ Boost [5], and LogitBoost [11].

**Algorithm 2.** ProbitBoost algorithm.

```
0. Given the total iteration number $M$, training data
   $\{(\mathbf{x}_i,Y_i), i=1,\ldots,n\}$ with $\mathbf{x}_i \in \mathbf{R}^d$ and $Y_i \in \{0,1\}$; initialize
   $f^{[0]}(\mathbf{x})=0$.
1. for $m=1$ to $M$ do:
2.     Calculate the transformed response as
         $Y_i^* = \frac{(\Phi(f_i^{[m-1]})-Y_i)\Phi^2[(2Y_i-1)f_i^{[m-1]}]}{G_{Y_i}(f_i^{[m-1]})\Phi(f_i^{[m-1]})\Phi(-f_i^{[m-1]})}$,
       for $i=1,2,\ldots,n$, where $f_i^{[m-1]}=f^{[m-1]}(\mathbf{x}_i)$.
```

```
3. Calculate the weight by $W_i = -\frac{\varphi(f^{[m-1]}(\mathbf{x}_i))G_{Y_i}(f^{[m-1]}(\mathbf{x}_i))}{\Phi^2[(2Y_i-1)f^{[m-1]}(\mathbf{x}_i)]}$,
   for $i=1,2,\ldots,n$.
4. Fit a function $g_m(\mathbf{x})$ by weighted least square regression
   of $Y_i^*$ on $\mathbf{x}_i$ with weights $W_i$, $i=1,2,\ldots,n$.
5. Update the function $f^{[m]}(\mathbf{x}) = f^{[m-1]}(\mathbf{x}) + g_m(\mathbf{x})$.
6. end for
7. Output the classifier $\text{sign}(f^{[M]}(\mathbf{x}))$.
```

For the weight function, we can further prove:

**Lemma 2.** *The weight function*

$$W(f) = -h(f) = -\frac{\varphi(f)G_Y(f)}{\Phi^2((2Y-1)f)}$$

*has the following property*:

$$\text{if} \quad Y=0, \quad \lim_{f\to-\infty} W(f) = 0 \quad \text{and} \quad \lim_{f\to\infty} W(f) = 1,$$

$$\text{if} \quad Y=1, \quad \lim_{f\to-\infty} W(f) = 1 \quad \text{and} \quad \lim_{f\to\infty} W(f) = 0.$$

**Proof.** We only prove the first half of the lemma because the second half is similar.

When $Y=0$,

$$W(f) = -\frac{\varphi(f)G_0(f)}{\Phi^2(-f)} = \frac{\varphi^2(f) - f\varphi(f)\Phi(-f)}{\Phi^2(-f)}.$$

It is easy to see that when $f\to-\infty$, $\Phi(-f)\to 1$, $\varphi^2(f)\to 0$, and $f\varphi(f)\to 0$. Thus, we conclude $W(f)\to 0$ as $f\to-\infty$.

By repeatedly using *L'Hôpital's rule*, we have

$$\begin{aligned}
\lim_{f\to\infty} W(f) &= \lim_{f\to\infty} \frac{\frac{d}{df}[\varphi^2(f) - f\varphi(f)\Phi(-f)]}{\frac{d}{df}\Phi^2(-f)} \\
&= \lim_{f\to\infty} \frac{-2\varphi^2(f)f - \varphi(f)\Phi(-f) + f\varphi^2(f) + f^2\varphi(f)\Phi(-f)}{-2\Phi(-f)\varphi(f)} \\
&= \frac{1}{2} + \lim_{f\to\infty} \frac{f\varphi(f) - f^2\Phi(-f)}{2\Phi(-f)} \\
&= \frac{1}{2} + \lim_{f\to\infty} \frac{\varphi(f) - f^2\varphi(f) - 2f\Phi(-f) + f^2\varphi(f)}{-2\varphi(f)} \\
&= \lim_{f\to\infty} \frac{f\Phi(-f)}{\varphi(f)} = \lim_{f\to\infty} \frac{\Phi(-f) - f\varphi(f)}{-f\varphi(f)} = 1 - \lim_{f\to\infty} \frac{\Phi(-f)}{f\varphi(f)} \\
&= 1 + \lim_{f\to\infty} \frac{\varphi(f)}{\varphi(f) - f^2\varphi(f)} = 1 + \lim_{f\to\infty} \frac{1}{1-f^2} = 1. \qquad \square
\end{aligned}$$

Fig. 1 shows the plots of the weight functions of positive and negative examples, as fitted values. The trend stated in Lemma 2 is clearly seen. Furthermore, it is evident from Fig. 1 that $W(f)$ is monotonically increasing for negative examples, and monotonically decreasing for positive examples. Thus, for any reasonable value of $f(\mathbf{x})$, the weight function falls in [0,1]. From Fig. 1, we can also observe that, when the algorithm makes a mistake at the current stage (i.e., $f<0$ for positive example or $f>0$ for negative example), the weight for this example will be

$$w > W(0) = \frac{2}{\pi} > 0.5,$$

that is, the algorithm will focus more on this mistakenly classified example in the next iteration. If the algorithm makes further mistake, i.e., for positive example, $f$ decreases; for negative example, $f$ increases, then the weight of this example is increased, forcing the algorithm to pay more attention to this particular example in the next iteration. These properties are desired for the algorithm to have good performance.

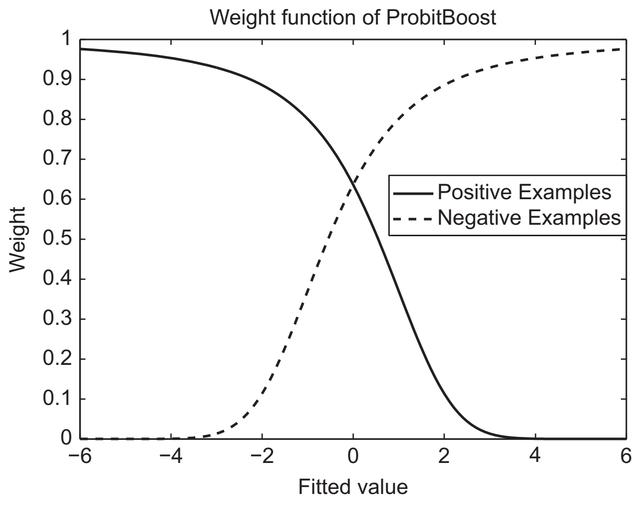

**Fig. 1.** The weights as a function of the fitted value, for positive examples (solid line) and negative examples (dashed line).

### 3.3. *Gradient ProbitBoost: first order functional gradient ascent for Probit regression*

As an alternative to the Newton–Raphson method, the simple gradient ascent method, for its simplicity, could also be used to maximize the expected log-likelihood function in Eq. (6), that is

$$\begin{aligned}
f^{[m+1]}(\mathbf{x}) &= f^{[m]}(\mathbf{x}) + \nu_m D(f^{[m]}(\mathbf{x})) \\
&= f^{[m]}(\mathbf{x}) + \nu_m \mathrm{E}\left[\left.\frac{\varphi(f^{[m]}(\mathbf{x}))(Y-\Phi(f^{[m]}(\mathbf{x})))}{\Phi(f^{[m]}(\mathbf{x}))\Phi(-f^{[m]}(\mathbf{x}))}\right|\mathbf{x}\right],
\end{aligned} \tag{20}$$

where $\nu_m$ is the step size parameter at iteration $m$. Eq. (20) indicates that in each iteration, we only need to fit a least square regression with new response

$$Y^* = \frac{\varphi(f^{[m]}(\mathbf{x}))(Y-\Phi(f^{[m]}(\mathbf{x})))}{\Phi(f^{[m]}(\mathbf{x}))\Phi(-f^{[m]}(\mathbf{x}))}.$$

Translating the above procedure into sample version, we could obtain the algorithm as listed in Algorithm 3.

**Algorithm 3.** Gradient ProbitBoost.

```
0. Given the total number of iterations $M$, training data $\{(\mathbf{x}_i, Y_i),$
   $i=1,\ldots,n\}$ with $\mathbf{x}_i \in \mathbf{R}^d$ and $Y_i \in \{0,1\}$; initialize $f^{[0]}(\mathbf{x}) = 0$.
1. for $m=1$ to $M$ do:
2.   Compute the gradient $(\partial/\partial f)l(Y_i,f)$ and evaluate at $f^{[m-1]}(\mathbf{x}_i)$:
     $Y_i^* = \frac{\partial}{\partial f} l(Y_i,f)\Big|_{f=f^{[m-1]}(\mathbf{x}_i)} = \frac{\varphi(f)(Y_i-\Phi(f))}{\Phi(f)\Phi(-f)}\Big|_{f=f^{[m-1]}(\mathbf{x}_i)}, i=1,\ldots,n.$
3.   Fit the gradient vector $Y_1^*,\ldots,Y_n^*$ to $\mathbf{x}_1,\ldots,\mathbf{x}_n$ by the weak
     learner:
     $\{(\mathbf{x}_i, Y_i^*), i=1,\ldots,n\} \longrightarrow g^{[m]}(\cdot)$.
4.   Update the estimation by $f^{[m]}(\cdot) = f^{[m-1]}(\cdot) + \nu_m g^{[m]}(\cdot)$,
     where $\nu_m$ is the step size at this iteration.
5. end for
6. Output the classifier $\text{sign}(f^{[M]}(\mathbf{x}))$.
```

Algorithm 3 could also be explained in the framework of functional gradient algorithm [10] (refer to Algorithm 1 in Section 2). Let the cost function $\rho(\cdot,\cdot)$ be the log-likelihood function $l(Y,f)$ defined in Eq. (5), then perform gradient ascent in functional space in each step. Since the purpose is to maximize the objective function, in each iteration, we approximate the gradient vector rather than the negative gradient vector. By this way, we can also obtain the algorithm presented in Algorithm 3, which is referred to as Gradient ProbitBoost.

Similar to [10], we let the weak learner be $g(\mathbf{x},\mathbf{a})$, where $\mathbf{a}$ is a parameter vector. Then the third step in Algorithm 3 can be performed by an ordinary least square regression:

$$\mathbf{a}_m = \arg\min_{\mathbf{a}} \sum_{i=1}^{n} [Y_i^* - g(\mathbf{x}_i,\mathbf{a})]^2, \tag{21}$$

hence the function $g^{[m]}(\mathbf{x}) = g(\mathbf{x},\mathbf{a}_m)$ can be regarded as an approximation to the gradient in the functional space. In step 4, the step size $v_m$ can be determined by line search

$$v_m = \arg\max_{\gamma} \sum_{i=1}^{n} l[Y_i f^{[m-1]}(\mathbf{x}_i) + \gamma g^{[m]}(\mathbf{x}_i)]. \tag{22}$$

The maximization problem in Eq. (22) can be solved by 1-D search algorithms, e.g., Fibonacci search, Golden section search [26].

In the generic functional gradient descent framework (shown in Algorithm 1), it is claimed in [4] that the choice of the step size factor in step 3 is of minor importance as long as it is "small". A smaller value of fixed step size $v_m$ typically requires a larger number of boosting iterations and thus more computing time, while the predictive accuracy has been empirically found to be good when choosing $v_m$ "sufficiently small" (e.g., $v_m = 0.1$) [10]. By simulations, Friedman [10] showed that the performances are similar if the step size parameter is less than 0.125. To balance the predictive accuracy and the computational burden, Bühlmann [4] suggested choose step size as 0.1. Thus, as an alternative option for the step size, we can fix it at a small value, e.g., 0.1.

### *3.4. Multi-class versions*

In this subsection, we generalize the proposed classification algorithms to multi-class situation. Suppose we have $J$ classes, that is, the class label $Y \in \{1,2,\ldots,J\}$. We fit a classifier for each class, and denote them as functions $f_j(\mathbf{x})$, $j=1,\ldots,J$, and let $\mathbf{f} = (f_1,\ldots,f_J)'$. Assume $P(Y=j|\mathbf{x}) \propto \Phi(f_j(\mathbf{x}))$, that is

$$p_j = P(Y=j|\mathbf{x}) = \frac{\Phi(f_j(\mathbf{x}))}{\sum_{k=1}^{J}\Phi(f_k(\mathbf{x}))}. \tag{23}$$

We make prediction by taking the maximum posterior probability, that is, the predicted class label $\hat{Y}$ is determined by

$$\hat{Y} = \arg\max_{j\in\{1,2,\ldots,J\}} P(Y=j|\mathbf{x}) = \arg\max_{j\in\{1,2,\ldots,J\}} f_j(\mathbf{x}).$$

We define binary variable $y_j = I(Y=j)$, for $j=1,2,\ldots,J$, where $I(\cdot)=1$ if the condition in the parenthesis is correct, and 0 otherwise. Obviously, for each example $\mathbf{x}$, there is only one $y_j=1$, and other $y_j$'s are all 0. With this notation, we can write the likelihood function as

$$P(Y|\mathbf{x}) = \prod_{j=1}^{J} p_j^{y_j}.$$

Then the expected log-likelihood function is

$$\begin{aligned} \mathrm{E}[l(f_1,\ldots,f_J)|\mathbf{x}] &= \mathrm{E}\left[\sum_{j=1}^{J} y_j \log p_j \middle| \mathbf{x}\right] \\ &= \mathrm{E}\left[\sum_{j=1}^{J} y_j \log \Phi(f_j(\mathbf{x})) - \sum_{j=1}^{J} y_j \log\left(\sum_{k=1}^{J}\Phi(f_k(\mathbf{x}))\right) \middle| \mathbf{x}\right] \\ &= \mathrm{E}\left[\sum_{j=1}^{J} y_j \log \Phi(f_j(\mathbf{x})) - \log\left(\sum_{k=1}^{J}\Phi(f_k(\mathbf{x}))\right) \middle| \mathbf{x}\right], \end{aligned} \tag{24}$$

the last step is true because $\sum_{j=1}^{J} y_j = 1$, for any example $\mathbf{x}$.

The first order derivatives are

$$\begin{aligned} D_j = \frac{\partial \mathrm{E}[l(f_1,\ldots,f_J)|\mathbf{x}]}{\partial f_j} &= \mathrm{E}\left[y_j \frac{\varphi(f_j(\mathbf{x}))}{\Phi(f_j(\mathbf{x}))} - \frac{\varphi(f_j(\mathbf{x}))}{\sum_{k=1}^{J}\Phi(f_k(\mathbf{x}))} \middle| \mathbf{x}\right] \\ &= \mathrm{E}\left[\frac{\varphi(f_j(\mathbf{x}))}{\Phi(f_j(\mathbf{x}))}(y_j - p_j) \middle| \mathbf{x}\right] \quad \text{for } j=1,2,\ldots,J. \end{aligned} \tag{25}$$

We define $D(\mathbf{f}) = (D_1,D_2,\ldots,D_J)'$, then the Newton–Raphson update equation is

$$\mathbf{f}^{[m+1]} = \mathbf{f}^{[m]} - H(\mathbf{f}^{[m]})^{-1} D(\mathbf{f}^{[m]}), \tag{26}$$

where $\mathbf{f}^{[m]}$ is the currently fitted functions, $D(\mathbf{f}^{[m]})$ and $H(\mathbf{f}^{[m]})$ are the gradient and the Hessian evaluated at the current fitting, respectively. Unlike the algorithm presented in Section 3.2, where we updated the fitted function efficiently by a weighted least square regression, it is not obvious to do so in Eq. (26) because the Hessian is a $J\times J$ matrix. As such, we choose to use the first order gradient method for maximizing the expected log likelihood function, that is

$$\mathbf{f}^{[m+1]} = \mathbf{f}^{[m]} + v_m D(\mathbf{f}^{[m]}),$$

with $v_m$ being the step size parameter at iteration $m$. For each classifier $f_j$, we have the updating equation

$$f_j^{[m+1]}(\mathbf{x}) = f_j^{[m]}(\mathbf{x}) + v_m \mathrm{E}\left[\frac{\varphi(f_j^{[m]}(\mathbf{x}))}{\Phi(f_j^{[m]}(\mathbf{x}))}(y_j - p_j^{[m]}) \middle| \mathbf{x}\right], \tag{27}$$

where $p_j^{[m]}$ is calculated according to Eq. (23). Eq. (27) shows that the function $f_j(\mathbf{x})$ could be updated efficiently by fitting a least square regression with response variable

$$y_j^* = \frac{\varphi(f_j^{[m]}(\mathbf{x}))}{\Phi(f_j^{[m]}(\mathbf{x}))}(y_j - p_j^{[m]}).$$

The above procedure can be applied to the sample version, and is summarized as the algorithm presented in Algorithm 4. As before, the step size parameter $v_m$ can be fixed at a small value or determined by 1-D search, such as golden section search.

**Algorithm 4.** Multi-class Gradient ProbitBoost.

```
0. Given the total iteration number $M$, training data
   $\{(\mathbf{x}_i,Y_i), i=1,\ldots,n\}$ with $\mathbf{x}_i \in \mathbf{R}^d$ and $Y_i \in \{1,2,\ldots,J\}$. Initialize
   $f_j^{[0]}(\mathbf{x})=0$ for $j=1,2,\ldots,J$; define $y_{ij}=I(Y_i=j)$, for
   $i=1,2,\ldots,n$ and $j=1,2,\ldots,J$.
1. for $m=1$ to $M$ do:
2.   Calculate the probability values
       $p_{ij} = \frac{\Phi(f_j^{[m-1]}(\mathbf{x}_i))}{\sum_{k=1}^{J}\Phi(f_k^{[m-1]}(\mathbf{x}_i))}$  for $i=1,2,\ldots,n$ and $j=1,2,\ldots,J$.
3.   Calculate the transformed response as
       $y_{ij}^* = \frac{\varphi(f_j^{[m-1]}(\mathbf{x}_i))}{\Phi(f_j^{[m-1]}(\mathbf{x}_i))}(y_{ij}-p_{ij})$  for $i=1,2,\ldots,n$ and $j=1,2,\ldots,J$.
4.   for $j=1$ to $J$ do:
5.     Fit a function $g_j^m(\mathbf{x})$ by least square regression of $y_{ij}^*$ on
       $\mathbf{x}_i$ with $i=1,2,\ldots,n$.
6.     Update the $j$-th function by $f_j^{[m]}(\mathbf{x}) = f_j^{[m-1]}(\mathbf{x}) + v_m g_j^m(\mathbf{x})$.
7.   end for
8. end for
9. Output the classifier $\arg\max_{j\in\{1,2,\ldots,J\}}(f_j^{[M]}(\mathbf{x}))$.
```

Alternatively, similar to [8,9], we can fit the multi-class classifier with one-versus-all approach by reducing the multi-class problem into $J$ binary classification problems. It is very easy to sketch the algorithm as the following:

**Algorithm 5.** Multi-class ProbitBoost by one-versus-all.

```
0. Given the total iteration number $M$, training data
   $\{(\mathbf{x}_i, Y_i), i = 1, \ldots, n\}$ with $\mathbf{x}_i \in \mathbf{R}^d$ and $Y_i \in \{1, 2, \ldots J\}$. Initialize
   $y_{ij} = I(Y_i = j)$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots J$.
1. for $j = 1$ to $J$ do:
2.     Fit classifier $f_j^{[M]}(\mathbf{x})$ by calling Algorithm 2 or 3 with data
       $\{(\mathbf{x}_i, y_{ij}), i = 1, \ldots, n\}$
3. end for
4. Output the classifier $\arg\max_{j \in \{1,2,\ldots,J\}}(f_j^{[M]}(\mathbf{x}))$.
```

### 3.5. Regression stump as weak learner

We have many options for the weak learners, for example, the ordinary least square regression. However, the ordinary least square regression is not stable for classification problems [11], especially when the number of variable is far more than the sample size. Furthermore, the least square regression is time consuming since matrix inversion is involved.

The most popular weak learners in conjunction with boosting are regression trees. In this paper, we use the regression tree with only two terminal nodes, i.e., regression stump, as weak learner. Not only easier to fit, regression stump was experimentally proved yielding better or equal performance than boosting large trees [8,11]. A regression stump is defined as

$$g(\mathbf{x}, a, b, \theta) = aI(\mathbf{x}^k > \theta) + b,$$

where $a$ and $b$ are coefficients, and $\theta$ is a threshold value, $k$ is the variable based on which the regression is made.

Given the training data $\{(\mathbf{x}_i, z_i), i = 1, \ldots, n\}$, and the $i$-th datum is assigned a weight $w_i$, with $\sum_{i=1}^n w_i = 1$. We fit a regression stump by minimizing the weighted square error with the objective function

$$J_k(a, b, \theta) = \sum_{i=1}^{n} w_i (z_i - g(\mathbf{x}_i^k, a, b, \theta))^2.$$

The solution to the above optimization problem is

$$b_k = \frac{\sum_i w_i z_i I(\mathbf{x}_i^k \le \theta_k)}{\sum_i w_i I(\mathbf{x}_i^k \le \theta_k)}$$

and

$$a_k = \frac{\sum_i w_i z_i I(\mathbf{x}_i^k > \theta_k)}{\sum_i w_i I(\mathbf{x}_i^k > \theta_k)} - b_k,$$

and $\theta_k$ is selected to minimize $J_k(a, b, \theta)$, the minimum of $J_k(a, b, \theta)$ is denoted as $J_{\min}(k) = J_k(a_k, b_k, \theta_k)$. Finally, the index of the selected variable is

$$k^* = \arg\min_k J_{\min}(k),$$

and the parameters $a_{k^*}$, $b_{k^*}$, and $\theta_{k^*}$ are obtained consequently.

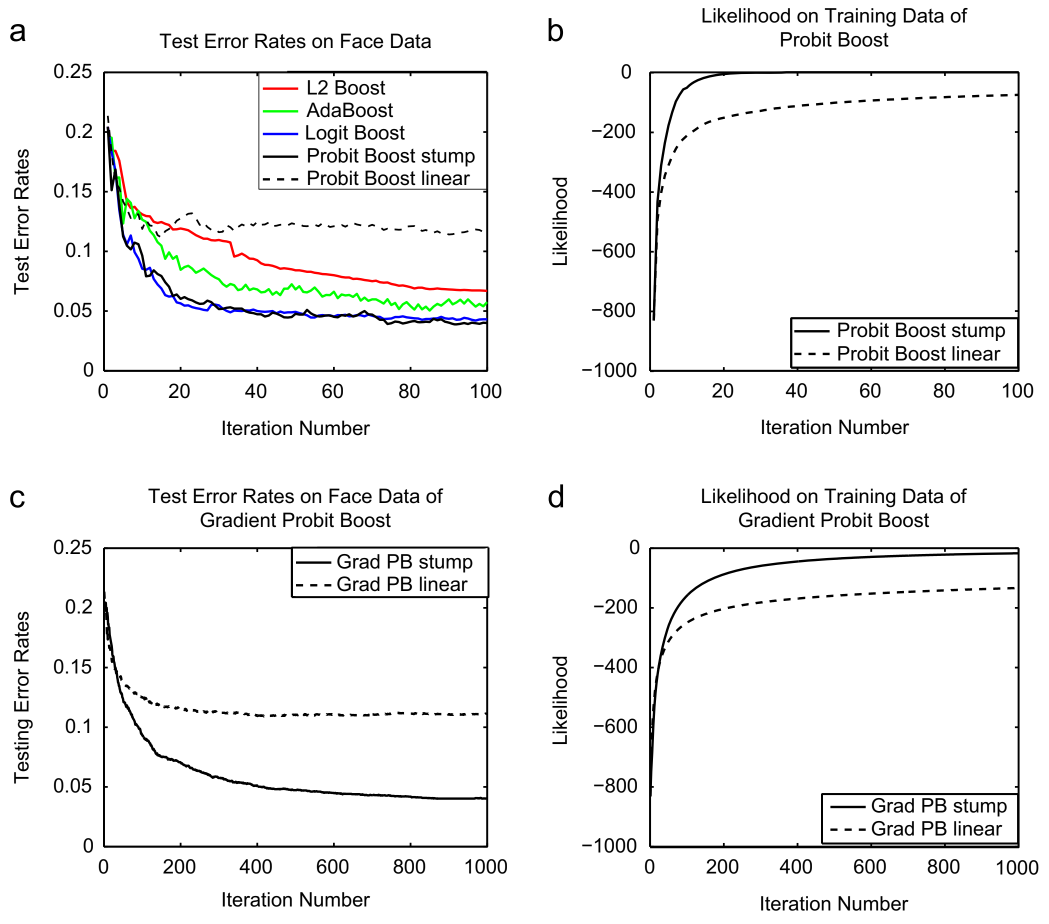

**Fig. 2.** The performance curves on face detection problem: (a) testing error curves of different boosting based algorithms; (b) the sample log-likelihood function of ProbitBoost on the training data with different weak learners; (c) testing error curves of Gradient ProbitBoost with different weak learners; and (d) the sample log-likelihood function of Gradient ProbitBoost on the training data with different weak learners.

## 4. Experimental results

In this section, we test the proposed ProbitBoost algorithms on several typical binary and multi-class classification problems, and compare the performance to the alternative algorithms. All the algorithms were implemented with MATLAB 2007b, without particular code optimization. The experiments were performed on a personal computer with Windows XP operating system, Pentium IV CPU 3.00 GHz, and 3.00 GB memory.

### 4.1. Face detection

Face detection is a classical problem in computer vision, and many machine learning algorithms have been successfully applied to face detection, e.g., support vector machines [21], AdaBoost [28]. In this experiment, we use a dataset with 3000 face and 3000 non-face images, each image is of size $16 \times 16$. The training set includes 600 face and 600 non-face images randomly selected from the whole dataset, and the rest images are used for testing. Similar to [28], we extract different types of Haar features from the images, with the minimum rectangle size as 4 by 4 (for example: 4 by 5, 8 by 4, 10 by 11), obtaining around 6000 features in total. We compare the performance of the proposed ProbitBoost algorithms to several famous boosting based methods, including AdaBoost [28], $L_2$ Boost [5], and LogitBoost [11]. The AdaBoost face detector follows [28], and all other algorithms employ regression stump [25] as weak learner since it is similar to the weak learner used in [28].

Fig. 2 gives various performance curves of the algorithms. Fig. 2(a) gives the testing error rates of different boosting based algorithms. As seen from Fig. 2(a), with regression stump as weak learner, $L_2$ Boost performs the worst, which is expected because the $L_2$ loss is not very appropriate for classification problem [12]. The performances of LogitBoost and ProbitBoost are roughly at the same level on this dataset, and AdaBoost performs worse than LogitBoost and ProbitBoost, but better than $L_2$ Boost. In the boosting based algorithms, we are free to choose the weak learners. As an illustration, we also ran ProbitBoost with simple linear regression with only one feature as weak learner. With such a weak learner, we will get a linear classifier by ProbitBoost. Fig. 2(a) also gives the testing error curve with this setting. Clearly, the result is very poor because, believably, the data is not linearly separable.

Fig. 2(b) shows the sample log-likelihood function on the training data when the ProbitBoost classifier is being trained. We observe that, as the training proceeds, the sample log-likelihood function increases monotonically, which is consistent with our theoretical analysis. We also observe that the training of ProbitBoost with regression stump converges in 20 iterations or so, and its counterpart with simple linear regression converges much more slowly. Furthermore, Fig. 2(b) shows that the algorithm with simple linear regression cannot achieve the optimal solution.

Fig. 2(c) presents the testing error curves of Gradient ProbitBoost with regression stump and simple linear regression as weak learners, and Fig. 2(d) gives the corresponding sample log-likelihood functions as the training proceeds. The algorithm was ran for 1000 iterations, with fixed small step size at 0.1. Fig. 2(c) shows that the performance of Gradient ProbitBoost is similar to that of the Newton–Raphson version of ProbitBoost, with testing error rate roughly 4% if regression stump is used, and about 11% if simple linear regression is used as weak learner. Comparing Fig. 2(d) to Fig. 2(b), we observe that Gradient ProbitBoost needs far more iterations to converge. The reason is that, Gradient ProbitBoost only uses first order derivative, and the fitted function only moves a small step in each iteration; while in the Newton–Raphson version, the second order derivative is also used, and in each iteration, the fitted function moves an optimal step.

By a close comparison of the algorithms, it is clear that the major part of ProbitBoost (Algorithm 2) is a **for** loop which includes four steps, while the major **for** loop for Gradient ProbitBoost (Algorithm 3) includes three steps; ProbitBoost has an extra weight calculating step (labeled as step 3 in Algorithm 2), while all other steps in both algorithms are similar. With the vector based implementation provided by MATLAB, the weights can be calculated efficiently. Thus, we would expect that in average, ProbitBoost only spends slightly more time than Gradient ProbitBoost per iteration. On the face dataset, we observed that, with regression stump as weak learner, running for 100 iterations, ProbitBoost spent about 169 s on our computer, and Gradient ProbitBoost used about 152 s. Therefore, our analysis and simulation verify that it is reasonable to use iteration number as a (rough) speed measure, as we did in this experiment.

For the Gradient ProbitBoost algorithm, we could have different strategies to choose the step size. On the face detection problem, we tested the performance of the algorithm with golden section search [26] for step size and the fixed small step size strategy, with regression stump as weak learner. We stop the algorithm once the improvement of the sample log-likelihood function on the training set is small (less than 0.01). We observe that with the golden section search strategy, the training algorithm converges with 50 iterations, spending 98.86 s on our computer; with the fixed step size, the algorithm converges in 940 iterations, using 1865 s. With the golden section search strategy, in every iteration, the step size is chosen to increase the sample log-likelihood function the most, thus it needs much fewer iterations to converge, compared to just moving a small step in each iteration. However, the classifier obtained from golden section search strategy has a higher testing error, 7.32%, compared to 3.86% with the fixed small step size. The reason might be that the golden section search is often too greedy, while the fixed small step size only changes the fitted function slightly in each iteration and thus could be considered as a less greedy strategy.

### 4.2. Cancer classification based on gene expression data

We test the proposed algorithms on six publicly available gene expression datasets for cancer classification: Leukemia [13], Estrogen [27], Colon [2], Lymphoma [1], Prostate [24], and DLBCL [23]. A common characteristic of these datasets is that the dimensionality is much higher than the dataset size, refer to Table 1 for details. As in the face detection problem, the considered classifiers include AdaBoost [9], $L_2$ Boost [5], and LogitBoost [11]. All the algorithms used decision stump [8] as base learner. In all the experiments, the step size parameter in Gradient ProbitBoost is fixed as 0.1.

Since all the datasets have small size $N$, leave-one-out (LOO) cross validation is carried out to estimate the classification accuracy. That is, we put aside the $i$-th observation and train the classifier on the remaining $(N-1)$ data points. We then apply the learned classifier to get $\hat{Y}_i$, the predicted class label of the $i$-th

**Table 1**
The leave-one-out error rates of the considered algorithms on the cancer classification datasets. The size and the dimensionality of each dataset are also given. The best performances are displayed in bold.

| Dataset | Size | Dimension | $L_2$ B. (%) | A. B. (%) | L. B. (%) | P. B. (%) | G. P. B. (%) |
|---|---|---|---|---|---|---|---|
| Leukemia | 72 | 3571 | 12.5 | 4.17 | **2.78** | **2.78** | 4.17 |
| Estrogen | 49 | 7129 | **6.12** | 10.20 | 12.24 | 8.16 | 10.20 |
| Colon | 62 | 2000 | **16.13** | 17.74 | **16.13** | **16.13** | **16.13** |
| Lymphoma | 96 | 4026 | 14.58 | **8.33** | 12.50 | 11.46 | 12.50 |
| Prostate | 102 | 5966 | 6.86 | 6.86 | 4.90 | **3.92** | 4.90 |
| DLBCL | 77 | 6285 | 9.09 | 6.49 | **3.90** | **3.90** | 5.19 |

observation. This procedure is repeated for all the $N$ observations in the dataset, so that each one is held out and predicted exactly once. The LOO error rate is determined by

$$E_{LOO} = \frac{1}{N}\sum_{i=1}^{N} I(\hat{Y}_i \neq Y_i).$$

Table 1 shows the LOO classification error rates of the considered classifiers on the six datasets, and the best performance is displayed in bold. From Table 1, it is readily observed that on four out of the six datasets, ProbitBoost has the best performance, and Gradient ProbitBoost performs very similar to ProbitBoost, usually with only slightly higher error rate.

Yang et al. [29] reported the performance of support vector machines (SVM) on Leukemia, DLBCL, and Prostate datasets, and the LOO error rates are 2.78%, 5.19%, and 5.90%, respectively. Compared to SVM, the proposed ProbitBoost and Gradient ProbitBoost perform better or the same. Yang et al. [29] performed gene selection as a preprocessing step while the proposed algorithms did not have such a preprocessing step. Although the main focus of this paper is not to compare SVM and the proposed methods experimentally, theoretical analysis makes it clear that the advantage of ProbitBoost or Gradient ProbitBoost over SVM is that we can select any weak classifier suitable of the specific application problem, but it is usually non-trivial to select an appropriate kernel function for SVM. Furthermore, the kernel functions of SVM often involve multiple parameters which are usually difficult to estimate.

In LOO cross validation, the training and testing sets are highly unbalanced, which will affect the evaluation result. To provide more thorough results, we have also conducted 5-fold cross validation, in which each dataset was randomly partitioned into five parts of roughly equal size, and we run the algorithm 5 times. In every run, one part was used as the testing set, and the other four parts were used as the training set. For each algorithm on every dataset, Table 2 lists the mean error rates and the standard deviations of the error rates of the 5 runs. From Table 2, we observe that ProbitBoost yields the best performance on three out of the six datasets, and Gradient ProbitBoost achieves the best performance on the other three datasets.

### 4.3. *Handwritten digit recognition*

We test the performance of the multi-class versions of the proposed algorithms on the task of recognizing handwritten digits. This problem captured the attention of pattern recognition community for a long time and has still been a benchmark problem. The dataset consists of normalized handwritten digits (0–9), automatically scanned from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; then the images have been deslanted and size normalized, resulting in $16 \times 16$ gray-scale images [17]. Since our purpose is not to develop a digit recognition system, we would not extract any specific features, instead, we simply use these 256 pixel values as inputs for training and testing the algorithms. The dataset consists of 7291 training examples and 2007 test examples, and is available at http://www-stat.stanford.edu/~tibs/ElemStatLearn/. This is a 10-class classification problem, thus we have to fit 10 classifiers. We use the multi-class error rate as the performance measure.

Fig. 3(a) shows the training and testing error curves of the multi-class Gradient ProbitBoost (Algorithm 4) with the step size parameter fixed at 0.1, with regression stump as weak learner. We observe from the curves that initially, the training and testing error decrease fast, and after about 1000 iterations, the improvement becomes small. This indicates that the algorithm converges with about 1000 iterations.

**Table 2**
The 5-fold cross validation mean error rates and the standard deviations of the considered algorithms on the datasets. The best mean error rates for a dataset are display in bold.

| Dataset | $L_2$ Boost | | AdaBoost | | LogitBoost | | ProbitBoost | | Gradient ProbitBoost | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Mean (%) | Std (%) | Mean (%) | Std (%) | Mean (%) | Std (%) | Mean (%) | Std (%) | Mean (%) | Std (%) |
| Leukemia | 10.00 | 3.91 | **2.86** | 8.15 | 7.14 | 8.75 | 4.28 | 3.91 | **2.86** | 3.91 |
| Estrogen | 15.55 | 9.30 | **8.89** | 9.30 | 13.33 | 6.09 | **8.89** | 9.30 | 11.11 | 11.11 |
| Colon | 25.00 | (0.21) | 21.67 | 9.50 | **16.67** | 10.21 | **16.67** | (10.21) | 19.99 | 7.47 |
| Lymphoma | 15.79 | 5.77 | 8.42 | 4.71 | 9.47 | 4.40 | **7.37** | 3.91 | 8.42 | 4.40 |
| Prostate | 13.00 | 8.94 | 11.00 | 8.54 | 7.00 | 4.47 | 5.00 | 3.54 | **3.00** | 4.47 |
| DLBCL | 14.67 | 10.95 | 8.00 | 2.98 | 10.67 | 7.60 | 6.67 | 6.67 | **5.33** | 5.58 |

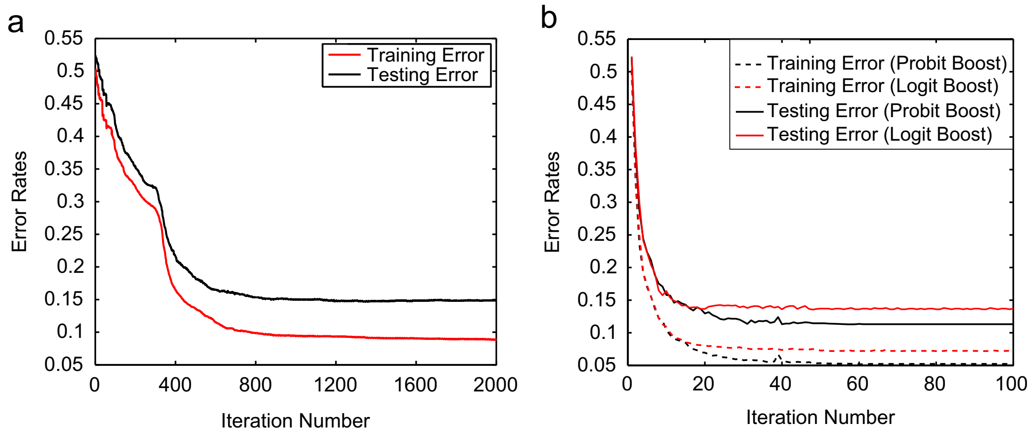

**Fig. 3.** The performance curves on handwritten digit dataset: (a) the training and testing error curves of multi-class Gradient ProbitBoost (Algorithm 4) and (b) comparison of multi-class LogitBoost and ProbitBoost with one-versus-all strategy (Algorithm 5).

Fig. 3(b) presents the performance curves of the one-versus-all version of multi-class ProbitBoost (Algorithm 5) and multi-class LogitBoost [8]. First of all, we notice that the one-versus-all approach converges very quickly, both LogitBoost and ProbitBoost converge in 60 iterations. This is because both algorithms use the Newton–Raphson approach which converges faster than the first order method used in Algorithm 4. Secondly, compared to the approach of fitting all the classifiers simultaneously which is presented in Fig. 3(a), the one-versus-all method can achieve much better performance, this could also be seen from Table 3, which gives the final training and testing error rates of the considered algorithms. The reason might be that in the one-versus-all approach, the training algorithm only focuses on one class in each iteration, thus it can learn better. Finally, we notice that, on this particular dataset, ProbitBoost perform better than LogitBoost classifier.

## 5. Discussion

In the literature, it is commonly agreed that the generalized linear model with Probit link function and Logit link function perform similarly [6,15]. As algorithms to fit the Probit and Logit models, ProbitBoost and LogitBoost are expected to perform similarly, and our experimental results support this conjecture. Yet from theoretical point of view, we could find some differences among ProbitBoost, LogitBoost, and AdaBoost. All three algorithms have a re-weighting step, and the re-weighted samples are fed to the next iteration. The weight functions are summarized in Table 4, in which $f$ is the fitted function at this stage. To gain an intuitive idea, Fig. 4 presents the weight functions of AdaBoost and LogitBoost, refer to Fig. 1 for the weight function of ProbitBoost.

From the formulas and graphs of the weight functions, we observe that the weights of AdaBoost and ProbitBoost depend on the fitted function and the class label of the examples in a way such that if an example is mistakenly classified, its weight will be increased (this is particularly true for AdaBoost). LogitBoost gives small weights to the examples about which it has high confidence (the fitted function is far away from 0), and it gives big weights to examples about which it has low confidence (i.e., fitted function values are around 0). However, if the LogitBoost algorithm makes a severe mistake (i.e., the fitted function value is a very large positive number for a negative example or a very large negative number for a positive example), it will give this mistakenly classified example a small weight, and this makes it difficult to correct the error in the successive iterations. In this sense, we could say that AdaBoost and ProbitBoost have better error correction ability than LogitBoost.

In the presented experiments, since only the first order derivative information is employed, Gradient ProbitBoost did not perform as well as the Newton–Raphson based ProbitBoost which uses the second order derivative information and moves the fitted function an optimal step in each iteration. For binary Probit regression, we are lucky to find an explicit and easy-to-implement way to evaluate the second order derivative, thus, the Newton–Raphson based method could be applied. However, in some cases, the second order derivative is not easy to calculate or unstable, then the first order based method is preferred. For example, the boosting based quantile regression presented in [16,32,33] only uses the first order derivative, and it is an algorithm similar to Gradient ProbitBoost introduced in this paper. As another example, Algorithm 4 presented in this paper tries to fit all the classifiers simultaneously for multi-class problem, in which the Hessian is a $J\times J$ matrix, and it is not obvious to develop an explicit Newton–Raphson method to maximize the objective function in Eq. (24), thus the first order derivative based gradient method is a better choice.

**Table 3**
The training and testing error rates of several multi-class algorithms on the handwritten digit recognition problem. The best performances on this problem are displayed in bold.

| Algorithm | LogitBoost (one vs. all) (%) | ProbitBoost (one vs. all) (%) | ProbitBoost (All) (%) |
|---|---|---|---|
| Training error | 7.21 | **5.17** | 8.57 |
| Testing error | 13.55 | **11.26** | 14.56 |

**Table 4**
The weight functions for AdaBoost, LogitBoost, and ProbitBoost.

| Algorithm | Weight functions |
|---|---|
| AdaBoost | $W(f)\propto \exp(-Yf(\mathbf{x}))$ |
| LogitBoost | $W(f)=p(1-p)\propto \dfrac{1}{[\exp(-f(\mathbf{x}))+\exp(f(\mathbf{x}))]^2}$ |
| ProbitBoost | $W(f)\propto -\dfrac{\phi(f(\mathbf{x}))G_Y(f(\mathbf{x}))}{\Phi^2((2Y-1)f(\mathbf{x}))}$ |

## 6. Conclusion and future works

In the framework of functional gradient ascent, this paper introduces two approaches for maximizing the sample log-likelihood function of the Probit regression model. The first approach,

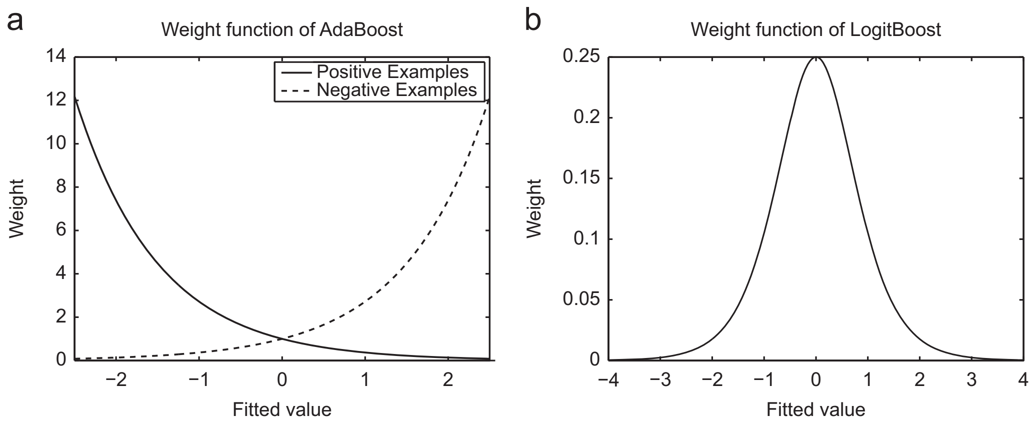

**Fig. 4.** (a) The weight function of AdaBoost and (b) the weight function of LogitBoost.

ProbitBoost, is based on Newton–Raphson method, which updates the fitted model by a weighted least square regression in each iteration. The second algorithm, Gradient ProbitBoost, only uses the first order gradient for updating the fitted model in the gradient direction by a small step in each iteration. We also investigate two approaches to generalize the methods to multi-class cases, which are fitting multiple classifiers simultaneously using the first order gradient information, and applying the one-versus-all scheme to reduce the multi-class problem into a series of binary classification problems.

The proposed algorithms were tested on typical classification problems including face detection, cancer classification based on gene expression data, and handwritten digit recognition. Each of the problems has high dimensional features. Detailed performance analysis shows that in general, the Newton–Raphson based ProbitBoost performs better in terms of both convergence speed and classification error rates than Gradient ProbitBoost which only uses the first order gradient information. On the considered classification problems, the proposed algorithms perform similar to or better than the alternative methods. To the best of our knowledge, this is the first attempt to use functional gradient optimization method for fitting Probit regression model, and the resulting algorithms are able to work in high dimensional spaces and easy to implement with performance comparable to state-of-the-art.

Currently, the proposed algorithms only have "forward" steps, that is, they select features which cause direct increasing of the sample log-likelihood function. Moreover, the proposed algorithms are in the framework of gradient ascent, which is greedy. Thus, it is possible that the algorithms are too greedy and pick up some irrelevant features in some iterations. In the literature, there are "backward" steps in model fitting [12,18,30]. As such, it is well motivated to introduce "backward" step in the algorithms, that is, deleting some variables at certain stage in order to improve the classification performance, and this is our next step of research.

## Acknowledgment

W. Liu was supported by SZU R/D Fund (201054) and the National Natural Science Foundation of China (60903113). The authors would like to extend their gratitude to the anonymous reviewers for their constructive suggestions and comments, which have greatly helped improve the quality of this paper.

## References

[1] A.A. Alizadeh, M.B. Eisen, et al., Distinct types of diffuse large B-cell Lymphoma identified by gene expression profiling, Nature 403 (6769) (2000) 503–511.
[2] U. Alon, N. Barkai, D.A. Notterman, et al., Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, Proceedings of the National Academy of Sciences USA 96 (1999) 6745–6750.
[3] D. Böhning, The lower bound method in probit regression, Computational Statistics & Data Analysis 30 (1) (1999) 13–17.
[4] P. Bühlmann, T. Hothorn, Boosting algorithms: regularization, prediction and model fitting, Statistical Science 22 (2007) 477–505.
[5] P. Bühlmann, B. Yu, Boosting with the $L_2$ loss: regression and classification, Journal of the American Statistical Association 98 (2003) 324–340.
[6] E.A. Chambers, D.R. Cox, Discrimination between alternative binary response models, Biometrika 54 (3/4) (1967), 573–578.
[7] S. Chakraborty, Bayesian binary kernel probit model for microarray based cancer classification and gene selection, Computational Statistics & Data Analysis 53 (12) (2009) 4198–4209.
[8] M. Dettling, P. Bühlmann, Boosting for tumor classification with gene expression data, Bioinformatics 19 (9) (2003) 1061–1069.
[9] Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.
[10] J.H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of Statistics 29 (2001) 1189–1232.
[11] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting, Annal of Statistics 28 (2000) 337–407.
[12] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd edition, Springer, New York, 2009.
[13] T.R. Golub, D.K. Slonim, P. Tamayo, et al., Molecular classification of cancer: class discovery and class prediction by gene expression, Science 286 (1999), 531–537.
[14] P.J. Green, Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives, Journal of the Royal Statistical Society. Series B (Methodological) 46 (2) (1984) 149–192.
[15] E.D. Hahn, R. Soyer, Probit and logit models: differences in a multivariate realm, Retrieved May 18, 2012, from ⟨http://home.gwu.edu/∼soyer/mv1h.pdf⟩.
[16] B. Kriegler, R. Berk, Boosting the Quantile Distribution: A Cost-sensitive Statistical Learning Procedure, Technical Report, Department of Statistics, University of California, Los Angeles, 2007.
[17] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Handwritten digit recognition with a back-propagation network, in: Proceedings of Advances in Neural Information Processing Systems, 1990, pp. 396–404.
[18] S.Z. Li, Z. Zhang, Floatboost learning and statistical face detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (9) (2004) 1112–1123.
[19] C. Liu, D.B. Rubin, Y. Wu, Parameter expansion to accelerate EM—the PX-EM algorithm, Biometrika 85 (1998) 755–770.
[20] J.S. Liu, Y. Wu, Parameter expansion for data augmentation, Journal of the American Statistical Association 94 (1999) 1264–1274.
[21] E. Osuna, R. Freund, F. Girosit, Training support vector machines: an application to face detection, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1997.
[22] N. Sha, M. Vannucci, M.G. Tadesse, P.J. Brown, I. Dragoni, N. Davies, T.C. Roberts, A. Contestabile, N. Salmon, C. Buckley, F. Falciani, Bayesian variable selection in multinomial probit models to identify molecular signatures of disease stage, Biometrics 60 (3) (2004) 812–819.
[23] M.A. Shipp, K.N. Ross, et al., Diffuse large B-cell lymphoma outcome prediction by gene expression profiling and supervised machine learning, Nature Medicine 8 (2002) 68–74.
[24] D. Singh, P.G. Febbo, K. Ross, D.G. Jackson, J. Manola, et al., Gene expression correlates of clinical prostate cancer behavior, Cancer Cell 1 (2002) 203–209.
[25] A. Torralba, K.P. Murphy, W.T. Freeman, Sharing features: efficient boosting procedures for multiclass object detection, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2004.
[26] G.R. Walsh, Methods of Optimization, John Wiley and Sons, 1975.
[27] M. West, C. Blanchette, H. Dressman, E. Huang, S. Ishida, R. Spang, H. Zuzan, J.R. Marks, J.R. Nevins, Predicting the clinical status of human breast cancer using gene expression profiles, Proceedings of the National Academy of Sciences USA 98 (2001) 11462–11467.
[28] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2001.
[29] K. Yang, Z. Cai, J. Li, G. Lin, A stable gene selection in microarray data analysis, BMC Bioinformatics 7 (2006) 228.
[30] P. Zhao, B. Yu, Stagewise Lasso, Journal of Machine Learning Research 8 (2007) 2701–2726.
[31] X. Zhou, X. Wang, E.R. Dougherty, Multi-class cancer classification using multinomial probit regression with Bayesian variable selection, IEE Proceedings System Biology 153(2) (2006) 70–78.
[32] S. Zheng, Boosting based conditional quantile estimation for regression and binary classification, in: Proceedings of the 9th Mexican International Conference on Artificial Intelligence (MICAI), 2010.
[33] S. Zheng, QBoost: Predicting quantiles with boosting for regression and binary classification, Expert Systems With Applications 39 (2) (2012) 1687–1697.

**Songfeng Zheng** received his B.S. degree in Electrical Engineering, and M.S. in Computer Science, from Xi'an JiaoTong University, China, in 2000 and 2003, respectively. In 2008, he received Ph.D. degree in Statistics from the University of California, Los Angeles. Since 2008, he has been an assistant professor with the Department of Mathematics, Missouri State University. His research interests include statistical learning, pattern recognition, statistical computation, and image analysis.

**Weixiang Liu** received his B.Sc. and M.Sc. degrees in mechanical engineering from Xi'an Shiyou University, China, in 1997 and 2000 respectively, and the Ph.D. in electronic and information engineering from Xi'an Jiaotong University, China, in 2005. He was a researcher at Tsinghua University from 2005 to 2007. Since 2007, he has been an associate professor with the Biomedical Engineering Department at Shenzhen University, China. His research interests include machine learning, pattern recognition, computer vision, and bioinformatics.