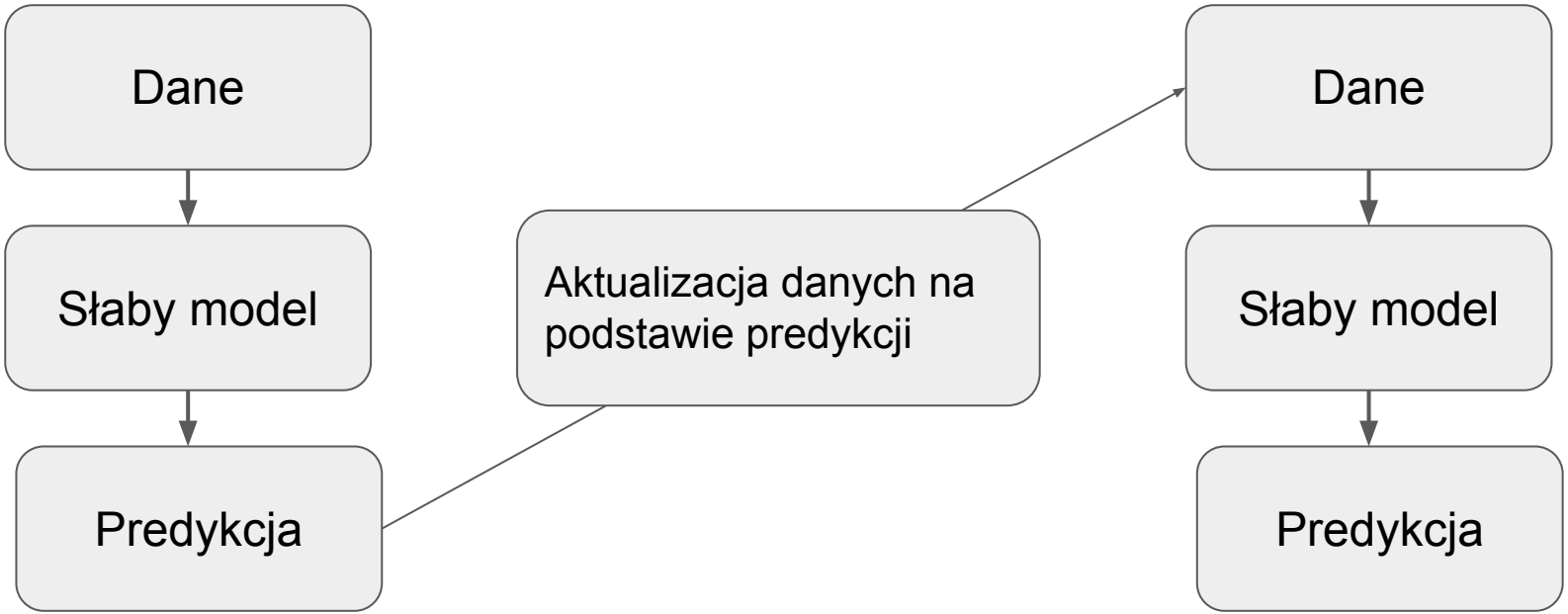
Urządzenie Wzmacniające Nachylenie

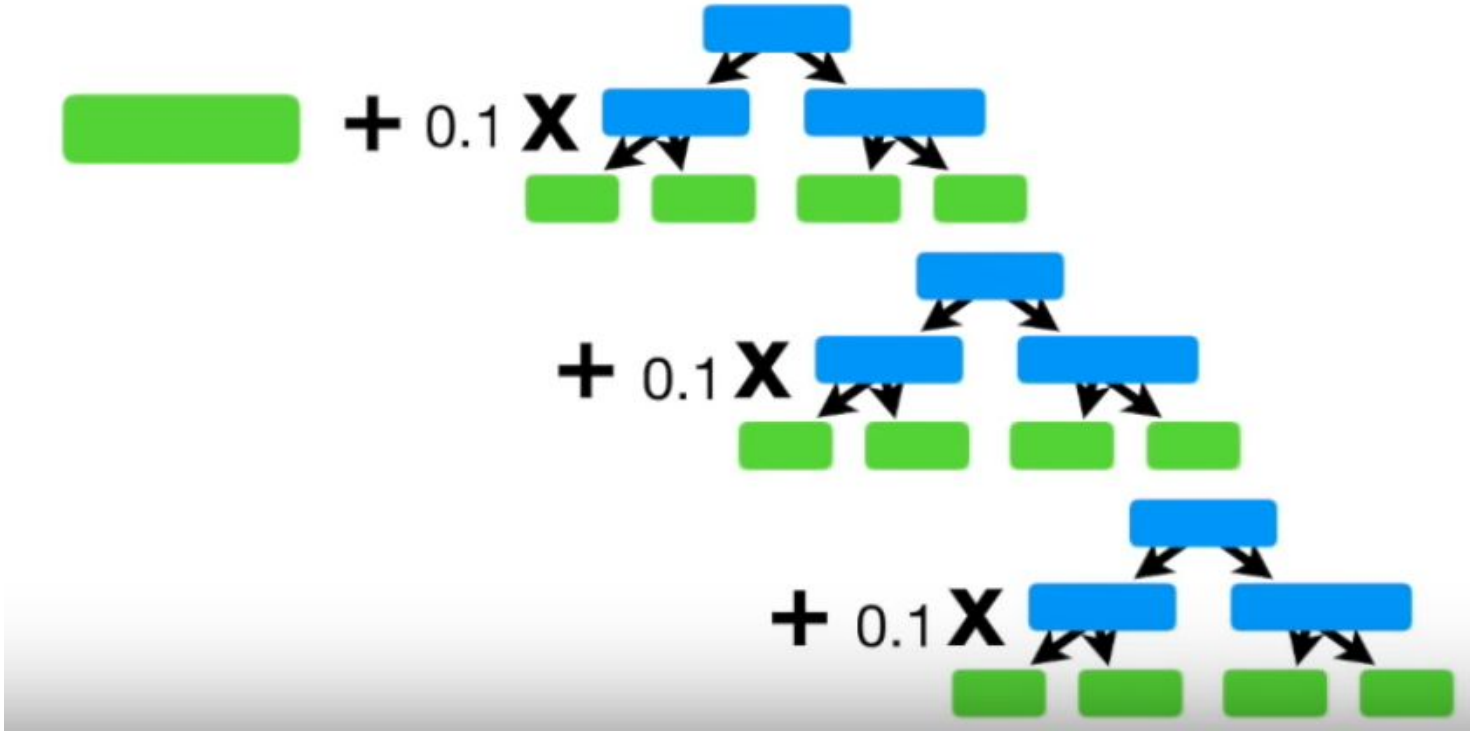
a.k.a. Gradient Boosting Machine

Spis treści

Idea - boosting



Gradient boosting i drzewa decyzyjne



Algorytm - co jest potrzebne?

- Dane wejściowe $\{(x_i, y_i)\}_{i=1}^{n}$
- Różniczkowalna funkcja straty $L(y_i, F(x))$
- Dla regresji najczęściej

  L=0.5(obserwacja - predykcja)^2

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Algorytm - inicjalizacja

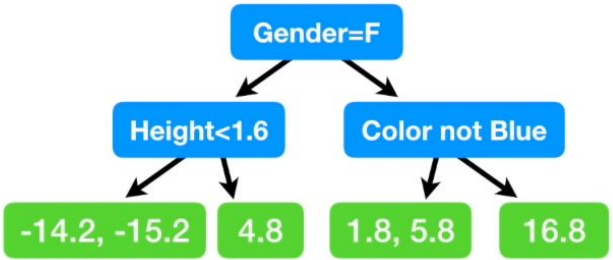
- Pierwszy model $F_0(x)$ jest drzewem składającym się z jednego liścia
- Dla L=0,5(obserwacja - predykcja)^2 $F_0(x)$ jest średnią arytmetyczną
- Ogólnie $F_0(x)$ wyraża się wzorem $$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$$

Algorytm spadku gradientów

- Liczymy pseudo rezydua używając predykcji poprzedniego modelu

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1,\ldots,n$$

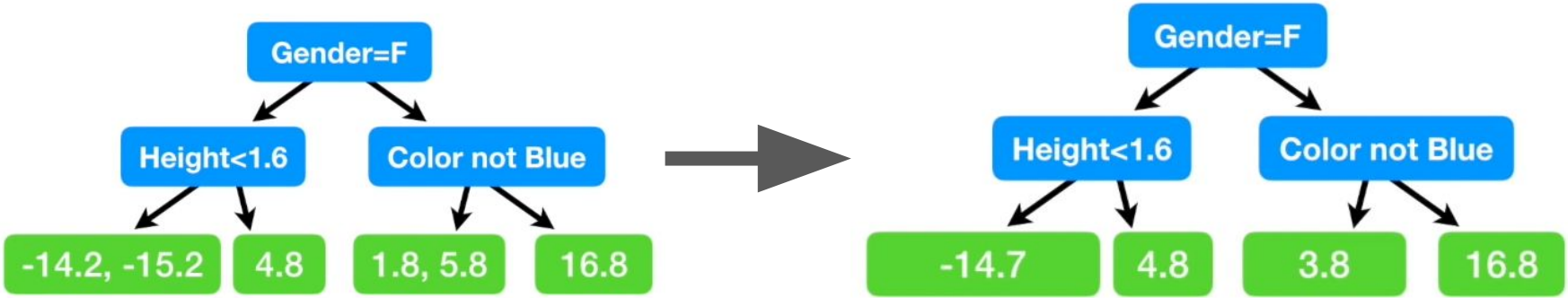
- Dopasowujemy drzewo decyzyjne do wartości r_{im} i otrzymujemy liście R_{jm}(j-ty liść w drzewie m)



Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

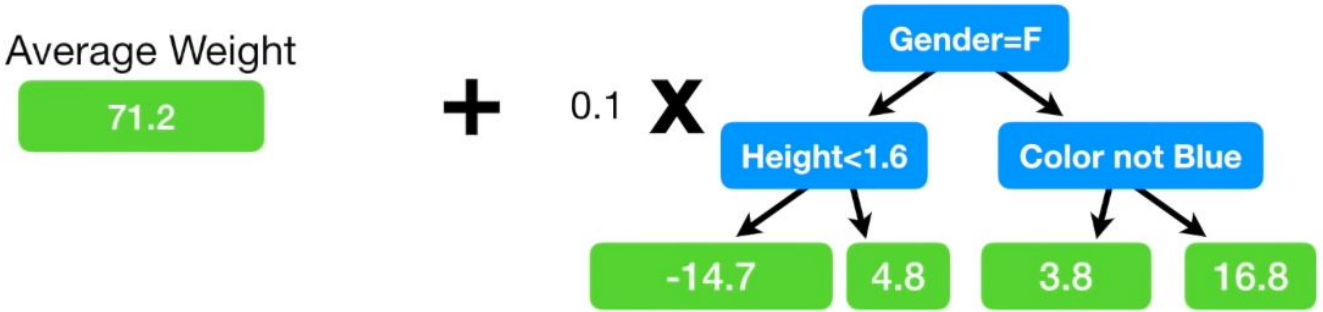
Algorytm spadku gradientów

- Dla każdego liścia w nowym drzewie liczymy $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$



Algorytm spadku gradientów

- Aktualizujemy $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$



Algorytm spadku gradientów

- Nowe rezydua są bliższe zeru niż poprzednie
- Spadek gradientów wykonujemy dla modeli od 1 do M, gdzie M to ustalona liczba drzew
- $F_M(x)$ jest wynikiem działania algorytmu.

Residual	Residual
16.8	15.1
4.8	4.3
-15.2	-13.7
1.8	1.4
5.8	5.4
-14.2	-12.7

Klasyfikacja a regresja

- regresja: przewidywanie wartości ciągłej
- klasyfikacja: przewidywanie wartości binarnej

Jak wykorzystać regresję w klasyfikacji??

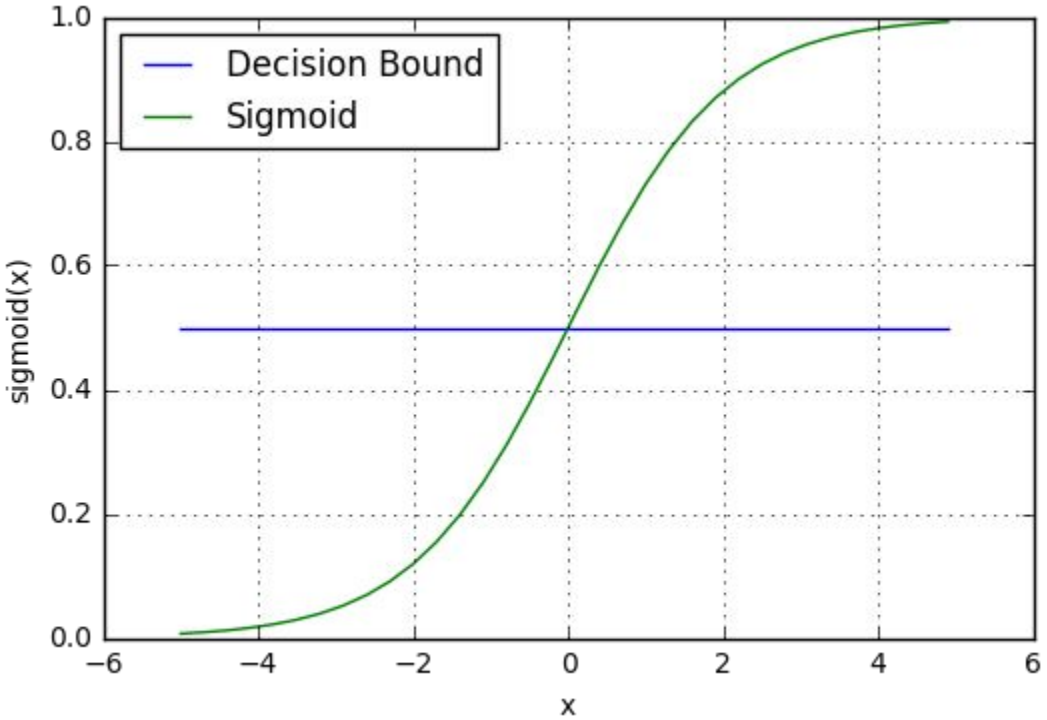
Zmienić wartość ciągłą na binarną

Klasyfikacja a regresja

Jak wykorzystać regresję w klasyfikacji??

Zmienić wartość ciągłą na binarną

$$f(x) = \frac{1}{1 + e^{-(x)}}$$



Klasyfikacja - pierwszy model

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

logarytm szans

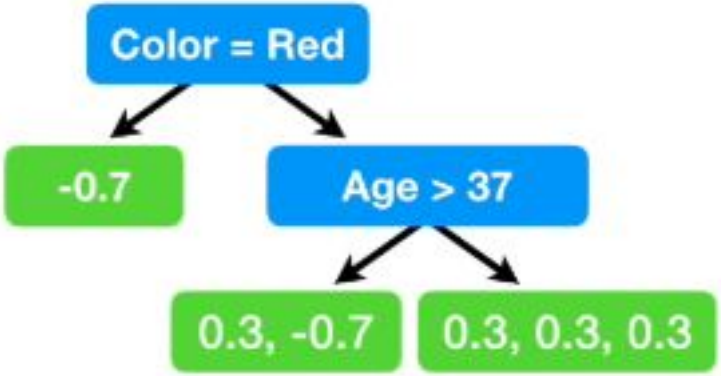
$$\log(\frac{4}{2}) = 0.6931$$

$$\frac{e^{\log(4/2)}}{1 + e^{\log(4/2)}} = 0.6667$$

prawdopodobieństwo pozytywnego wyniku

Klasyfikacja - drugi model

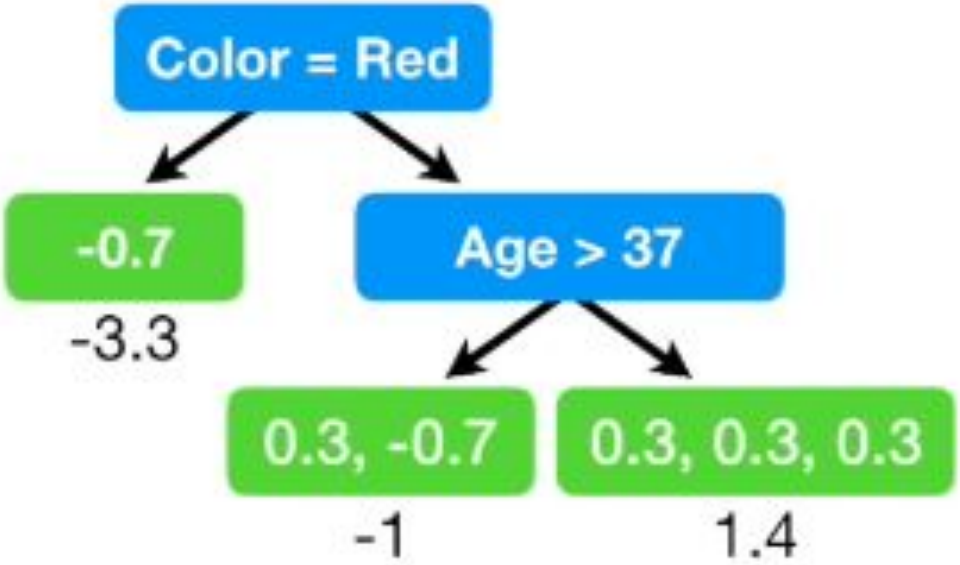
Residual = True - Predicted



Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3

Klasyfikacja - drugi model

$$\frac{\sum \text{Residual}_i}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]}$$



Klasyfikacja - predykcja

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes

$$\log(\text{odds}) \text{ Prediction} = 0.7 + (0.8 \times 1.4) = 1.8$$

$$\text{Probability} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

Klasyfikacja - predykcja

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	0.5
No	44	Blue	No	0.5	-0.5
Yes	19	Red	No	0.1	-0.1
No	32	Green	Yes	0.9	0.1
No	**14**	**Blue**	**Yes**	**0.9**	**0.1**

Klasyfikacja - algorytm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step 2: for m = 1 to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Step 3: Output $F_M(x)$

Klasyfikacja - algorytm

$$y_i \times \log(p) + (1 - y_i) \times \log(1 - p)$$

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum L(y_i, F_{m-1}(x_i) + \gamma)$

Klasyfikacja - algorytm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Step 3: Output $F_M(x)$

Podobne modele

1. Random Forest
    a. modele niezależne
    b. mniejszy overfitting niż w GBM
    c. GBM redukuje bias
2. Ada boost
    a. małe modele - tylko po 2 liście
    b. różna waga kolejnych modeli (różny learning rate)

```
library(OpenML)
library(gbm)

dset ← getOMLDataSet(1067)$data
levels(dset$defects) ← c(0, 1)

m_base ← gbm(defects~.,
             distribution = "bernoulli",
             data = dset, cv.folds = 10)
```

```
library(OpenML)
library(mlr)

dset ← getOMLDataSet(1067)$data
levels(dset$defects) ← c(0, 1)

p_task ← makeClassifTask("task", data = dset,
                                 target = "defects")
p_learner ← makeLearner("classif.gbm")

model ← crossval(p_learner, p_task, 10)
```

```
Resampling: cross-validation
Measures:             auc
[Resample] iter 1:  0.7597953
[Resample] iter 2:  0.8037634
[Resample] iter 3:  0.7918122
[Resample] iter 4:  0.8469841
[Resample] iter 5:  0.7868421
[Resample] iter 6:  0.8615101
[Resample] iter 7:  0.8237605
[Resample] iter 8:  0.8061278
[Resample] iter 9:  0.7987526
[Resample] iter 10:   0.7845578


Aggregated Result: auc.test.mean=0.8063906
```

Wady i zalety

+ duża skuteczność
+ duża dostosowywalność (ale przez to wymaga dużo pracy)
+ brak konieczności wstępnej obróbki danych, radzi sobie z brakami

- groźba silnego overfittingu
- duży koszt obliczeniowy i pamięciowy
- niska interpretowalność (względem modeli prostych)

Referencje

- **UC Business Analytics R Programming Guide**
  - http://uc-r.github.io/gbm_regression
- Gradient boosting machines, a tutorial
  - https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full
- StatQuest:
  - https://www.youtube.com/watch?v=StWY5QWMXCw
  - https://www.youtube.com/watch?v=jxuNLH5dXCs
  - https://www.youtube.com/watch?v=2xudPOBz-vs
  - https://www.youtube.com/watch?v=3CC4N4z3GJc