Component-wise AdaBoost Algorithms for High-dimensional Binary Classification and Class Probability Prediction

Jianghao Chu* Tae-Hwy Lee† Aman Ullah‡

July 31, 2018

Abstract

Freund and Schapire (1997) introduced "Discrete AdaBoost"(DAB) which has been mysteriously effective for the high-dimensional binary classification or binary prediction. In an effort to understand the myth, Friedman, Hastie and Tibshirani (FHT, 2000) show that DAB can be understood as statistical learning which builds an additive logistic regression model via Newton-like updating minimization of the "exponential loss". From this statistical point of view, FHT proposed three modifications of DAB, namely, Real AdaBoost (RAB), LogitBoost (LB), and Gentle AdaBoost (GAB). All of DAB, RAB, LB, GAB solve for the logistic regression via different algorithmic designs and different objective functions. The RAB algorithm uses class probability estimates to construct real-valued contributions of the weak learner, LB is an adaptive Newton algorithm by stagewise optimization of the Bernoulli likelihood, and GAB is an adaptive Newton algorithm via stagewise optimization of the exponential loss. The same authors of FHT published an influential textbook, *The Elements of Statistical Learning* (ESL, 2001 and 2008). A companion book *An Introduction to Statistical Learning* (ISL) by James et al. (2013) was published with applications in R. However, both ESL and ISL (e.g., sections 4.5 and 4.6) do not cover these four AdaBoost algorithms while FHT provided some simulation and empirical studies to compare these methods. Given numerous potential applications, we believe it would be useful to collect the `R` libraries of these AdaBoost algorithms, as well as more recently developed extensions to AdaBoost for probability prediction with examples and illustrations. Therefore, the goal of this chapter is to do just that, i.e., (i) to provide a user guide of these alternative AdaBoost algorithms with step-by-step tutorial of using `R` (in a way similar to ISL, e.g., Section 4.6), (ii) to compare AdaBoost with alternative machine learning classification tools such as the deep neural network (DNN), logistic regression with LASSO

*Department of Economics, University of California, Riverside, 900 University Ave, Riverside, CA 92521, USA. Phone: 951-525-8996, fax: 951-827-5685, e-mail: `jianghao.chu@email.ucr.edu`.

†Department of Economics, University of California, Riverside, 900 University Ave, Riverside, CA 92521, USA. Phone: 951-827-1509, fax: 951-827-5685, e-mail: `taelee@ucr.edu`.

‡corresponding author. Department of Economics, University of California, Riverside, 900 University Ave, Riverside, CA 92521, USA. Phone: 951-827-1591, fax: 951-827-5685, e-mail: `aman.ullah@ucr.edu`.

and SIM-RODEO, and (iii) to demonstrate the empirical applications in economics, such as prediction of business cycle turning points and directional prediction of stock price indexes. We revisit Ng (2014) who used DAB for prediction of the business cycle turning points by comparing the results from RAB, LB, GAB, DNN, logistic regression and SIM-RODEO.

Keywords: AdaBoost, R, Binary classification, Logistic regression, DAB, RAB, LB, GAB, DNN.

1 Introduction

A large number of important variables in economics are binary. Let

$$\pi(x) \equiv P(y = 1|x)$$

and y takes value 1 with probability $\pi(x)$ and -1 with probability $1 - \pi(x)$. The studies on making the best forecast on y can be classified into two classes (Lahiri and Yang, 2012). One is focusing on getting the right probability model $\hat{\pi}(x)$, e.g., logit and probit models (Bliss, 1934; Cox, 1958; Walker and Duncan, 1967), then making the forecast on y with $\hat{\pi}(x) > 0.5$ using the estimated probability model. The other is to get the optimal forecast rule on y directly, e.g., the maximum score approach (Manski, 1975, 1985; Elliott and Lieli, 2013), without having to (correctly) model the probability $\hat{\pi}(x)$.

Given the availability of high-dimensional data, the binary classification or binary probability prediction problems can be improved by incorporating a large number of covariates (x). A number of new methods are proposed to take advantage of the great number of covariates. Freund and Schapire (1997) introduce machine learning method called Discrete AdaBoost algorithm, which takes a functional descent procedure and selects the covariates (or predictors) sequentially. Friedman et al. (2000) show that AdaBoost can be understood as a regularized logistic regression, which selects the covariates one-at-a-time. The influential paper also discusses several extensions to the original idea of Discrete AdaBoost and proposes new Boosting methods, namely Real AdaBoost, LogitBoost and Gentle Boost, which uses the exponential loss or Bernoulli log-likelihood as fitting criteria. Later on, Friedman (2001) generalize the idea to any fitting criteria and proposes the Gradient Boosting Machine. Bühlmann and Yu (2003) and Bühlmann (2006) propose the L_2 Boost and prove its consistency for regression and classification. Mease et al. (2007) use the logistic function to convert the class label output of boosting algorithms into probability and/or quantile predictions. Chu et al. (2018a) show the linkage between the Discrete AdaBoost and the maximum score approach and propose Asymmetric AdaBoost for utility based high-dimensional binary classification.

On the other hand, efforts are made to incorporate traditional binary classification and probability prediction methods into the high-dimensional sparse matrix set-up. The key feature of high-dimensional data is the redundancy of covariates in the data. Hence, methods are proposed to select useful covariates while/before estimation of the models. Tibshirani (1996) proposes the LASSO that is to add L_1 penalty to including more covariates in the

model. Zou (2006) drives a necessary condition for consistency of the LASSO variable selection and proposes the Adaptive LASSO which is showed to enjoy oracle property. LASSO type methods are often used with parametric models such as linear model or logistic model. To relax the parametric assumptions, Lafferty and Wasserman (2008) propose the Regularization of the Derivative Expectation Operator (RODEO) for variable selection in kernel regression. Chu et al. (2018b) proposes SIM-RODEO for variable selection in semiparametric single-index model. See Su and Zhang (2014) for a thorough review of variable selection in nonparametric and semiparametric models.

This paper gives a overview of recently developed machine learning methods, namely AdaBoost in the role of binary prediction. AdaBoost algorithm focuses on making the optimal forecast directly without modeling the conditional probability of the events. AdaBoost gets an additive model by iteratively minimizing an exponential loss function. In each iteration, AdaBoost puts more weights on the observations that cannot be predicted correctly using the previous predictors. Moreover, AdaBoost algorithm is able to solve classification problem with high-dimensional data which is an advantage to traditional classification methods.

The rest of the paper is organized as follow. In Section 2 we provide a brief introduction of AdaBoost from minimizing the 'exponential loss'. In Section 3 we show popular variants of AdaBoost. Section 5 gives numerical examples of the boosting algorithms. Section 6 compares the mentioned boosting algorithms with Deep Neural Network (DNN) and logistic regression with LASSO. Section 7 concludes.

2 AdaBoost

The algorithm of AdaBoost is as shown in Algorithm 1. Let y be the binary class taking a value in $\{-1, 1\}$ that we wish to predict. Let $f_m(x)$ be the weak learner (weak classifier) for the binary target y that we fit to predict using the high-dimensional covariates x in the mth iteration. Let err_m denote the error rate of the weak learner $f_m(x)$, and $E_w(\cdot)$ denote the weighted expectation (to be defined below) of the variable in the parenthesis with weight w. Note that the error rate $E_w\left[1_{(y \neq f_m(x))}\right]$ is estimated by $err_m = \sum_{i=1}^{n} w_i 1_{(y_i \neq f_m(x_i))}$ with the weight w_i given by step 2(c) from the previous iteration. n is the number of observations. The symbol $1_{(\cdot)}$ is the indicator function which takes the value 1 if a logical condition inside the parenthesis is satisfied and takes the value 0 otherwise. The symbol $\text{sign}(z) = 1$ if $z > 0$, $\text{sign}(z) = -1$ if $z < 0$, and hence $\text{sign}(z) = 1_{(z>0)} - 1_{(z<0)}$.

Algorithm 1 Discrete AdaBoost (DAB, Freund and Schapire, 1997)

1. Start with weights $w_i = \frac{1}{n}, i = 1, \ldots, n$.
2. For $m = 1$ to M
   (a) For $j = 1$ to k (for each variable)
      i. Fit the classifier $f_{mj}(x_{ij}) \in \{-1, 1\}$ using weights w_i on the training data.
      ii. Compute $err_{mj} = \sum_{i=1}^{n} w_i 1_{(y_i \neq f_{mj}(x_{ji}))}$.
   (b) Find $\hat{j}_m = \arg\min_j err_{mj}$
   (c) Compute $c_m = \log\left(\frac{1-err_{m,\hat{j}_m}}{err_{m,\hat{j}_m}}\right)$.
   (d) Set $w_i \leftarrow w_i \exp[c_m 1_{(y_i \neq f_{m,\hat{j}_m}(x_{\hat{j}_m,i}))}]$, $i = 1, \ldots, n$, and normalize so that $\sum_{i=1}^{n} w_i = 1$.
3. Output the binary classifier $\text{sign}[F_M(x)]$ and the class probability prediction $\hat{\pi}(x) = \frac{e^{F_M(x)}}{e^{F_M(x)}+e^{-F_M(x)}}$ where $F_M(x) = \sum_{m=1}^{M} c_m f_{m,\hat{j}_m}(x_{\hat{j}_m})$.

Remark 1. Note that the presented version of Discrete AdaBoost as well as Real AdaBoost (RAB), LogitBoost (LB) and Gentle AdaBoost (GAB) which will be introduced later in the paper are different from their original version when they are first introduced. The original version of these algorithms only output the class label. In this paper, we follow the idea of Mease et al. (2007) and modified the algorithms to output both the class label and probability prediction. The probability prediction is attained using

$$\hat{\pi}(x) = \frac{e^{F_M(x)}}{e^{F_M(x)} + e^{-F_M(x)}}, \tag{1}$$

where $F_M(x)$ is the sum of weak learner in the algorithms.

The most widely used weak learner is the classification tree. The simplest classification tree, the stump, takes the following functional form

$$f(x_j, a) = \begin{cases} 1 & x_j > a \\ -1 & x_j < a \end{cases}$$

where the parameter a is found by minimizing the error rate

$$\min_a \sum_{i=1}^{n} w_i 1\left(y_i \neq f\left(x_{ji}, a\right)\right). \tag{2}$$

In addition to the commonly used classification tree weak learners in machine learning literature described above, Discrete AdaBoost, in principle, can take any classifier and boost

its performance through the weighted voting scheme. For example, we can also use a one-variable Logistic Regression as a weak learner which we will call the logistic weak learner. Simulation results of Chu et al. (2018a) show that the logistic weak learner generally has better performance than the stump in traditional econometric models. In the logistic weak learner, we assume the probability

$$\pi\left(x_j\right) \equiv P\left(y=1|x_j\right) = \frac{e^{x_j\beta}}{1+e^{x_j\beta}}.$$

Let $Y = \frac{y+1}{2} \in \{0,1\}$. We estimate the parameter β by maximizing the weighted logistic log-likelihood function

$$\begin{aligned}
\max_{\beta} \log L &= \log \prod_{i=1}^{n} \left[\left(\frac{e^{x_{ji}\beta}}{1+e^{x_{ji}\beta}} \right)^{Y_i} \left(\frac{1}{1+e^{x_{ji}\beta}} \right)^{1-Y_i} \right]^{w_i} \\
&= \log \prod_{i=1}^{n} \left(\frac{e^{Y_i x_{ji}\beta}}{1+e^{x_{ji}\beta}} \right)^{w_i} \qquad (3) \\
&= \sum_{i=1}^{n} \log \left(\frac{e^{Y_i x_{ji}\beta}}{1+e^{x_{ji}\beta}} \right)^{w_i} \\
&= \sum_{i=1}^{n} w_i \left[Y_i x_{ji}\beta - \log\left(1+e^{x_{ji}\beta}\right) \right]. \qquad (4)
\end{aligned}$$

Then the resulting logistic weak learner will be

$$f\left(x_j, \beta, \tau\right) = \begin{cases} 1 & \pi\left(x_j, \beta\right) > 0.5 \\ -1 & \pi\left(x_j, \beta\right) < 0.5. \end{cases}$$

Several packages in `R` provide off-the-shelf implementations of Discrete AdaBoost. *JOUSBoost* gives an implementation of the Discrete AdaBoost algorithm from Freund and Schapire (1997) applied to decision tree classifiers and provides a convenient function to generate test sample of the algorithms.

Here we use the *circle_data* function from *JOUSBoost* to generate a test sample. The *circle_data* function simulate draws from a Bernoulli distribution over $\{-1,1\}$. First, the predictors x are drawn i.i.d. uniformly over the square in the two dimensional plane centered at the origin with side length $2 * outer_r$, and then the response is drawn according to $\pi(x)$, which depends on $r(x)$, the euclidean norm of x. If $r(x) \leq inner_r$, then $\pi(x) = 1$, if $r(x) \geq outer_r$ then $\pi(x) = 0$, and $\pi(x) = (outer_r - r(x))/(outer_r - inner_r)$ when $inner_r \leq r(x) \leq outer_r$ as in Mease et al. (2007). The code of the function is shown below.

```
circle_data <- function (n = 500, inner_r = 8, outer_r = 28) {
        if (outer_r <= inner_r)
        stop("outer_r must be strictly larger than inner_r")
        X = matrix(stats::runif(2 * n, -outer_r, outer_r), nrow = n,
        ncol = 2)
```

```
        r = apply(X, 1, function(x) sqrt(sum(x^2)))
        p = 1 * (r < inner_r) + (outer_r - r)/(outer_r - inner_r) *
        ((inner_r < r) & (r < outer_r))
        y = 2 * stats::rbinom(n, 1, p) - 1
        list(X = X, y = y, p = p)
}
```

Then we use the implementation of Discrete AdaBoost from *ada* package since the *ada* package provides implementation of not only Discrete AdaBoost and also Real AdaBoost, LogitBoost and Gentle AdaBoost which we will discuss about in the next section.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 500)
x <- dat$X
y <- dat$y

library(ada)
model <-  ada(x, y, loss = "exponential", type = "discrete", iter = 200)
print(model)
```

where y and x are the training samples, and *iter* controls the number of boosting iterations.

Remark 2. The algorithms in *ada* for Discrete AdaBoost, Real AdaBoost, LogitBoost and Gentle Boost may not follow exactly the same steps and/or criteria as described in the paper. However, the major settings, the loss function and characteristics of weak learners, are the same. We choose to use the *ada* package since it's widely accessible and easy to use for the readers.

The output is as follow.

```
Call:
ada(x, y = y, loss = "exponential", type = "discrete", iter = 200)

Loss: exponential Method: discrete   Iteration: 200

Final Confusion Matrix for Data:
        Final Prediction
True value  -1   1
        -1 300  14
        1   15 171

Train Error: 0.058

Out-Of-Bag Error:  0.094  iteration= 195
```

```
Additional Estimates of number of iterations:

train.err1 train.kap1
197        197
```

Other packages include *fastAdaboost* which uses C++ code in the backend to provide an implementation of AdaBoost that is about 100 times faster than native `R` based libraries.

```
library(fastAdaboost)
adaboost(y~x, nIter)
```

where y and x are the training samples and $nIter$ is the number of boosting iterations. Note that *fastAdaboost* also contains implementation of Real AdaBoost which we will introduce later. *GBM* which is short for Generalized Boosting Regression Models contains implementation of extensions to Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine.

Friedman et al. (2000) show that AdaBoost builds an additive logistic regression model

$$F_M(x) = \sum_{m=1}^{M} c_m f_m(x) \tag{5}$$

via Newton-like updates for minimizing the exponential loss

$$J(F) = E\left(e^{-yF(x)}|x\right). \tag{6}$$

We use greedy method to minimize the exponential loss function iteratively. After m iterations, the current classifier is denoted as $F_m(x) = \sum_{s=1}^{m} c_s f_s(x)$. In the next iteration, we are seeking an update $c_{m+1} f_{m+1}(x)$ for the function fitted from previous iterations $F_m(x)$. The updated classifier would take the form

$$F_{m+1}(x) = F_m(x) + c_{m+1} f_{m+1}(x).$$

The loss for $F_{m+1}(x)$ will be

$$\begin{aligned} J(F_{m+1}(x)) &= J(F_m(x) + c_{m+1} f_{m+1}(x)) \\ &= E\left[e^{-y(F_m(x) + c_{m+1} f_{m+1}(x))}\right]. \end{aligned} \tag{7}$$

Expand w.r.t. $f_{m+1}(x)$

$$\begin{aligned} J(F_{m+1}(x)) &\approx E\left[e^{-yF_m(x)}\left[1 - yc_{m+1} f_{m+1}(x) + \frac{y^2 c_{m+1}^2 f_{m+1}^2(x)}{2}\right]\right] \\ &= E\left[e^{-yF_m(x)}\left(1 - yc_{m+1} f_{m+1}(x) + \frac{c_{m+1}^2}{2}\right)\right]. \end{aligned}$$

The last equality holds since $y \in \{-1,1\}$, $f_{m+1}(x) \in \{-1,1\}$, and $y^2 = f_{m+1}^2(x) = 1$. $f_{m+1}(x)$ only appears in the second term in the parenthesis, so minimizing the loss function (7) w.r.t. $f_{m+1}(x)$ is equivalent to maximizing the second term in the parenthesis which results in the following conditional expectation

$$\max_f E\left[e^{-yF_m(x)} y c_{m+1} f_{m+1}(x) | x\right].$$

For any $c > 0$ (we will prove this later), we can omit c_{m+1} in the above objective function

$$\max_f E\left[e^{-yF_m(x)} y f_{m+1}(x) | x\right].$$

To compare it with the Discrete AdaBoost algorithm, here we define weight $w = w(y,x) = e^{-yF_m(x)}$. Later we will see that this weight w is equivalent to that shown in the Discrete AdaBoost algorithm. So the above optimization can be seen as maximizing a weighted conditional expectation

$$\max_f E_w\left[y f_{m+1}(x) | x\right] \tag{8}$$

where $E_w(y|x) := \frac{E(wy|x)}{E(w|x)}$ refers to a weighted conditional expectation. Note that (8)

$$\begin{aligned} & E_w\left[y f_{m+1}(x) | x\right] \\ = \; & P_w(y=1|x) f_{m+1}(x) - P_w(y=-1|x) f_{m+1}(x) \\ = \; & \left[P_w(y=1|x) - P_w(y=-1|x)\right] f_{m+1}(x) \\ = \; & E_w(y|x) f_{m+1}(x). \end{aligned}$$

where $P_w(y|x) = \frac{E(w|y,x)P(y|x)}{E(w|x)}$. Solve the maximization problem (8). Since $f_{m+1}(x)$ only takes 1 or -1, it should be positive whenever $E_w(y|x)$ is positive and -1 whenever $E_w(y|x)$ is negative. The solution for $f_{m+1}(x)$ is

$$f_{m+1}(x) = \begin{cases} 1 & E_w(y|x) > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Next, minimize the loss function (7) w.r.t. c_{m+1}

$$\begin{aligned} c_{m+1} &= \arg\min_{c_{m+1}} E_w\left(e^{-c_{m+1} y f_{m+1}(x)}\right) \\ E_w\left(e^{-c_{m+1} y f_{m+1}(x)}\right) &= P_w(y = f_{m+1}(x)) e^{-c_{m+1}} + P_w(y \neq f_{m+1}(x)) e^{c_{m+1}} \\ \frac{\partial E_w\left(e^{-cy f_{m+1}(x)}\right)}{\partial c} &= -P_w(y = f_{m+1}(x)) c_{m+1} e^{-c_{m+1}} + P_w(y \neq f_{m+1}(x)) c_{m+1} e^{c_{m+1}} \end{aligned}$$

Let

$$\frac{\partial E_w\left(e^{-c_{m+1} y f_{m+1}(x)}\right)}{\partial c_{m+1}} = 0,$$

and we have

$$P_w\left(y = f_{m+1}\left(x\right)\right) c_{m+1} e^{-c_{m+1}} = P_w\left(y \neq f_{m+1}\left(x\right)\right) c_{m+1} e^{c_{m+1}},$$

Solve for c_{m+1}, we obtain

$$c_{m+1} = \frac{1}{2} \log \frac{P_w\left(y = f_{m+1}\left(x\right)\right)}{P_w\left(y \neq f_{m+1}\left(x\right)\right)} = \frac{1}{2} \log\left(\frac{1 - err_{m+1}}{err_{m+1}}\right),$$

where $err_{m+1} = P_w\left(y \neq f_{m+1}\left(x\right)\right)$ is the error rate of $f_{m+1}\left(x\right)$. Note that $c_{m+1} > 0$ as long as the error rate is smaller than 50%. Our assumption $c_{m+1} > 0$ holds for any learner that is better than random guessing.

Now we have finished the steps of one iteration and can get our updated classifier by

$$F_{m+1}\left(x\right) \leftarrow F_m\left(x\right) + \left(\frac{1}{2} \log\left(\frac{1 - err_{m+1}}{err_{m+1}}\right)\right) f_{m+1}\left(x\right).$$

Note that in the next iteration, the weight we defined w_{m+1} will be

$$w_{m+1} = e^{-yF_{m+1}(x)} = e^{-y(F_m(x) + c_{m+1} f_{m+1}(x))} = w_m \times e^{-c_{m+1} f_{m+1}(x) y}.$$

Since $-y f_{m+1}\left(x\right) = 2 \times 1_{\{y \neq f_{m+1}(x)\}} - 1$, the update is equivalent to

$$w_{m+1} = w_m \times e^{\left(\log\left(\frac{1 - err_{m+1}}{err_{m+1}}\right) 1_{[y \neq f_{m+1}(x)]}\right)} = w_m \times \left(\frac{1 - err_{m+1}}{err_{m+1}}\right)^{1_{[y \neq f_{m+1}(x)]}}.$$

Thus the function and weights update are of an identical form to those used in AdaBoost. AdaBoost could do better than any single weak classifier since it iteratively minimizes the loss function via a Newton-like procedure. Interestingly, the function $F\left(x\right)$ from minimizing the exponential loss is the same as maximizing a logistic log-likelihood. Let

$$\begin{aligned} J\left(F\left(x\right)\right) &= E\left[E\left(e^{-yF(x)} | x\right)\right] \\ &= E\left[P\left(y = 1|x\right) e^{-F(x)} + P\left(y = -1|x\right) e^{F(x)}\right]. \end{aligned}$$

Taking derivative w.r.t. $F\left(x\right)$ and making it equal to zero, we obtain

$$\begin{aligned} \frac{\partial E\left(e^{-yF(x)} | x\right)}{\partial F\left(x\right)} &= -P\left(y = 1|x\right) e^{-F(x)} + P\left(y = -1|x\right) e^{F(x)} = 0 \\ F^*\left(x\right) &= \frac{1}{2} \log\left[\frac{P\left(y = 1|x\right)}{P\left(y = -1|x\right)}\right]. \end{aligned}$$

Moreover, if the true probability

$$P\left(y = 1|x\right) = \frac{e^{2F(x)}}{1 + e^{2F(x)}},$$

for $Y = \frac{y+1}{2}$, the log-likelihood is

$$E\left(\log L|x\right) = E\left[2YF\left(x\right) - \log\left(1 + e^{2F(x)}\right)|x\right].$$

The solution $F^*\left(x\right)$ that maximize the log-likelihood must equals the $F\left(x\right)$ in the true model $P\left(y=1|x\right) = \frac{e^{2F(x)}}{1+e^{2F(x)}}$. Hence,

$$\begin{aligned} e^{2F^*(x)} &= P\left(y=1|x\right)\left(1 + e^{2F^*(x)}\right) \\ e^{2F^*(x)} &= \frac{P\left(y=1|x\right)}{1 - P\left(y=1|x\right)} \\ F^*\left(x\right) &= \frac{1}{2}\log\left[\frac{P\left(y=1|x\right)}{P\left(y=-1|x\right)}\right]. \end{aligned} \tag{9}$$

AdaBoost that minimizes the exponential loss yield the same solution as logistic regression that maximizes the logistic log-likelihood.

3 Extensions to AdaBoost Algorithms

In this section we introduce extensions of Discrete AdaBoost, namely Real AdaBoost (RAB), LogitBoost (LB) and Gentle AdaBoost (GAB), and discuss how some aspects of the DAB may be modified to yield RAB, LB and GAB. In the last section, we learned that Discrete AdaBoost minimizes an exponential loss via iteratively adding a binary weaker learner to the pool of weak learners. The addition of a new weak learner can be seen as taking a step on the direction that loss function descents in the Newton method. There are two major ways to extend the idea of Discrete AdaBoost. One focuses on making the minimization method more efficient by adding a more flexible weak learner. The other is to use different loss functions that may lead to better results. Next, we give an introduction to several extensions of Discrete AdaBoost.

3.1 Real AdaBoost

Algorithm 2 Real AdaBoost (RAB, Friedman et al. 2000)

1. Start with weights $w_i = \frac{1}{n}, i = 1, \ldots, n$.
2. For $m = 1$ to M
   (a) For $j = 1$ to k (for each variable)
      i. Fit the classifier to obtain a class probability estimate $p_m(x_j) = \hat{P}_w(y = 1|x_j) \in [0, 1]$ using weights w_i on the training data.
      ii. Let $f_{mj}(x_j) = \frac{1}{2} \log \frac{p_m(x_j)}{1-p_m(x_j)}$.
      iii. Compute $err_{mj} = \sum_{i=1}^{n} w_i 1_{(y_i \neq \texttt{sign}(f_{mj}(x_{ji})))}$.
   (b) Find $\hat{j}_m = \arg\min_j err_{mj}$.
   (c) Set $w_i \leftarrow w_i \exp\left[-y_i f_{m,\hat{j}_m}(x_{\hat{j}_m,i})\right]$, $i = 1, \ldots, n$, and normalize so that $\sum_{i=1}^{n} w_i = 1$.
3. Output the classifier $\text{sign}[F_M(x)]$ and the class probability prediction $\hat{\pi}(x) = \frac{e^{F_M(x)}}{e^{F_M(x)}+e^{-F_M(x)}}$ where $F_M(x) = \sum_{m=1}^{M} f_m(x)$.

Real AdaBoost focuses solely on improving the minimization procedure of Discrete AdaBoost. In Real AdaBoost, the weak learners are continuous comparing to Discrete AdaBoost where the weak learners are binary (discrete). Real AdaBoost is minimizing the exponential loss with continuous updates where Discrete AdaBoost minimizes the exponential loss with discrete updates. Hence, Real AdaBoost is more flexible with the step size and direction of the minimization and minimizes the exponential loss faster and more accurately. However, Real AdaBoost also imposes restriction that the classifier must produces a probability prediction which reduces the flexibility of the model. As we shall see in the numerical examples, Real AdaBoost may achieve a larger in-sample training error due to the flexibility of its model. On the other hand, this also reduces the chances of fitting and would in the end achieve a smaller out-of-sample test error.

As we mentioned earlier, *ada* gives an implementation of the Real AdaBoost as well as Discrete AdaBoost.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 500)
x <- dat$X
y <- dat$y
```

```
library(ada)
model <-  ada(x, y, loss = "exponential", type = "real", iter = 200)
print(model)
```

where y and x are the training samples, and *iter* controls the number of boosting iterations. The output is as follow.

```
Call:
ada(x, y = y, loss = "exponential", type = "real", iter = 200)

Loss: exponential Method: real   Iteration: 200

Final Confusion Matrix for Data:
        Final Prediction
True value  -1   1
        -1 293  21
        1   29 157

Train Error: 0.1

Out-Of-Bag Error:  0.114  iteration= 189

Additional Estimates of number of iterations:

train.err1 train.kap1
189        189
```

3.2 LogitBoost

Friedman et al. (2000) propose LogitBoost by minimizing the Bernoulli log-likelihood via an adaptive Newton algorithm for fitting an additive logistic regression model. LogitBoost extends Discrete AdaBoost in two ways. First, it uses the Bernoulli log-likelihood instead of exponential function as loss function. Furthermore, it updates the classifier by adding a linear model instead of a binary weak learner.

Algorithm 3 LogitBoost (LB, Friedman et al., 2000)

1. Start with weights $w_i = \frac{1}{n}, i = 1, \ldots, n$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. For $m = 1$ to M
   (a) Compute the working response and weights
   $$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))} \tag{10}$$
   $$w_i = p(x_i)(1 - p(x_i)) \tag{11}$$
   (b) For $j = 1$ to k (for each variable)
      i. Fit the function $f_{mj}(x_{ji})$ by a weighted least-squares regression of z_i to $x_j i$ using weights w_i on the training data.
      ii. Compute $err_{mj} = 1 - R^2$ from the weighted least-squares regression.
   (c) Find $\hat{j}_m = \arg\min_j err_{mj}$
   (d) Update $F(x) \leftarrow F(x) + \frac{1}{2} f_{m,\hat{j}}(x_{\hat{j}})$ and $p(x) \leftarrow \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$, $i = 1, \ldots, n$.
3. Output the classifier $\text{sign}[F_M(x)]$ and the class probability prediction $\hat{\pi}(x) = \frac{e^{F_M(x)}}{e^{F_M(x)} + e^{-F_M(x)}}$ where $F_M(x) = \sum_{m=1}^{M} f_{m,\hat{j}_m}(x_{\hat{j}_m})$.

In LogitBoost, continuous weak learner are used similar to Real AdaBoost. However, LogitBoost specified the use of linear weak learner while Real AdaBoost allows any weak learner that returns a probability between zero and one. A bigger and more fundamental difference here is that LogitBoost uses the Bernoulli log-likelihood as loss function instead of the exponential loss. Hence, LogitBoost is more similar to logistic regression than Discrete AdaBoost and Real AdaBoost. As we will see in the simulation result, LogitBoost has the smallest in-sample training error but the largest out-of-sample test error. This implies that while LogitBoost is the most flexible of the four, it suffers the most from overfitting.

LogitBoost is arguably one of the most well-known boosting algorithm. Popular packages are available such as *caTools*. For consistency of the paper, here we stick with the *ada* package which gives an ideal implementation of LogitBoost algorithm for small to moderate-sized data sets.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 500)
x <- dat$X
y <- dat$y
```

```
library(ada)
model <-  ada(x, y, loss = "logistic", type = "gentle", iter = 200)
print(model)
```

where y and x are the training samples, and *iter* controls the number of boosting iterations. The output is as follow.

```
Call:
ada(x, y = y, loss = "logisitc", type = "gentle", iter = 200)

Loss: logisitc Method: gentle   Iteration: 200

Final Confusion Matrix for Data:
        Final Prediction
True value  -1   1
        -1 309   5
        1    8 178

Train Error: 0.026

Out-Of-Bag Error:  0.07  iteration= 196

Additional Estimates of number of iterations:

train.err1 train.kap1
195        195
```

3.3 Gentle AdaBoost

Algorithm 4 Gentle AdaBoost (GAB, Friedman et al. 2010)

1. Start with weights $w_i = \frac{1}{n}, i = 1, \ldots, n$.
2. For $m = 1$ to M
   (a) For $j = 1$ to k (for each variable)
      i. Fit the regression function $f_{mj}(x_{ij})$ by weighted least-squares of y_i on x_i using weights w_i on the training data.
      ii. Compute $err_{mj} = 1 - R^2$ from the weighted least-squares regression.
   (b) Find $\hat{j}_m = \arg\min_j err_{mj}$
   (c) Set $w_i \leftarrow w_i \exp[-y_i f_{m,\hat{j}_m}(x_{\hat{j}_m,i})]$, $i = 1, \ldots, n$, and normalize so that $\sum_{i=1}^{n} w_i = 1$.
3. Output the classifier $\text{sign}[F_M(x)]$ and the class probability prediction $\hat{\pi}(x) = \frac{e^{F_M(x)}}{e^{F_M(x)} + e^{-F_M(x)}}$ where $F_M(x) = \sum_{m=1}^{M} f_{m,\hat{j}_m}(x_{\hat{j}_m})$.

Gentle AdaBoost extends Discrete AdaBoost in the sense that it allows each weak learner to be a linear model. This is similar to LogitBoost and more flexible than Discrete AdaBoost and Real AdaBoost. However, it is closer to Discrete AdaBoost and Real AdaBoost than LogitBoost in the sense that Gentle AdaBoost, Discrete AdaBoost and Real AdaBoost all minimize the exponential loss while LogitBoost minimizes the Bernoulli log-likelihood. Another point that Gentle AdaBoost is more similar to Real AdaBoost than Discrete AdaBoost is that since the weak learners are continuous, there is no need to find an optimal step size for each iteration because the weak learner is already optimal. As we will see in the simulation results, Gentle Boost often lies between Real AdaBoost and LogitBoost in terms of in-sample training error and out-of-sample test error.

ada also gives an implementation of the Gentle AdaBoost algorithm.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 500)
x <- dat$X
y <- dat$y

library(ada)
model <-  ada(x, y, loss = "exponential", type = "gentle", iter = 200)
print(model)
```

where y and x are the training samples, and *iter* controls the number of boosting iterations. The output is as follow.

```
Call:
ada(x, y = y, loss = "exponential", type = "gentle", iter = 200)

Loss: exponential Method: gentle   Iteration: 200

Final Confusion Matrix for Data:
Final Prediction
True value  -1   1
-1 305   9
1   15 171

Train Error: 0.048

Out-Of-Bag Error:  0.078  iteration= 198

Additional Estimates of number of iterations:

train.err1 train.kap1
196        196
```

For all the four boosting algorithms mentioned above, *ada* outputs the class label by default. However, we can use the command *predict* to output probability prediction and/or of class label using *ada*.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 500)
x <- dat$X
y <- dat$y

library(ada)
model <-  ada(x, y, loss = "exponential", type = "discrete", iter = 200)

#New Data for Prediction
newx <- data.frame(1,1)
names(newx) <- c('V1', 'V2')
predict(model, newdata = newx, type = "F")
predict(model, newdata = newx, type = "prob")
```

where y and x are the training samples, *iter* controls the number of boosting iterations. *model* is the output from fitting the model using *ada*, *newdata* is the data to be used in prediction and *type* specifies the type of output from the *predict* function. When *type* = "*vector*", the function outputs class labels. When *type* = "*F*", the function outputs $F(x)$ which the sum of all weak learners. When *type* = "*prob*", the function outputs the class probability using 1. The output is as follow.

```
> predict(model, newdata = newx, type = "F")
1
4.345358
> predict(model, newdata = newx, type = "prob")
        [,1]      [,2]
1 0.0001681114 0.9998319
```

Note that manually transforming the sum of all weak learners $F(x)$ into probability prediction using equation 9 would lead to the same result as directly output the probability prediction from the package as in the second line.

4 Alternative Classification Methods

Apart from Boosting algorithms, we also consider Deep Neural Network, Logistic Regression and semiparametric single-index model as alternative methods to obtain a predictor of y. Deep Neural Network is able to deal with high-dimensional data. For Logistic Regression, we have to select useful information from noises. Hence, a shrinkage parameter is used with the logistic log-likelihood which we call LASSO. Semiparametric single-index model is an extension to parametric single-index model such as Logistic Regression. It relaxes the parametric assumptions and uses the kernel function of fit the data locally. For high-dimensional problem, we use SIM-RODEO to select useful explanation variables for semiparametric single-index models.

4.1 Deep Neural Network

Deep Neural Network is undoubtedly one of the most state-of-the-art classification methods. The model is similar to a multi-stage regression or classification model. The idea is to build a flexible nonlinear statistical model consisted of several layers and each layer is consisted of neurons as in Figure 1.

For binary classification, there is only one output Y that is the class probability or class label. Since the transformation from class probability to class label is straight-forward, we focus on the case where the output is the class probability. The layer labeled X is the input layer which contains all the explanatory variables in the data set. Note that the number of explanatory variables k is allowed to be extremely large (larger than the number of observations) as in high-dimensional settings. The layers labeled Z are the hidden layers. The number of hidden layers p can be arbitrarily set by the user and each hidden layer can contain arbitrarily many neurons denoted by q_t where t stands for the tth hidden layer.

The output z_{ts} of the sth neuron in the tth hidden layer is normally a single-index function $g(\alpha_{ts}+\beta_{ts}'Z_{t-1})$ where α is a scalar, β is a vector of same length q_{t-1} as the number of neurons in the $(t-1)$th hidden layer or the input layer if $t=1$ and $Z_{t-1}=(z_{t-1,1}, z_{t-1,2}, \ldots, z_{t-1,q_{t-1}})$ is a vector of outputs from all neurons of the $(t-1)$th hidden layer or the input layer if $t=1$. Similarly, the output layer of the model is also chosen to be a single-index function of

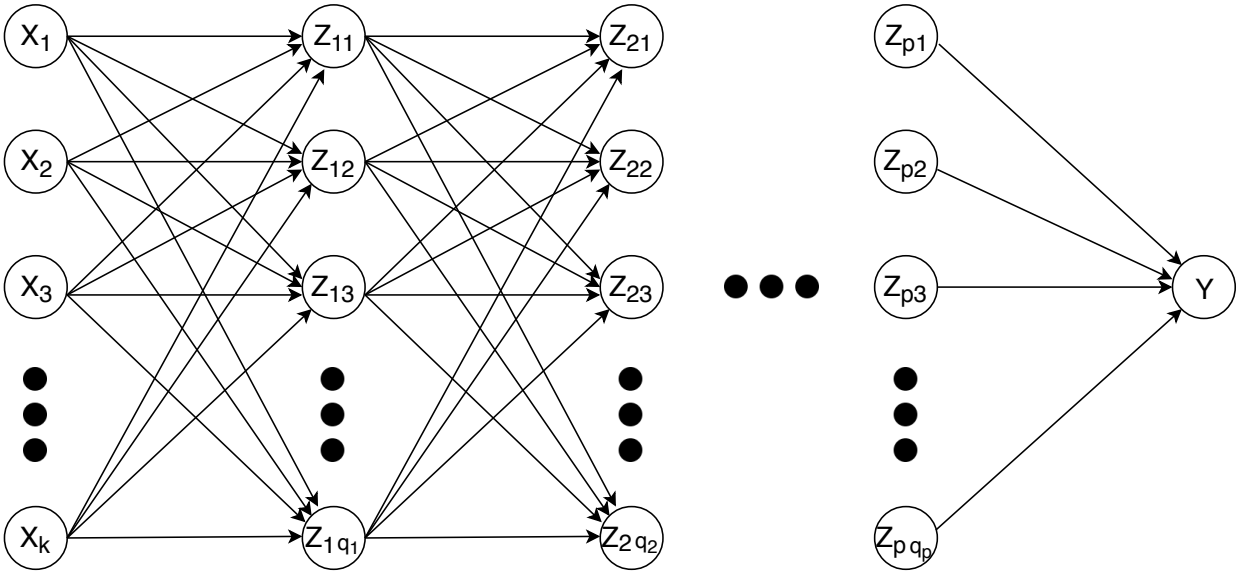

Figure 1: Diagram of Deep Neural Network

the outputs of the last hidden layer. Hence,

$$\begin{aligned} z_{1s} &= g(w_{01s} + w'_{1s}X) && (12)\\ z_{ts} &= g(w_{0ts} + w'_{ts}Z_{t-1}) && (13)\\ Y &= \hat{\pi}(x) = f(w_0 + w'Z_t). && (14) \end{aligned}$$

The function $g(v)$ is called the activation function. It is often chosen to be a sigmoid. Popular choices are the Rectified Linear Unit (ReLU)

$$g(v) = \max(0, v)$$

and the logistic function

$$g(v) = \frac{1}{1 + e^{-v}}.$$

The function $f(v)$ in the output layer can also be a sigmoid. In addition to the ReLU and logistic function, the identity function can also be used as the output function.

Since the activation function, output function, and number of hidden layers and neurons are all chosen by the user prior to fitting the model, the only parameters to be determined by the data are the weights α's and β's. We choose the best values for α's and β's to minimize a given loss function. For binary classification, the squared error loss

$$L(w) = \sum_i (y_i - \hat{\pi}(x_i))^2$$

and the cross-entropy

$$L(w) = -\sum_i y_i \log \hat{\pi}(x_i)$$

are often used. The minimization procedure of Deep Neural Network is often time-consuming. Moreover, convergence and optimality can not be guaranteed. Hence, multiple attempts need to be made for a single problem. Two techniques, stochastic gradient descent and back-propagation, are often used for minimization of Deep Neural Network. Fortunately, we do not have to worry about the implementation of the minimization procedure since packages are available in R.

Remark 3. Note that the class probability can be converted to class label easily by the rule $\hat{Y} = 1\left(\hat{\pi(x)} > 0.5\right)$ where $1(\cdot)$ is the indicator function.

We now turn to the implementation of Deep Neural Network using R. There are two packages in R for Deep Neural Networks, *neuralnet* and *keras*. *neuralnet* is a package in R that solves Deep Neural Network. *keras*, on the other hand, is an interface of `tensorflow` which we will introduce later in R. Hence, *neuralnet* is easier to use for R users and works fairly well on moderate-size problems. Let us introduce the use of *neuralnet* first.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 500)
x <- dat$X
y <- dat$y > 0
sum.data <- data.frame(x,y)

library(neuralnet)
print(net.sum <- neuralnet(y ~ X1 + X2, sum.data, hidden = 2,
act.fct = "logistic", err.fct = "sse"))
```

where y is the class label, $X1$ and $X2$ are the explanatory variables, *hidden* is a vector that specifies the number of neurons in each layer and $act.fct$ specifies the kind of activation function to be used. In our example, there is only one hidden layer with two neurons in it. The activation function is logistic and the loss function to be minimized is the sum of squared errors.

```
$call
neuralnet(formula = y ~ X1 + X2, data = sum.data, hidden = 2,
err.fct = "sse", act.fct = "logistic")

$response
y
1    TRUE
2   FALSE
3    TRUE
4   FALSE
5    TRUE
```

```
...

$covariate
[,1]           [,2]
[1,]   5.2069519311   4.2558353692
[2,]  12.6829428039 -20.3719270937
[3,]  -7.2563678008  24.6251029056
[4,]   0.8357344829 -18.1280552782
[5,]  -6.8508599121  17.9034926556
...

$model.list
$model.list$response
[1] "y"

$model.list$variables
[1] "X1" "X2"


$err.fct
function (x, y)
{
        1/2 * (y - x)^2
}
<bytecode: 0x4f10880>
<environment: 0x85c0258>
attr(,"type")
[1] "sse"

$act.fct
function (x)
{
        1/(1 + exp(-x))
}
<bytecode: 0x6b3fdf8>
<environment: 0x85c0258>
attr(,"type")
[1] "logistic"

$linear.output
[1] TRUE

$data
X1             X2     y
```

```
1     5.2069519311   4.2558353692  TRUE
2    12.6829428039 -20.3719270937 FALSE
3    -7.2563678008  24.6251029056  TRUE
4     0.8357344829 -18.1280552782 FALSE
5    -6.8508599121  17.9034926556  TRUE
...

$net.result
$net.result[[1]]
[,1]
1    0.580442461921
2    0.166226565353
3    0.746461567986
4    0.504631826509
5    0.880228917245
...

$weights
$weights[[1]]
$weights[[1]][[1]]
[,1]          [,2]
[1,]  0.9066077391 19.3160782267
[2,] -0.1145926600  1.3507916927
[3,]  0.0356239265 -0.3322296486

$weights[[1]][[2]]
[,1]
[1,] -0.8702072766
[2,]  1.0498484362
[3,]  0.8066945796

$startweights
$startweights[[1]]
$startweights[[1]][[1]]
[,1]          [,2]
[1,] -3.3233349646 -0.6039894538
[2,] -0.4675154531  0.6744466927
[3,]  0.4315402657  0.6359205358

$startweights[[1]][[2]]
[,1]
[1,] -0.6129703876
[2,]  0.4148913454
[3,]  0.8773433726
```

```
$generalized.weights
$generalized.weights[[1]]
[,1]            [,2]
1   -0.117151263217  0.036419330807
2   -0.148384521998  0.046128951957
3    0.910325193217 -0.221275093627
4   -0.119499601088  0.037149368973
5    0.071284648051 -0.011543348159
...


$result.matrix
1
error                     8.553736459613
reached.threshold         0.009821180033
steps                  2145.000000000000
Intercept.to.1layhid1     0.906607739056
X1.to.1layhid1           -0.114592659965
X2.to.1layhid1            0.035623926505
Intercept.to.1layhid2    19.316078226658
X1.to.1layhid2            1.350791692688
X2.to.1layhid2           -0.332229648639
Intercept.to.y           -0.870207276552
1layhid.1.to.y            1.049848436176
1layhid.2.to.y            0.806694579605

attr(,"class")
[1] "nn"
```

where ... represents that the rest of the output for this feature is omitted.

The *keras* package is a high-level interface of `tensorflow` which is an open source machine learning framework maintained by `Google` and is the most used library for fitting Deep Neural Networks. *keras* defines the structure and features of the neural network and send the informations to `tensorflow` which then solves the minimization problem and returns the results. *keras* is suitable for more complicated and larger problems since `tensorflow` actually doing the hard work.

Now we give a simple illustration of constructing neural networks with *keras* by constructing the same network as in the previous example in *keras*. More details about *keras* can be found at `https://tensorflow.rstudio.com`.

```
#Generate data from the circle model
library(JOUSBoost)
set.seed(111)
```

```
dat <- circle_data(n = 200)
x <- dat$X
y <- dat$y > 0
sum.data <- data.frame(x,y)

library(keras)
x_train <- x[1:100,]
y_train <- y[1:100,]
x_test <- x[101:200,]
y_test <- y[101:200,]

model <- keras_model_sequential()
model %>%
layer_dense(units = 2, activation = 'sigmoid', input_shape = c(100)) %>%
layer_dense(units = 1, activation = 'softmax')

model %>% compile(
loss = 'categorical_crossentropy',
optimizer = optimizer_rmsprop(),
metrics = c('accuracy')
)

history <- model %>% fit(
x_train, y_train,
epochs = 30, batch_size = 100,
validation_split = 0.2
)
```

4.2 Logistic Regression with LASSO

In traditional econometrics, the most used classification and probability prediction method should be logistic regression. Logistic regression assumes that the proabability that the output variable $Y = \frac{y+1}{2} \in \{0, 1\}$ takes value one follows a logistic function of x. That is

$$\pi(x) = P(Y = 1|x) = \frac{1}{1 + e^{-x\beta}}.$$

Given a sample data of y and x, the likelihood of the sample can be rewritten as

$$L(\beta) = \prod_i \left(\frac{1}{1 + e^{-x_i\beta}}\right)^{Y_i} \left(\frac{1}{1 + e^{x_i\beta}}\right)^{1-Y_i}. \tag{15}$$

Taking the log transformation, the log-likelihood is

$$\begin{aligned}
\log L(\beta) &= \log\left(\prod_i \left(\frac{1}{1+e^{-x_i\beta}}\right)^{Y_i}\left(\frac{1}{1+e^{x_i\beta}}\right)^{1-Y_i}\right) && (16)\\
&= \sum_i \log\left(\left(\frac{1}{1+e^{-x_i\beta}}\right)^{Y_i}\left(\frac{1}{1+e^{x_i\beta}}\right)^{1-Y_i}\right) && (17)\\
&= \sum_i \log\left(\frac{1}{1+e^{-x_i\beta}}\right)^{Y_i} + \log\left(\frac{1}{1+e^{x_i\beta}}\right)^{1-Y_i} && (18)\\
&= \sum_i Y_i x_i\beta - \log\left(1+e^{-x_i\beta}\right). && (19)
\end{aligned}$$

Because of the high-dimensional feature of our problem, we have to control the number of explanatory variables included in the model. Hence, an L_1 penalty a.k.a LASSO penalty is added to the log-likelihood as a penalty to including more explanatory variables in the model. Logistic regression with LASSO minimizes the negative logistic log-likelihood (4) with a Lasso penalty as below

$$\min -\sum_{t=1}^{N}\left[Y_i x_i\beta - \log\left(1+e^{x_i\beta}\right)\right] + \lambda|\beta|_1. \qquad (20)$$

A well-known package called *glmnet* package provided by Hastie and Qian uses a quadratic approximation to the log-likelihood, and then coordinate descent on the resulting penalized weighted least-squares problem. And it is so far the most trust-worthy package in `R` for logistic regression with LASSO. For binary classification, we use the estimated β to construct a logistic probability model for y. Then, get our prediction from the model. If $\hat{\pi}(x) > 0.5$, the predicted class will be 1. And if $\hat{\pi}(x) < 0.5$, the predicted class will be 0.

We can use the following command for logistic regression.

```
library(JOUSBoost)
set.seed(111)
dat <- circle_data(n = 100)
x <- dat$X
y <- dat$y > 0
sum.data <- data.frame(x,y)

library(glmnet)
model <- cv.glmnet(y,x, family = "binomial")
y.fit <- predict(model, newx = x, s = "lambda.1se", type = "response") > 0.5
train.error <- print(sum(y.fit != y) / n)
```

The output is the in-sample training error rate.

```
[1] 0.086
```

If we are interested in knowing the estimated coefficients of the model, we can check the components in the *model* object.

```
# The value of lambda's as shown in the objective function
print(model$glmnet.fit$lambda)
# The estimated beta's using the corresponding lambda
# shown by the previous command
print(model$glmnet.fit$beta)
```

The output looks like this.

```
[1] 0.029100660 0.026515438 0.024159880 0.022013582 0.020057956 0.018276063 0.016652468
[8] 0.015173109 0.013825171 0.012596981 0.011477900 0.010458235 0.009529154 0.008682611
[15] 0.007911271 0.007208456 0.006568076 0.005984587 0.005452932 0.004968509 0.004527120
[22] 0.004124943 0.003758495 0.003424601 0.003120368 0.002843164 0.002590585
2 x 27 sparse Matrix of class "dgCMatrix"
[[ suppressing 27 column names ‘s0’, ‘s1’, ‘s2’ ... ]]

V1 . -0.0006865767 -0.001312223 -0.001882394 -0.00240204 -0.0028939228
  -0.0033458069 -0.003757714
V2 .  .             .            .            .           0.0003774458
  0.0007948127  0.001175215

V1 -0.004133202 -0.004475500 -0.004787549 -0.005072024 -0.005331360 -0.005567777
  -0.005783296
V2  0.001521948  0.001838004  0.002126106  0.002388728  0.002628126  0.002846351
  0.003045275

V1 -0.005979761 -0.006158854 -0.006322106 -0.006470914 -0.006606555 -0.006730189
  -0.006842877
V2  0.003226601  0.003391885  0.003542542  0.003679864  0.003805030  0.003919112
  0.004023090

V1 -0.006945586 -0.007039197 -0.007124515 -0.007202272 -0.007273138
V2  0.004117857  0.004204228  0.004282945  0.004354685  0.004420064
```

Note that I reformat the output to fit the size of the paper.

4.3 Semiparametric Single-Index Model

Chu et al. (2018b) consider a standard single index model,

$$y = m\left(x'\beta\right) + u, \tag{21}$$

where $\beta = (\beta_1, \ldots, \beta_k)$ is a vector of coefficients. Under the sparsity condition, we assume that $\beta_j \neq 0$ for $j \leq r$ and $\beta_j = 0$ for $j > r$. We also assume that the random errors u

are independent. However, we allow the presence of heteroskedasticity to encompass a large category of models for binary prediction, e.g. Logit and Probit models. The kernel estimator (Ichimura 1993) we use is as shown below

$$\hat{m}\left(x'\beta; h\right) = \frac{\sum_{i=1}^{n} y_i K\left(\frac{X_i'\beta - x'\beta}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{X_i'\beta - x'\beta}{h}\right)}, \tag{22}$$

where $K(\cdot)$ is a kernel function. The semiparametric kernel regression looks for the best β and h to minimize a weighted squared error loss. However, exact identification is not available. If one blows up β and θ simultaneously by multiplying the same constant, the kernel estimator would yield identical estimates and losses. The standard identification approach is to set the first element of β to be 1 (Ichimura 1993).

In terms of variable selection and prediction, we only need to focus on finding the best $\theta \equiv \frac{\beta}{h}$. Hence, we can simplify the estimator to

$$\hat{m}\left(x'\theta\right) = \frac{\sum_{i=1}^{n} y_i K\left(X_i'\theta - x'\theta\right)}{\sum_{i=1}^{n} K\left(X_i'\theta - x'\theta\right)}. \tag{23}$$

The basic idea of the SIM-Rodeo is to view the local bandwidth selection as a variable selection in sparse semiparametric single index model. The SIM-Rodeo algorithm amplifies the inverse of the bandwidths for relevant variables while keeping the inverse of the bandwidths of irrelevant variables relatively small. The SIM-Rodeo algorithm is greedy as it solves for the locally optimal path choice at each iteration. It can also be shown to attain the consistency in mean square error when it is applied for sparse semiparametric single index models. SIM-Rodeo is able to distinguish truly relevant explanatory variables from noisy irrelevant variables and gives a consistent estimator of the regression function. In addition, the algorithm is fast to finish the greedy steps.

Now we derive the Rodeo for Single Index Models. First we introduce some notation. Let

$$W_x = \begin{pmatrix} K\left(X_1'\theta - x'\theta\right) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & K\left(X_n'\theta - x'\theta\right) \end{pmatrix} \tag{24}$$

where $K(\cdot)$ is the Gaussian kernel. The standard Ichimura (1993) estimator takes the form

$$\hat{m}\left(x'\theta\right) = \frac{\sum_{i=1}^{n} y_i K\left(X_i'\theta - x'\theta\right)}{\sum_{i=1}^{n} K\left(X_i'\theta - x'\theta\right)} = \left(\iota' W_x \iota\right)^{-1} \iota' W_x y. \tag{25}$$

The derivative of the estimator Z_j with respect to θ_j is

$$\begin{aligned} Z_j &\equiv \frac{\partial \hat{m}\left(x'\theta\right)}{\partial \theta_j} \quad (26) \\ &= \left(\iota' W_x \iota\right)^{-1} \iota' \frac{\partial W_x}{\partial \theta_j} y - \left(\iota' W_x \iota\right)^{-1} \iota' \frac{\partial W_x}{\partial \theta_j} \iota \left(\iota' W_x \iota\right)^{-1} \iota' W_x y \\ &= \left(\iota' W_x \iota\right)^{-1} \iota' \frac{\partial W_x}{\partial \theta_j} \left(y - \iota \hat{m}\left(x'\theta\right)\right). \quad (27) \end{aligned}$$

For the ease of computation, let

$$L_j = \begin{pmatrix} \frac{\partial \log K\left(X_1'\theta - x'\theta\right)}{\partial \theta_j} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial \log K\left(X_n'\theta - x'\theta\right)}{\partial \theta_j} \end{pmatrix}. \quad (28)$$

Note that

$$\frac{\partial W_x}{\partial \theta_j} = W_x L_j, \quad (29)$$

which appears in equation (27). With the Gaussian kernel, $K\left(t\right) = e^{-\frac{t^2}{2}}$, then L_j becomes

$$\begin{aligned} L_j &= \begin{pmatrix} -\frac{1}{2}\frac{\partial \left(X_1'\theta - x'\theta\right)^2}{\partial \theta_j} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -\frac{1}{2}\frac{\partial \left(X_n'\theta - x'\theta\right)^2}{\partial \theta_j} \end{pmatrix} \\ &= \begin{pmatrix} -\left(X_1'\theta - x'\theta\right)\left(X_{1j} - x_j\right) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & -\left(X_n'\theta - x'\theta\right)\left(X_{nj} - x_j\right) \end{pmatrix}, \end{aligned}$$

where X_{1j} and X_{nj} are the jth elements of vectors X_1 and X_n. And x_j is the jth element of vector x. To simplify the notation, let $B_x = \left(\iota' W_x \iota\right)^{-1} \iota' W_x$. Then, the derivative Z_j becomes

$$\begin{aligned} Z_j &= \left(\iota' W_x \iota\right)^{-1} \iota' \frac{\partial W_x}{\partial \theta_j} \left(y - \iota \hat{m}\left(x'\theta\right)\right) \\ &= B_x L_j \left(I - \iota B_x\right) y \\ &\equiv G_j\left(x, \theta\right) y. \quad (30) \end{aligned}$$

Next, we give the conditional expectation and variance of Z_j.

$$\begin{aligned} Z_j &= G_j\left(x,\theta\right) y = G_j\left(x,\theta\right)\left(m\left(x'\beta\right) + u\right), \quad (31) \\ E\left(Z_j | X\right) &= E\left(G_j\left(x,\theta\right)\left(m\left(x'\beta\right) + u\right) | X\right) = G_j\left(x,\theta\right) m\left(x'\beta\right), \quad (32) \\ \mathrm{Var}\left(Z_j | X\right) &= \mathrm{Var}\left(G_j\left(x,\theta\right)\left(m\left(x'\beta\right) + u\right) | X\right) = \boldsymbol{\sigma}' G_j\left(x,\theta\right)' G_j\left(x,\theta\right) \boldsymbol{\sigma}, \quad (33) \end{aligned}$$

where $\boldsymbol{\sigma} = (\sigma(u_1), \ldots, \sigma(u_n))'$ is the vector of standard deviations of u. In the algorithm, it is necessary to insert an estimate of σ. Since we allow the errors to be heteroskedastic as in Logit and Probit models and estimate $\sigma(u_i)$ using the estimator $\hat{\sigma}(u_i) = m(x_i'\hat{\theta})(1 - m(x_i'\hat{\theta}))$.

SIM-Rodeo is described in Algorithm 5, which is a modified algorithm of Rodeo (Lafferty and Wasserman 2008).

Algorithm 5 SIM-Rodeo (Chu et al., 2018b)

1. Select a constant $0 < \alpha < 1$ and the initial value

$$\theta_0 = c_0 \log\log n$$

   where c_0 is sufficiently small. Compute Z_j with $\theta_j = \theta_0$ for all j.

2. Initialize the coefficients θ, and activate all covariates:

   (a) $\theta_j = \begin{cases} \theta_0 & Z_j > 0 \\ -\theta_0 & \text{otherwise,} \end{cases} \quad j = 1, \ldots, k.$

   (b) $\mathcal{A} = \{1, \ldots, k\}$.

3. While $\mathcal{A} \neq \emptyset$ is nonempty, do for each $j \in \mathcal{A}$:

   (a) Compute Z_j and $s_j = \sqrt{\text{Var}(Z_j|X)}$ using (30) and (33) respectively.

   (b) Compute the threshold $\lambda_j = s_j\sqrt{2\log n}$.

   (c) If $|Z_j| > \lambda_j$, then set $\theta_j \leftarrow \frac{\theta_j}{\alpha}$; Otherwise, remove j from $\mathcal{A}$ (i.e., $\mathcal{A} \leftarrow \mathcal{A} - \{j\}$).

4. Obtain $\hat{\theta} = (\theta_1, \ldots, \theta_k)$. Output the class probability prediction $\hat{\pi}(x) = \hat{m}\left(x'\hat{\theta}\right)$ and the classifier $F(x) = 1_{(\hat{\pi}(x) > 0.5)}$.

We start by setting $\theta_j = \theta_0$ that is close to zero. Hence, $(X_i'\theta - x'\theta)$ are close to zero and $K(X_i'\theta - x'\theta)$ are close to $K(0)$. This means our estimator starts with the simple average of all observations, $\bar{y}$. If the derivative of θ_j is statistically different from zero. We amplify θ_j. If x_j is indeed a relevant explanatory variable, then the weights $K(X_i'\theta - x'\theta)$ change according to x_j. The estimator will give higher weights to observations close to $x'\theta$ and lower weights to observations away from $x'\theta$.

5 Monte Carlo

In this section, we demonstrate the above DAB, RAD, LB and GB via small Monte Carlo simulation designs to illustrate `R` functions and library.

We construct the two DGPs to check the finite sample properties of the Boosting algorithms. DGP1 is a binary logistic model where y follows a Bernoulli distribution with

probability

$$\pi(x) \equiv \frac{1}{1+e^{-x\beta}}$$

to be 1 and $1-\pi(x)$ to be -1 where

$$\underset{n\times k}{x} \sim N\left(0, \frac{\Sigma}{\beta'\Sigma\beta}\right),\ \Sigma_{ij} = \rho^{|i-j|},$$

$$n = 100, k = \{2, 20\} \text{ and } \rho \in \{0\}.$$

We have two settings for the β. In the low-dimension case ($k = 2$) , we let

$$\beta = (1, 1).$$

In the high-dimension case ($k = 20$), we let $\beta = (\beta_1, \ldots, \beta_k)$ where

$$\beta_i = 0.9^i. \tag{34}$$

that decrease exponentially. Hence, most of the β's are very close 0.

DGP2 is the circle model introduced in Section 2. Here we have two settings for the circle model. In the low dimension case, only the two relevant x's are used to train the models as shown in the toy demo in previous sections. In the high dimension (sparse) case, three irrelevant x's are added in addition to the two relevant ones. Table 1 and Table 2 show the in-sample training error and the out-of-sample test error in the two cases for different methods.

To construct the training and testing samples, we randomly generate x using the above distribution and calculate $\pi(x)$. To generate the random variable y based on x, we first generate a random variable ϵ that follows uniform distribution between $[0, 1]$. Next, we compare ϵ with $\pi(x)$. There is a probability of $\pi(x)$ that ϵ is smaller than $\pi(x)$ and a probability $1-\pi(x)$ otherwise. Hence, we set

$$y = \begin{cases} 1 & \epsilon < \pi(x) \\ -1 & \epsilon > \pi(x). \end{cases}$$

Given a set of observations $\{(x, y)\}$, we compare the average loss (classification error) achieved by using different methods. The formula for the average loss is as below.

$$ErrorRate = \frac{1}{n}\sum 1\left(y_i \neq \texttt{sign}\left(F_M(x_i)\right)\right), \tag{35}$$

where n is the number of observations in the set.

To evaluate the algorithms, first we train our predictors with the training data of size $n = 100$. Then, we use a testing data set that contains 100 new observations of (x, y) to compute the average loss (35) achieved by the Boosting algorithms, Deep Neural Network, Logistic Regression and semiparametric Single-Index Model for out-of-sample evaluations. The boosting algorithms are component-wise versions of the four methods as shown before.

The alternative methods we have, Deep Neural Network, Logistic Regression with LASSO penalty and semiparametric single-index model with SIM-RODEO considers all variables at the same time. The number of Monte Carlo repetition for each DGP is 1000.

The results are shown below.

Table 1: Error Rate of Low Dimension Circle Model

	Train Error	Test Error
Discrete AdaBoost	0.0820	0.2053
Real AdaBoost	0.0853	0.2038
LogitBoost	0.0602	0.2090
Gentle AdaBoost	0.0718	0.2062
Deep Neural Network	0.2601	0.3533
Logistic Regression	0.3586	0.3573
SIM-RODEO	0.2986	0.3421

Table 2: Error Rate of High Dimension (Sparse) Circle Model

	Train Error	Test Error
Discrete AdaBoost	0.0202	0.2203
Real AdaBoost	0.0295	0.2165
LogitBoost	0.0081	0.2232
Gentle AdaBoost	0.0133	0.2208
Deep Neural Network	0.2838	0.4017
Logistic Regression	0.3569	0.3572
SIM-RODEO	0.3542	0.3541

Table 3: Error Rate of Low Dimension Logistic Model

	Train Error	Test Error
Discrete AdaBoost	0.1431	0.3129
Real AdaBoost	0.1519	0.3120
LogitBoost	0.1302	0.3160
Gentle AdaBoost	0.1339	0.3154
Deep Neural Network	0.2304	0.3090
Logistic Regression	0.2773	0.3083
SIM-RODEO	0.3069	0.3415

From the simulation results, we can see that the four boosting methods work well in both the circle model and the logistic model. LogitBoost has the smallest training error among

Table 4: Error Rate of High Dimension (Sparse) Logistic Model

	Train Error	Test Error
Discrete AdaBoost	0.0007	0.3217
Real AdaBoost	0.0015	0.3215
LogitBoost	0.00007	0.3214
Gentle AdaBoost	0.0001	0.3204
Deep Neural Network	0.0523	0.3172
Logistic Regression	0.2328	0.3432
SIM-RODEO	0.3580	0.3971

all four boosting algorithms as well as the largest testing error. On the other hand, Real AdaBoost has the largest training error as well as the smallest testing error. Similar rules applies to the other two boosting methods. Smaller training errors implies larger testing errors. This is an evidence of overfitting which is related to the hyper-parameters in the boosting algorithms. If the number of boosting iterations is small, then we will have a larger training error but less risk of overfitting. On the other hand, if we have more boosting iterations, then the boosting methods will fit the training data better but raise higher risk on overfitting. The number of iterations in the boosting algorithms are fixed by the users. However, cross-validation could be used to determine the optimal number of iterations.

As for the alternative methods, Deep Neural Network works better in the logistic model than the circle model. This is a result of the set-up of the Deep Neural Network. We use the logistic function as the activation function and output function, and the entropy as the loss function. The set-up will give better results when logistic model is the true model. For the circle model, Deep Neural Network gives a comparable result to the Logistic Regression in the low-dimension case. However, the result is much worse for the high-dimension case. Again, this could be a result of our set-up of the Deep Neural Network. We acknowledge that the Deep Neural Network is high flexible with lots of hyper-parameters Different set-up of the model may lead to dramatically distinct results. Our setting by no means is the optimal one and Deep Neural Network could perform better with a different set-up.

For Logistic Regression, it works best in the low-dimension logistic model as all parametric assumptions are satisfied. However, in the high-dimension case, Logistic Regression will have a larger bias due to the need to shrink the coefficients of irrelevant variables to zero. To fix this bias, one may try the De-biased Machine Learning method(Chernozhukov et al., 2018).

6 Applications

In this section we illustrate the `R` functions in two economics applications.

In the application, we use the FRED monthly data `https://research.stlouisfed.org/econ/mccracken/fred-databases/` to predict the moving direction of real personal income in the United States. After removing the observations with missing values, our obtain 341 effective observation with a sample period starting from September, 1989 to January

2018. We use 125 variables which are all variables in the data except for the Consumer Sentiment Index that is only available quarterly and New Orders for Consumer Goods which has too many missing data. We generate the direction of the real personal income as our dependent variable and take the lag of the dependent variable as one explanatory variable. Hence, we have in total $k = 126$ explanatory variables and $(341 - 1 = 340)$ observations. We use rolling training sample with window width $W = 100$ and predict the one month ahead moving direction. We have $(n = 340 - W = 240)$ subsamples and predictions.

Table 5: Error Rate of Application

	Train Error	Test Error
Discrete AdaBoost	0.0028	0.3125
Real AdaBoost	0.0407	0.3291
LogitBoost	0.0003	0.3041
Gentle AdaBoost	0.0020	0.3083
Deep Neural Network	0.2389	0.2666
Logistic Regression	0.2479	0.2708
SIM-RODEO	0.2257	0.2958

The results are very similar to the simulation results for logistic models. The boosting methods have very small in-sample training errors. However, the out-of-sample testing error are much larger than the alternatives. This may indicate that the boosting algorithms are overfitting the model.

7 Conclusions

This paper shows recent developed methods for high-dimensional binary classification and probability prediction. We start by introducing four component-wise boosting methods, namely component-wise Discrete AdaBoost, component-wise Real AdaBoost, component-wise LogitBoost and component-wise Gentle AdaBoost. Discrete AdaBoost, Real AdaBoost and Gentle AdaBoost minimizes the exponential loss via Newton-like procedures. LogitBoost minimizes the Bernoulli log-likelihood via adaptive Newton method. These methods are extremely popular since they are both computationally efficient and easy to implement. Moreover, the component-wise Boosting algorithms deal with high dimensional issue by considering the explanatory one at a time. In each iteration, only the most effective explanatory variable is chosen to train a weak learner. Hence, these methods allows $k \gg n$. However, hyper-parameters such as the number of boosting iteration normally need to be determined by the user prior to the estimation procedure. Cross-validation may also be used to choose the number of iterations.

Next, we give an introduction to alternative methods such as Deep Neural Network, Logistic Regression and SIM-RODEO. Deep Neural Network is a kind of nonlinear statistical learning model features a network structure that is similar to the relationship between the neurons of human brain. Deep Neural Network may be explained partly as a kind of basis

transformation which leads to extreme flexibility of the model. Deep Neural Network and its variants are the most popular prediction method at this time and are widely used in fields such as image and voice recognition.

Logistic Regression is a traditional method used intensively in economics for binary classification and probability prediction. Logistic Regression assumes that the probability that the output label is 1 conditional on x follows a logistic function of x. Under such assumption, the parameters of the model often has practical economic meaning unlike machine learning methods that are often hard to interpret. However, logistic regression relies heavily on its parametric assumptions and is the least flexible model introduced in this paper. In addition, to deal with high-dimensional problem, we have to use the LASSO to control the number of explanatory variables chosen in the model.

SIM-RODEO relaxes the parametric assumption of Logistic Regression. As a result, SIM-RODEO is more flexible but, to some extent, still interpretable as Logistic Regression. However, the flexibility of SIM-RODEO may lead to a slower convergence rate and less time efficiency.

This paper conducted extensive comparison of the above mentioned methods through Monte Carlo experiments. We compare the methods using both traditional binary classification model (logistic model) and irregular model (circle model). The boosting methods work well in both the traditional models and irregular models. Logistic Regression works better in the low dimension logistic model when the parametric assumptions of Logistic Regression are satisfied. However, in the high-dimensional case, the LASSO introduces high bias in Logistic Regression and lead to lower classification accuracy. In the irregular models, Logistic Regression performs poor compared to the boosting algorithms. The Deep Neural Network performed best in the traditional methods as a result of our configuration of the Neural Network. We acknowledge that our configuration of Deep Neural Network is by no means the best and the results here may improve with different activation function, output function and/or number of hidden layers and neurons. SIM-RODEO is an extension to parametric methods such as Logistic Regression. It performs reasonably well in the models.

We also use these methods for predicting the changing direction of the real personal income in the United States. The application show similar results as in the simulation of logistic models.

This paper gives a thorough introduction of newly developed methods for binary classification and probability prediction. Advantages and disadvantages of each method are discussed and compared. We conclude that no single method has an absolute advantage in all aspects over the other methods. We believe binary classification and probability prediction will remain important for business and economics and look forward to future works on this problem.

References

Allaire, J., Chollet, F., 2018. keras: R Interface to 'Keras'. URL: `https://CRAN.R-project.org/package=keras`. r package version 2.1.6.

Bliss, C.I., 1934. The method of probits. Science 79, 38–39. URL: http://www.ncbi.nlm.nih.gov/pubmed/17813446, doi:10.1126/science.79.2037.38.

Bühlmann, P., 2006. Boosting for high-dimensional linear models. The Annals of Statistics 34, 559–583. doi:10.1214/009053606000000092.

Bühlmann, P., Yu, B., 2003. Boosting with the L_2 loss: Regression and classification. Journal of the American Statistical Association 98, 324–339. URL: http://www.tandfonline.com/doi/abs/10.1198/016214503000125, doi:10.1198/016214503000125.

Chatterjee, S., 2016. fastAdaboost: a Fast Implementation of Adaboost. URL: https://CRAN.R-project.org/package=fastAdaboost. r package version 1.0.0.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., Robins, J., 2018. Double/debiased machine learning for treatment and structural parameters. Econometrics Journal 21, C1–C68. URL: http://arxiv.org/abs/1608.00060, doi:10.1111/ectj.12097, arXiv:1608.00060.

Chu, J., Lee, T.H., Ullah, A., 2018a. Asymmetric AdaBoost for High-Dimensional Maximum Score Regression.

Chu, J., Lee, T.H., Ullah, A., 2018b. Variable Selection in Sparse Semiparametric Single Index Model.

Cox, D.R., 1958. The regression analysis of binary sequences. Journal of the Royal Statistical Society. Series B (Methodological) 20, 215–242. URL: http://www.jstor.org/stable/2983890.

Culp, M., Johnson, K., Michailidis, G., 2016. ada: The R Package Ada for Stochastic Boosting. URL: https://CRAN.R-project.org/package=ada. r package version 2.0-5.

Elliott, G., Lieli, R.P., 2013. Predicting binary outcomes. Journal of Econometrics 174, 15–26. URL: https://www.sciencedirect.com/science/article/pii/S0304407613000171, doi:10.1016/j.jeconom.2013.01.003.

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

Friedman, J., Hastie, T., Tibshirani, R., 2000. Additive logistic regression: A statistical view of boosting. Annals of Statistics 28, 337–407. doi:10.1214/aos/1016218223, arXiv:0804.2330.

Friedman, J., Hastie, T., Tibshirani, R., 2010. Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software 33, 1–22. URL: http://www.jstatsoft.org/v33/i01/, doi:10.18637/jss.v033.i01, arXiv:NIHMS150003.

Friedman, J.H., 2001. Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics 29, 1189–1232. doi:10.2307/2699986.

Fritsch, S., Guenther, F., 2016. neuralnet: Training of Neural Networks. URL: https://CRAN.R-project.org/package=neuralnet. r package version 1.33.

James, G., Witten, D., Hastie, T., Tibshirani, R., 2013. An Introduction to Statistical Learning with Applications in R. Springer.

Lafferty, J., Wasserman, L., 2008. Rodeo: Sparse, greedy nonparametric regression. Annals of Statistics 36, 28–63. doi:10.1214/009053607000000811, arXiv:0803.1709.

Lahiri, K., Yang, L., 2012. Forecasting binary outcomes, in: Elliott, G., Timmermann, A. (Eds.), Handbook of Economic Forecasting. SSRN. volume 2, pp. 1025–1106. doi:10.1016/B978-0-444-62731-5.00019-1.

Manski, C.F., 1975. Maximum score estimation of the stochastic utility model of choice. Journal of Econometrics 3, 205–228. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.587.6474{&}rep=rep1{&}type=pdf, doi:10.1016/0304-4076(75)90032-9.

Manski, C.F., 1985. Semiparametric analysis of discrete response. Asymptotic properties of the maximum score estimator. Journal of Econometrics 27, 313–333. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.504.7329{&}rep=rep1{&}type=pdf, doi:10.1016/0304-4076(85)90009-0.

Mease, D., Wyner, A., Buja, A., 2007. Cost-weighted boosting with jittering and over/under-sampling: Jous-boost. Journal of Machine Learning Research 8, 409–439.

Olson, M., 2017. JOUSBoost: Implements Under/Oversampling for Probability Estimation. URL: https://CRAN.R-project.org/package=JOUSBoost. r package version 2.1.0.

R Core Team, 2018. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Ridgeway, G., 2017. gbm: Generalized Boosted Regression Models. URL: https://CRAN.R-project.org/package=gbm. r package version 2.1.3.

Su, L., Zhang, Y., 2014. Variable Selection in Nonparametric and Semiparametric Regression Models, in: Racine, J.S., Su, L., Ullah, A. (Eds.), The Oxford Handbook of Applied Nonparametric and Semiparametric Econometrics and Statistics. Oxford University Press. URL: http://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199857944.001.0001/oxfordhb-9780199857944-e-009, doi:10.1093/oxfordhb/9780199857944.013.009.

Tibshirani, R., 1996. Regression Selection and Shrinkage via the Lasso. Journal of the Royal Statistical Society B 58, 267–288. URL: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.7574, doi:10.2307/2346178, arXiv:11/73273.

Tuszynski, J., 2018. caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc. URL: https://CRAN.R-project.org/package=caTools. r package version 1.17.1.1.

Walker, S.H., Duncan, D.B., 1967. Estimation of the probability of an event as a function of several independent variables. Biometrika 54, 167–179. doi:10.1093/biomet/54.1-2.167.

Zou, H., 2006. The Adaptive Lasso and Its Oracle Properties. Journal of the American Statistical Association 101, 1418–1429. URL: http://users.stat.umn.edu/{~}zouxx019/Papers/adalasso.pdf, doi:10.1198/016214506000000735.