**Machine Learning - a probabilistic approach**
Report

# Gradient Boosting

Written by

Paulina Kurowska

**Abstract**

The purpose of this report is to present Boosting as a Machine Learning method with the focus on the statistical perspective. Introduced algorithms were tested and analyzed also on a datasets, which are widely available in R. In the centre stays gradient boosting, nevertheless other methods like AdaBoost, L2Boosting, LogitBoost are discussed as well in larger or smaller extent.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

### 1.1.1 Problem definition

Boosting can be an answer for a non trivial problem, where the classes in classification problem cannot be separated by a straight line without any errors. It can be also applied to regression problem, but for the introduction purposes we will focus now on classification. As an example, consider a two-category problem as in the first picture in Figure 1. Creating three different classifier, we can see that each of them deliver a pretty high error rate. The idea is to combine them together into one classifier and to build an ensemble[1] model called boosting.
Those component classifiers C1, C2, C3 are characterized by the fact, that they are a little bit better than random guessing, which means, that the error rate is about 0.5, given the scale from 0 for no errors and 1 when everything is wrong. Having that in mind, in order to find C1 first we randomly select a set of n1 patterns from the full training set D (without replacement), call this set D1. Then we train the first classifier, C1, with D1. Now we seek a second training set, D2, that is the "most informative" given component classifier C1. Specifically, half of the patterns in D2 should be correctly classified by C1, half incorrectly classified by C1. Such an informative set D2 is created as follows: we flip a fair coin. If the coin is heads, we select remaining samples from D and present them, one by one to C1 until C1 misclassifies a pattern. We add this misclassified pattern to D2. Next we flip the coin again. If heads, we continue through D to find another pattern misclassified by C1 and add it to D2 as just described; if tails we find a pattern which C1

[1] this concept is explained in the next section

classifies correctly. We continue until no more patterns can be added in this manner. Thus half of the patterns in D2 are correctly classified by C1, half are not. As such D2 provides information complementary to that represented in C1. Now we train a second component classifier C2 with D2 [?].
Next we seek a third data set, D3, which is not well classified by the combined system C1 and C2. We randomly select a training pattern from those remaining in D, and classify that pattern with C1 and with C2. If C1 and C2 disagree, we add this pattern to the third training set D3; otherwise we ignore the pattern. We continue adding informative patterns to D3 in this way. Thus D3 contains those not well represented by the combined decisions of C1 and C2. Finally, we train the last component classifier, C3, with the patterns in D3 [?].
Now consider the use of the ensemble of three trained component classifiers for classifying a test pattern x. Classification is based on the votes of the component classifiers. Specifically, if C1 and C2 agree on the category label of x, we use that label; if they disagree, then we use the label given by C3 [?].

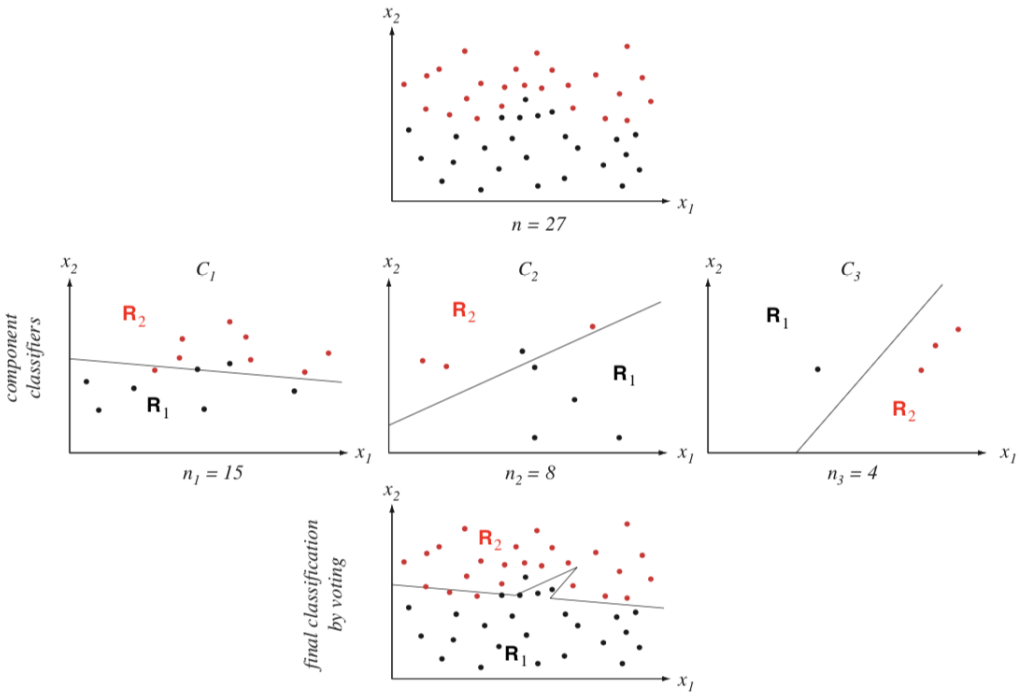



Figure 1.1: Boosting as a classification problem - introductory example [?].

## 1.2 Motivation

Boosting aims to reduce the bias and also a variance. It was primarily discovered in computational learning theory by Freund and Shapire (1990 and 1996), who showed that the performance of a weak learner can be always slightly improved. In 1998 Breimann used shallow decision tree as a weak learner and called it to be "best off-the-shelf classifier in the world", later in 2006 this was confirmed by the Caruana and Niculescu-Mizil, who claimed that boosted decision trees are the best in terms of misclassification error and creation of well-calibrated probabilities. It has become very popular due to its remarkable empirical success and the fact, that it is resistant to overfitting. The statisticians looked deeper into that method, Breimann proved that boosting can be understood as a gradient descent in function space and Friedman extended boosting to variety of loss functions for regression, robust regression, poisson regression[**?**].

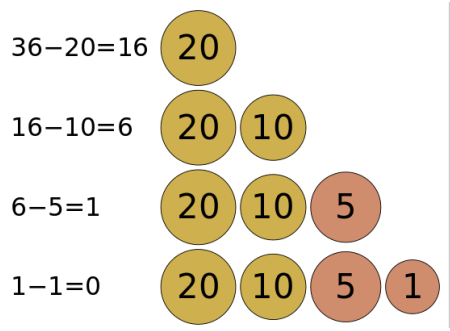
## 1.3 Definition

According to Schapire and Freund (2012) boosting is a greedy algorithm for fitting adaptive basis-function (ABM) of the form:

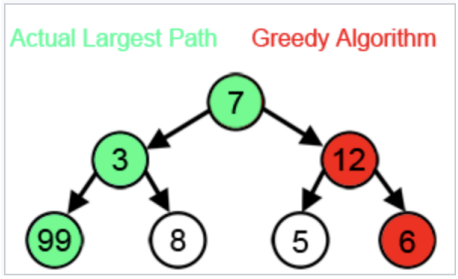
$$f(y|x,\pi) = w_0 + \sum_{m \in M} w_m f_m(y|x),$$

where $\phi_m(x)$ is the m'th basis function, which is learned from the data. $\phi_m$ are generated by an algorithm called a **weak learner** or **base learner**. The algorithm works by applying the weak learner sequentially to weighted versions of the data, where more weight is given to examples that were misclassified by earlier rounds [**?**]. If the weak learner can predict with high accuracy on a training set, then the benefit of boosting is not big[**?**].
A greediness of an algorithm consists in making local optimal choice at each iteration or stage in order to find global optimum. In the Figure 2 (a) we can observe the implementation of a greedy algorithm to a task of giving a minimum number of coins in making change. It illustrates how a human would apply it for making change of 36 cents using only coins with values 1, 5, 10, 20. It is said, that greedy is able to reach global optimum in a reasonable amount of time. However, sometimes it may fail. On the right hand-side is the case, where following local optimum will not reach the global one. Here the goal is to find the largest sum, greedy algorithm will pursue to the six

through 12 as the bigger sum between 3 and 12. After that the selection will lead to 6, whereas the global maximum sum is actually 99 [?].




(a) Application of greedy algorithm in making change of 36 cents using only coins with values 1, 5, 10, 20




(b) Local vs. global optimum in greedy algorithm

Figure 1.2: Greedy algorithm[?].

We can also think of boosting as kind of ensemble learning, where the weights on the base models are determined sequentially:

$$f(x) = w_0 + \sum_{m=1}^{M} w_m \phi_m(x),$$

where the $w_m$ are tunable parameters. Ensemble learning is sometimes called a committee method, since each base model $f_m$ gets a weighted "vote" [?].

### 1.3.1 Goal

The goal is to solve the following optimization problem:

$$\operatorname*{argmin}_f \sum_{i=1}^{N} L(y_i, f(x_i))$$

where $L(y, \hat{y})$ is some loss function, f is assumed to be an ABM model [?].

Selecting the loss function is dependent on the problem, for regression it is recommended to use squared error or absolute error, whereas for classification exponential or log loss. Gradient boosting can be used for both regression or classification problems. A table in Figure 1.3 shows also what

| Name | Loss | Derivative | $f^*$ | Algorithm |
|---|---|---|---|---|
| Squared error | $\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$ | $y_i - f(\mathbf{x}_i)$ | $\mathbb{E}[y\|\mathbf{x}_i]$ | L2Boosting |
| Absolute error | $\|y_i - f(\mathbf{x}_i)\|$ | $\text{sgn}(y_i - f(\mathbf{x}_i))$ | $\text{median}(y\|\mathbf{x}_i)$ | Gradient boosting |
| Exponential loss | $\exp(-\tilde{y}_i f(\mathbf{x}_i))$ | $-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$ | $\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$ | AdaBoost |
| Logloss | $\log(1 + e^{-\tilde{y}_i f_i})$ | $y_i - \pi_i$ | $\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$ | LogitBoost |

Figure 1.3: List of loss functions used in boosting[**?**].

loss function is used in each type of boosting algorithm, as well as the corresponding derivative of a function and population minimizer (optimal $f*$). For binary classification problems, we assume $\tilde{y} \in \{-1, +1\}, y_i \in \{0, 1\}$and $\pi_i = sigm(2f(xi))$. For regression problems, we assume $y_i \in \mathbb{R}$ [**?**].

### 1.3.2 Early stopping

In the introductory example of classification we skipped a practical detail, which is the number of learners $M$.It is the main tunning parameter of boosting method and can be picked by monitoring the performance on a separate validation set, and then stopping once performance starts to decrease. This is called **early stopping** (alternatively one can use AIC or BIC as model selection criteria) [**?**].
In practice better performance can be obtained by partial updates of :

$$f_m(x) = f_{m-1}(x) + v\beta_m(x; \gamma_m),$$

where $0 < v \leq 1$ is a step size parameter, it is common to use a small value such as $v = 0.1$ - this is called **shrinkage**.

# Chapter 2

# Boosting algorithms

## 2.1 Types of boosting algorithms

### 2.1.1 L2Boosting

L2Boosting uses squared error as a loss function and can be used for regression. Optimal estimate $f^*$ is given by:

$f^*(x) = \underset{f}{\text{argmin}} = \mathbb{E}_{y|x}[(Y - f(x))^2] = \mathbb{E}[Y|x]$
At step m the loss has the form:
$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\text{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta\phi(x_i;\gamma)) = (r_{im} - \phi(x_i;\gamma))^2,$
where $r_{im} \triangleq y_i - f_{m-1}(x_i)$ is the current residual and $\beta = 1$ wlog.

To present the performance of L2Boosting algorithm, I applied it to the "bodyfat" dataset, which is available under TH.data package. The explanation and overview of the variables is shown in the Figure 2.1. Below analysis was executed using a package called "l2boost".
The data was divided randomly into train and test set with the proportion 0.8 to 0.2. To find the optimal number of steps, I used cv.l2boost() function and looped it over 10 shrinkage values - nu from 0.1 to 1 and 3 different number of learners: 100,1000,10000. Seed was set to 15091987 in order to ensure reproducibility. After first run, it was clear, that this data returns the same optimal number of iterations for 100, 1000 and 10000, so the decision was to drop two columns in the results below. Nevertheless, if this simulation is performed with different dataset, it would give different optimal M for each number of learners, depending on the fact if number of possibilities is exhausted. This cross-validation method tries to pick up the best model based

Table 1: Available variables in the `bodyfat` data, for details see Garcia et al. (2005).

| Name | Description |
|---|---|
| `DEXfat` | body fat measured by DXA (response variable) |
| `age` | age of the women in years |
| `waistcirc` | waist circumference |
| `hipcirc` | hip circumference |
| `elbowbreadth` | breadth of the elbow |
| `kneebreadth` | breadth of the knee |
| `anthro3a` | sum of logarithm of three anthropometric measurements |
| `anthro3b` | sum of logarithm of three anthropometric measurements |
| `anthro3c` | sum of logarithm of three anthropometric measurements |
| `anthro4` | sum of logarithm of four anthropometric measurements |

Figure 2.1: Dataset "bodyfat" from TH.data package[**?**].

on mean squared error (MSE). In the table below we will find the results for each nu value and optimal numbers of steps, MSE for cross validation can be found in the next column and MSE of the test set in the last one.

| Nr | Nu | M* | MSE CV | MSE test |
|---|---|---|---|---|
| 1 | 0.1 | 50 | 11.67 | 9.80 |
| 2 | 0.2 | 25 | 11.55 | 9.63 |
| 3 | 0.3 | 19 | 11.71 | 9.98 |
| 4 | 0.4 | 12 | 12.16 | 10.08 |
| 5 | 0.5 | 10 | 12.37 | 10.80 |
| 6 | 0.6 | 5 | 12.64 | 11.97 |
| 7 | 0.7 | 47 | 12.16 | 10.67 |
| 8 | 0.8 | 38 | 11.98 | 10.63 |
| 9 | 0.9 | 46 | 12.07 | 10.52 |
| 10 | 1.0 | 39 | 12.19 | 10.54 |

According to this experiment the winner boosted model consists of 25 models (M=25) and has shrinkage set to 0.2. In the plot of Figure 2.2 we can observe how it behaves in every step until reaching the optimum number of 25 steps. Left plot displays a gradient-correlation versus iteration steps, right one presents beta coefficients versus the step number m along the x-axis. Gradient correlation tends to 0 as it approaches the optimal number of iterations, which equals 25.

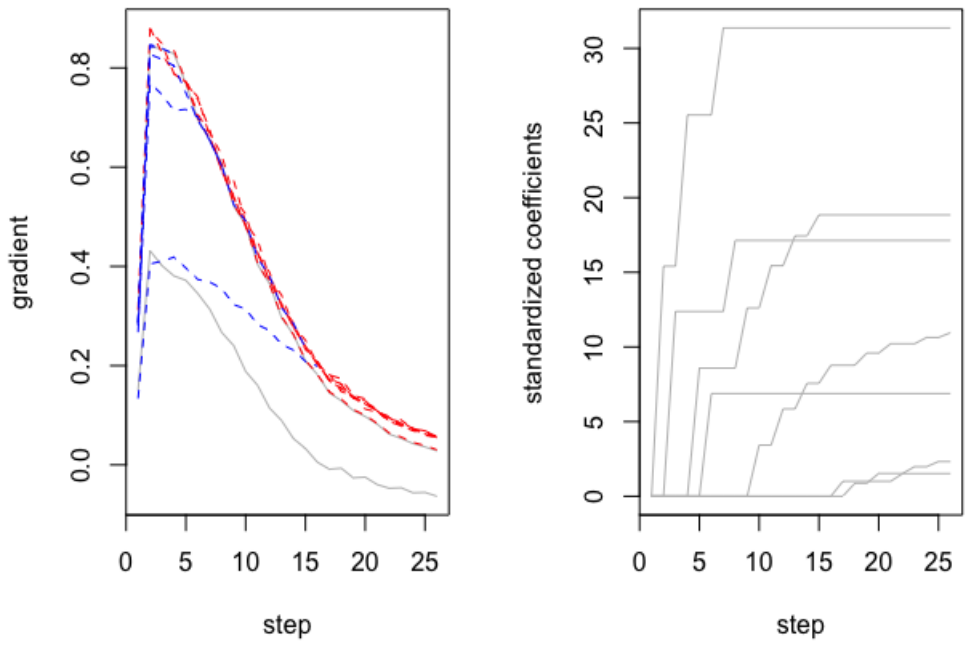



Figure 2.2: Results for best L2boosted model (M=24, nu = 1) , own calculations with dataset "bodyfat" from TH.data package

### 2.1.2 AdaBoost

AdaBoost owes its popularity due to stunning empirical success and the fact, that it is very slow to overfit. It operates with exponential loss function, which is accused of being too sensitive to outliers. Optimal estimate $f^*$ can be calculated as follows:

$$\frac{\partial}{\partial f(x)}\mathbb{E}[e^{-\tilde{y}f(x)}|x] = \frac{\partial}{\partial f(x)}[p(\tilde{y}=1|x)e^{-f(x)} + p(\tilde{y}=-1|x)e^{f(x)}]$$

$$= -p(\tilde{y}=1|x)e^{-f(x)} + p(\tilde{y}=-1|x)e^{f(x)}] \stackrel{!}{=} 0$$

$$\implies \frac{p(\tilde{y}=1|x)}{p(\tilde{y}=-1|x)} = e^{2f(x)} \text{ and } f^*(x) = \frac{1}{2}log\frac{p(\tilde{y}=1|x)}{p(\tilde{y}=-1|x)}$$

At step m we have to minimize: $L_m(\phi) = \sum_{i=1}^{N} exp[-\tilde{y}_i(f_{m-1}(x) + \beta\phi(x_i))] =$

$$\sum_{i=1}^{N} w_{i,m} exp(-\beta \tilde{y}_i \phi(x_i)),$$

where $w_{i,m} \triangleq exp(-\tilde{y}_i f_{m-1}(x_i))$ and $\tilde{y} \in \{-1, +1\}$

We can rewrite this:

$$L_m = e^{-\beta} \sum_{\tilde{y}_i = \phi(x_i)} w_{i,m} + e^{\beta} \sum_{\tilde{y}_i \neq \phi(x_i)} w_{i,m}$$

$$= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_{i,m} \mathbb{I}(\tilde{y} \neq \phi(x)) + e^{-\beta} \sum_{i=1}^{N} w_{i,m}$$

so the optimal function to add is:

$\phi_m = \underset{\phi}{\text{argmin}} w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi(x_i))$ Now substitute for $\phi_m$ into $L_m$ and solving for $\beta$:

$\beta = \frac{1}{2} log \frac{1 - err_m}{err_m}$, where $err_m = \dfrac{\sum_{i=1}^{N} w_{i,m} \mathbb{I}(\tilde{y} \neq \phi(x))}{\sum_{i=1}^{N} w_{i,m}}$

The overall update becomes: $f_m = f_{m-1}(x) + \beta_m \phi_m(x_i)$

The weights at the next iteration become:

$$w_{i,m+1} = w_{i,m} e^{-\beta_m \tilde{y}_i \phi_m x_i} = w_{i,m} e^{\beta_m (2\mathbb{I}(\tilde{y}_i \neq \phi_m x_i)) - 1)}$$

$$= w_{i,m} e^{\beta_m (2\mathbb{I}(\tilde{y}_i \neq \phi_m x_i))} e^{-\beta_m}$$

This pseudo-code imitates an algorithm for binary classification[**?**]:
1. $w_i = 1/N$
2. for $m = 1 : M$ do:
a) Fit a classifier $\phi_m(x)$ to the training set using weights w;
b) Compute $err_m = \dfrac{\sum_{i=1}^{N} w_{i,m} \mathbb{I}(\tilde{y} \neq \phi(x))}{\sum_{i=1}^{N} w_{i,m}}$;

c) Compute $\beta = 1/2log[(1 - err_m)/(err_m)]$;
d) Set $w_i \leftarrow w_i exp[\beta_m \mathbb{I}(\tilde{y}_i \neq \phi_m(x_i))]$
3. Return $f(x) = sgn[\sum_{m=1}^{M} \beta_m \phi_m(x)]$;

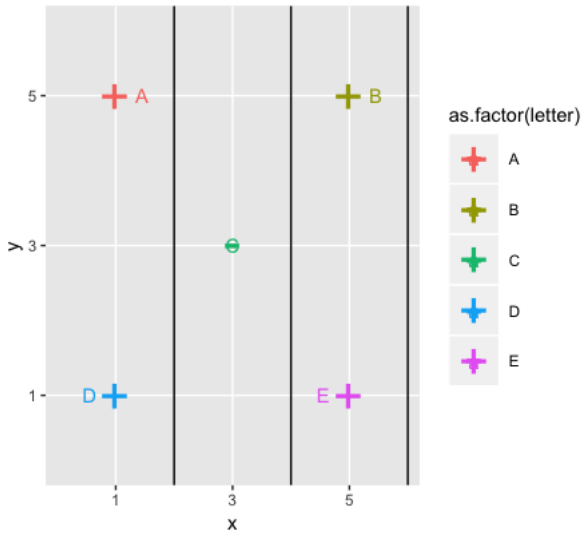



Figure 2.3: Visualization of example for AdaBoost, adapted from MIT tutorial

For the demonstration purposes I will describe below one exercise taken from the tutorials on AdaBoost at the MIT (Massachusetts Institute of Technology)[**?**]. Tables below contain the calculations for 3 rounds of this algorithm. First table consists of weights for every rounds, second calculated error rates of decision stumps. We want to investigate binary classification problem of 5 points: A (1,5), B (5,5), C(3,3), D(1,1), E(5,1). The coordinates of the points are given in the brackets next to the letters. Letters: A, B, D and E are "+", they represent one category, only C belongs to the another category and is marked as "-". The following weak learners are considered in this case : $x < 2$, $x < 4$, $x < 6$. They mean, that everything on the right is a minus and everything on the left is a plus, so for instance for $x < 2$ every point that is smaller than 2 will be assigned to "+", otherwise to "-". Additionally, we will add the contradictory decision stumps: $x > 2$, $x > 4$, $x > 6$.

| Point | Round 1 Weights | Round 2 Weights | Round 3 Weights |
|---|---|---|---|
| A | $w_a = 1/5$ | $w_a = 1/8$ | $w_a = 1/12$ |
| B | $w_b = 1/5$ | $w_b = 1/8$ | $w_b = 1/4$ |
| C | $w_c = 1/5$ | $w_c = 1/2$ | $w_c = 4/12$ |
| D | $w_d = 1/5$ | $w_d = 1/8$ | $w_d = 1/12$ |
| E | $w_e = 1/5$ | $w_e = 1/8$ | $w_e = 1/4$ |

Now it is also a good time to define, in which point learners fail to predict accurately. The errors of each learner are listed in the third column of the table below. Application of AdaBoost demands setting the weights with the equal importance in the first step: 1/n = 1/5. Then we will continue with the algorithm and calculate the error rate for every decision stump. As we can see first learner $x < 2$ makes mistakes in B and E, so after summing up their weights we will get an error rate of 2/5: 1/5 + 1/5 = 2/5. After computing error rates for the rest decision stumps, we need to determine, which one bring the lowest error rate (the lowest error rate is marked with * ). In the first round it is the third learner: $x < 6$. For this learner we will calculate $\beta = 1/2log[(1-err_m)/(err_m)] = 1/2log[(1-1/5)/(1/5)] = 1/2log4$. At this point and after each round, we need to fulfill the following conditions in order to decide if to stop the algorithm: 1. classifier is good enough, 2. number of rounds has been exhausted, 3. no good classifier is left. Since one round is not enough, we will proceed to the next and here again weights must be updated first. This is different than before, because we have the knowledge from the previous round, which learner is the best and its error rate. Therefore, we set the weight of C to 1/2 to maintain the assumption of a weak learner, that the error rate is close to random guessing 1/2, because only C is misclassified using $x < 6$ . The general fact is that, the weights add up to 1 and the sums of the wrong and right weights are equal. According to that, the sum of the rest weights must be 1/2 and they are equal to each other, so $w_a = w_b = w_d = w_e = 1/8$. Error rate for $x < 2$ will be now $w_b + w_e = 1/8 + 1/8 = 2/8$, for $x < 4 : w_b + w_c + w_e = 1/8 + 1/2 + 1/8 = 6/8$ and so on. In the second round we deal with a tie, both first and the 5th learner have the same error rate, we would prefer the earlier one, but basically it is up to us. In this round, computing weights is more challenging, earlier we used some convenient tricks to overcome computations, but in the third iteration a proper formula for weights is required:
$w_{m+1} = 1/2(1/1 - err_m)w_m$, if the points was correctly classified
$w_{m+1} = 1/2(1/err_m)w_m$, if the points was misclassified.
Applying it to A and knowing that : $err_2 = 2/8$, $w_a, 2 = 1/8$, we will get

$1/2 * (1/1 - 2/8) * 1/8 = 1/12$. We used the first formula, because A was correctly classified by the $x < 2$. In order to compute error rate in point B, we will use second formula, as it was misclassified in the previous round. The winner of this round is the classifier $x > 4$ with the error rate of 1/6.

| Nr | decision stump | error | Round 1 error | Round 2 error | Round 3 error |
|---|---|---|---|---|---|
| 1 | $x < 2$ | B, E | 2/5 | 2/8* | 2/4 |
| 2 | $x < 4$ | B, C, E | 3/5 | 6/8 | 10/12 |
| 3 | $x < 6$ | C | 1/5* | 4/8 | 4/12 |
| 4 | $x > 2$ | A, C, D | 3/5 | 6/8 | 2/4 |
| 5 | $x > 4$ | A, D | 2/5 | 2/8* | 2/12* |
| 6 | $x > 6$ | A, B, D, E | 4/5 | 4/8 | 8/12 |
| best learner | | | $x < 6$ | $x < 2$ | $x > 4$ |
| $err_m$ | | | 1/5 | 2/8 | 2/12 |
| $\beta$ | | | 1/2 log 4 | 1/2 log 3 | 1/2 log 5 |

The overall classifier H(x) can be written as :

$$f(x) = sgn[1/2log4(x < 6) + 1/2log3(x < 2) + 1/2log5(x > 4)]$$

We may stop the algorithm right now, because the overall classifier consisting of 3 classifier can be considered as good enough. First classifier $x < 6$ makes mistakes in C, second $x < 2$ in B, E and third $x > 4$ in A, D, so the overall classifier is powerful enough to classify correctly the points.

### 2.1.3 LogitBoost

Optimal estimate for the binary classification is defined as :

$$f^*(x) = \frac{1}{2} log \frac{p(\tilde{y}=1|x)}{p(\tilde{y}=-1|x)}$$

where $\tilde{y} \in \{-1, +1\}$

and goal is to minimize the expected log-loss by:

$$L_m(\phi) = \sum_{i=1}^{N} log[1 + exp(-2\tilde{y}_i(f_{m-1}(x) + \phi(x_i)))]$$

LogitBoost, for binary classification with log-loss looks like this [**?**]:

1. $w_i = 1/N, \pi_i = 1/2$;
2. for $m = 1 : M$ do:
a. Compute the working response $z_i = \frac{y_i^* - \pi_i}{\pi_i(1-\pi_i)}$;
b. Compute the weights $w_i = \pi_i(1 - \pi_i)$;
c. $\phi_m = \underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{N} w_i(z_i - \phi_(x_i))^2$;
d. Update $f(x) \leftarrow f(x) + 1/2\phi_m(x)$;
e. Compute $\pi_i = 1/(1 + exp(-2f(x_i)))$;
3. Return $f(x) = sgn[\sum_{m=1}^{M} \phi_m(x)]$;
As explained by Friedman in his work, it can be generalized to the multi-class [**?**].

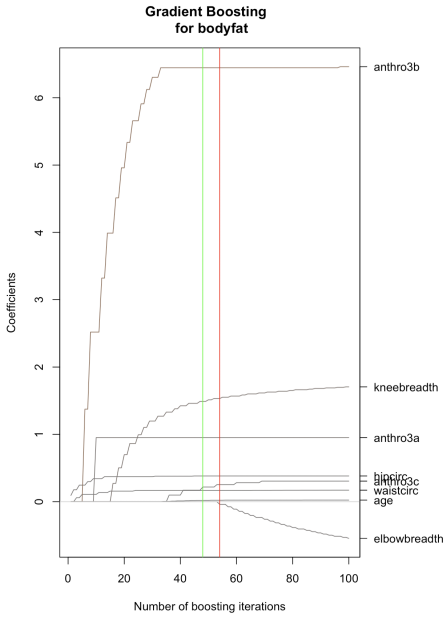
### 2.1.4 Boosting as functional gradient

Instead of deriving different algorithm for every loss function we can derive one common algorithm called gradient boosting. It uses gradient descent for stagewise solution. Gradient descent is the first order iterative optimization algorithm for finding a minimum of a function, it can be also illustrated by the following analogy [**?**]:
A person is stuck in the mountains and is trying to get down, in other words as if algorithm is trying to get minima. The weather conditions are very poor, heavy fog prevents visibility. Therefore, the path down the mountain is not visible, so he must use local information to find the minima. He can use the method of gradient descent, which involves looking at the steepness of the hill at his current position, then proceeding in the direction with the steepest descent (i.e. downhill). If he was trying to find the top of the mountain (i.e. the maxima), then he would proceed in the direction steepest ascent (i.e. uphill). Using this method, he would eventually find his way down the mountain. However, assume also that the steepness of the hill is not immediately obvious with simple observation, but rather it requires a sophisticated instrument to measure, which the person happens to have at the moment. It takes quite some time to measure the steepness of the hill with the instrument, thus he should minimize his use of the instrument if he wanted to get down the mountain before sunset. The difficulty then is choosing the frequency at which he should measure the steepness of the hill so not to go off track.

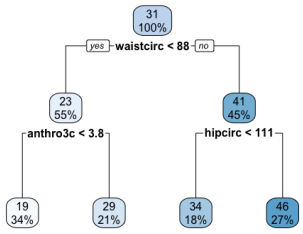
The algorithm itself looks as follows[**?**] :
1. Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \phi(x_i; \gamma))$;
2. for $m = 1 : M$ do:

a. Compute the gradient residual using $r_{im} = -[\frac{\partial L(y_i; f(x_i))}{\partial f(x_i)}]_{f(x_i)=f_{m-1}(x_i)}$
b. Use the weak learner to compute $\gamma_m$ which minimizes $\sum_{i=1}^{N} r_{im} - \phi(x_i; \gamma_m))^2$
c. Update $f_m(x) = f_{m-1}(x) + v\phi(x; \gamma_m)$;
3. Return $f(x) = f_M(x)$

The benefits of gradient boosting is that, it can be applied with various loss functions, with squared loss we will get L2Boosting, with log-loss - BinomialBoost, or can be easily extended to multi-class case.




(a) Gradient Boosting of bodyfat, own calculations, optimal iteration according to AIC in green, to CV in red.




(b) Regression Tree of bodyfat, own calculations.

Figure 2.4: Training of bodyfat dataset in R, using boosting (left) and regression tree (right).

Package "mboost" in R enables to implement this algorithm to "bodyfat" dataset. I trained several different models and compared their performance on the train set and test set, such that the metrics in sample and out of sample estimations could be available to make a decision, which model should be preferred. Root of mean squared error (RMSE) was used as a metric for diagnostic checking , which is just the square root of mean squared error and mean squared error is just the averaged sum of squared differences between true and estimated values. In the below table we can see the outputs for:

gradient boosting, regression tree, random forest and simple OLS estimation (ordinary least square).
Gradient boosting was trained primarily with number of iterations equals to 100 and from that the best model was picked according to Akaike Information Criterion (AIC). This approach allows to select the number of M, which is equivalent to the number of iterations to ensure early stopping before overfitting. This criterion attains here its minimum at 48th step. Another way to identify the optimal number of models can be achieved by resampling methods such as cross-validation or the bootstrap. They can be used to estimate the out-of- sample error for a varying number of boosting iterations. "Mboost" offers for that purposes an automatic function, that returns an optimal number of iterations according to empirical risk. In this example 10-fold cross validation was used, the appropriate number of boosting iterations was reached at 54. It needs to be mentioned, that like in the previous example with L2Boosting, it is required to set the value for seed for the sake of reproducibility and was set to 1509. The graphical representation of it is in the Figure 2.4 on the left. Green line shows the optimal number of iterations according to AIC and red line according to cross validation and its empirical risk. A plot depicts the coefficient paths, similar to the ones commonly known from the LARS algorithm. All of them were trained with the shrinkage value $\nu = 0.1$.
Regression tree was built with the help of "rpart" package and the product of this estimation is visible in Figure 2.4 on the right. The advantage of regression tree is, that it performs the variable selection. In the picture we can see, that 3 variables out of 9 are significant for this problem: waistcirc, anthro3c and hipcirc. Regression tree can be optimized by usage of k-fold cross validation. In this case 5-folds with 3 repetitions were executed, the same setting was passed on the random forest. Last but not least two estimation based on OLS was performed, one with all variables and second one with the reduced number of variables based on Akaike Information Criterion. The smallest value is obtained for the model with the following variables:
DEXfat  waistcirc + hipcirc + kneebreadth + anthro3b .

| Model Name | RMSE train | RMSE test | $R^2$ |
|---|---|---|---|
| Gradient Boosting CV | 3.07 | 3.13 | 0.93 |
| Gradient Boosting AIC | 3.08 | 3.12 | 0.93 |
| Regression Tree | 4.50 | 5.05 | 0.84 |
| Regression Tree with CV | 6.56 | 2.51 | 0.71 |
| Random Forest | 3.67 | 1.90 | 0.92 |
| OLS | 3.02 | 3.22 | 0.91 |
| OLS tunned | 3.09 | 3.17 | 0.92 |

Root mean square error of train set clearly points out OLS to be the winner of this competition, boosted model with optimal iteration set according to cross validation is the second best model. However, when we take RMSE of test set under consideration, then the best model would be random forest. On the other hand coefficient of determination $R^2$ awards both versions of boosting with the first place. Nonetheless, we should not forget about the result from L2Boosting, which with RMSE of train set at the level of $\sqrt{9}.63 = 3.10$ and RMSE of test set being $\sqrt{1}1.55 = 3.40$ results in not bad model, but worse than boosting versions with AIC or CV optimal number of iterations.

# Chapter 3

# Conclusion

In this report I presented the idea of boosting and its different algorithms: L2Boosting, AdaBoost, LogitBoost and gradient boosting. L2Boosting and gradient boosting were tested with the dataset available in library called "TH.data". Additionally, other models were trained to compare the results and to show how strong boosting performs. In the demonstrated example boosting won considering $R^2$ and had second best results of root mean square error of the training set. Furthermore, detailed calculations of AdaBoost algorithm were shown on another theoretical classification problem in order to familiarize with the mathematics behind this concept.

# References


[1] Richard O. Duda, Peter E. Hart, David G. Stork (2001), Pattern Classification *New York [u.a.] : Wiley* , pages 26 – 29.

[2] Kevin P. Murphy (2012) Machine Learning: A Probabilistic Perspective *MIT Press 2012* , pages 554 – 563.

[3] Wikipedia: Greedy algortihm `https://en.wikipedia.org/wiki/Greedy_algorithm`, Accessed November 2018

[4] Benjamin Hofner,Andreas Mayr, Nikolay Robinzonov, Matthias Schmid,`https://cran.r-project.org/web/packages/mboost/vignettes/mboost_tutorial.pdf`, Model-based Boosting in R, Accessed November 2018

[5] Boosting (Adaboost), tutorials MIT,`https://www.youtube.com/watch?v=gmok1h8wG-Q&t=1065s`, Accessed November 2018

[6] Wikipedia: Gradient descent `https://en.wikipedia.org/wiki/Gradient_descent`, Accessed November 2018