

# Machine learning

- evaluating performance
- improving performance
  - parameter tuning
  - ensemble methods

Ensemble learning assumes that we may not always be able to find the optimal set of hyperparameters for a single model and that, even if we did, the model may not always be able to capture all the underlying patterns in the data.

Therefore, instead of simply focusing on optimizing the performance of a single model, we should use several complementary weak models to build a much more effective and powerful model.

Ensemble learning is a method to combine results produced by different learners into one format, with the aim of producing better classification results and regression results.

The most common methods:

- bagging
- random forest
- boosting

**Bagging** is a voting method, which first uses Bootstrap to generate a different training set, and then uses the training set to make different base learners. The bagging method employs a combination of base learners to make a better prediction.

**Random forest** uses the classification results voted from many classification trees. The idea is simple; a single classification tree will obtain a single classification result with a single input vector. However, a random forest grows many classification trees, obtaining multiple results from a single input. Therefore, a random forest will use the majority of votes from all the decision trees to classify data or use an average output for regression.

**Boosting** is similar to the bagging method. However, what makes boosting different is that it first constructs the base learning in sequence, where each successive learner is built for the prediction residuals of the preceding learner. With the means to create a complementary learner, it uses the mistakes made by previous learners to train the next base learner.

Methods of constructing ensemble models may involve modifying

- **training data** - the additional training data sets used by the individual classifiers are created by a multiple sampling with a return;
- **lists of variables** - the training data of individual classifiers contain only the variables selected (in the simplest case, randomly) from the original training data set;
- **machine learning algorithm** - individual classifiers are learned using the same training dataset, but each algorithm has different parameters.

### Bagging:

If exactly the same predictors are used to build each tree in bagging, it may happen that in the case of variables that are very strongly correlated with the phenomenon being explained, that each tree selects the same variable for the first split, even though other variables may be only slightly less correlated with the target variable.

### Random forest:

A random forest is a stable and non-overfitted model. Its weakness is its sometimes weak predictive ability.

Ensemble methods, which combine the prediction power of each single learner into a strong learner.

The trees in the random forests and bagging can be developed in parallel calculations.



Boosting starts with a simple or weak classifier and gradually improves it by **reweighting the misclassified observations**. Thus, the new classifier can learn from previous classifiers.

Boosting algorithm is a sequential ensemble method in that the residual/misclassified point of the previous learner is improvised in the next run of the algorithm.

The idea of boosting is to "boost" weak learners, a single decision tree, into strong learners.

An alternative to repeated sampling is to generate modified copies of the train data by modifying the weights vector. Base models  $h_1, h_2, \dots, h_m$  would then be generated using the same training set, but different vectors of weights  $w_1, w_2, \dots, w_m$ .

- let us assume that we have  $n$  points in our training dataset and we can assign a weight,  $w_i$  ( $1 \leq i \leq n$ ), for each point.
- during  $m$  iterations, we can reweight each point in accordance with the classification result in each iteration.
- if the point is correctly classified, we should decrease the weight. Otherwise, we increase the weight of the point.
- when the iteration process is finished, we can then obtain the  $m$  fitted model,  $f_i(x)$  ( $1 \leq i \leq m$ ).

1. Set the weight vector,  $w$ , to uniform weights, where  $\sum_i w_i = 1$ .
2. For  $j$  in  $m$  boosting rounds, do the following:
  - a. Train a weighted weak learner:  $C_j = \text{train}(X, y, w)$ .
  - b. Predict class labels:  $\hat{y} = \text{predict}(C_j, X)$ .
  - c. Compute the weighted error rate:  $\varepsilon = w \cdot (\hat{y} \neq y)$ .
  - d. Compute the coefficient:  $\alpha_j = 0.5 \log \frac{1-\varepsilon}{\varepsilon}$ .
  - e. Update the weights:  $w := w \times \exp(-\alpha_j \times \hat{y} \times y)$ .
  - f. Normalize the weights to sum to 1:  $w := w / \sum_i w_i$ .
3. Compute the final prediction:  $\hat{y} = (\sum_{j=1}^m (\alpha_j \times \text{predict}(C_j, X)) > 0)$ .

- let us assume that we have ten observations. Each observation will have initial weight 0.1.
- we build a decision tree that misclassifies, for example, four observations (7, 8, 9 and 10).
- we can calculate the sum of the weights of these misclassified observations, which is 0.4 (we denote it by  $\epsilon$ ).
- We continue to use the value of  $\epsilon$  as the measure used to update the weights and determine the weight of the model.

- let's calculate  $\alpha$

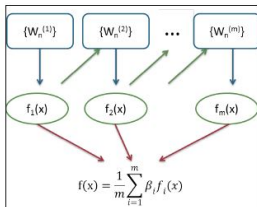
$$\alpha = 0.5 \cdot \log \left( \frac{1 - \epsilon}{\epsilon} \right).$$

- the new weights for misclassified observations are calculated as  $e^\alpha$  multiplied by the old weight.
- so  $\alpha = 0.2027 = 0.5 \cdot \log \left( \frac{1-0.4}{0.4} \right)$ , new weights for 7, 8, 9 and 10 are equal as  $e^\alpha \times 0.1 = 0.1225$ .

- this new model will again have errors. Suppose this model misclassifies observations 1 and 8. Their current weights are 0.1 and 0.1225, respectively.
- thus, the new value of  $\epsilon = 0.1 + 0.1225 = 0.2225$ . Next  $\alpha = 0.6256$ .
- We use this value to modify the weights of misclassified observations. Observation 1 now gets weight  $0.1 \cdot e^{\alpha} = 0.1869$ , Observation 8 now gets weight  $0.1225 \cdot e^{\alpha} = 0.229$ .

The idea of boosting is to "boost" weak learners, a single decision tree, into strong learners.

- finally, we can obtain the final prediction through the weighted average of each tree's prediction, where the weight,  $\beta$ , is based on the quality of each tree.



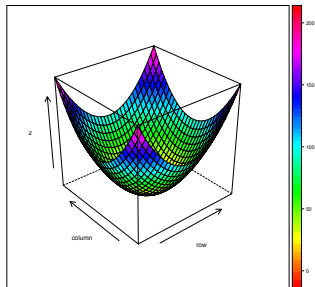
- AdaBoost also called Adaptive Boosting is a technique in machine learning used as an ensemble method.
- the most common algorithm used with AdaBoost is decision trees with one level that means with decision trees with only 1 split. These trees are also called **decision stumps**.



Raffle 2021-gis-19 19:04:13 Monika

The disadvantage of AdaBoost is its lack of robustness to noisy data and to outliers, because the algorithm tries to perfectly match each observation.





```
f <- function (x, y) {  
  return (x^2 + y^2)  
}  
  
x <- seq(-10, 10, length= 30)  
y <- x  
z <- outer(x, y, f)  
library(lattice)  
wireframe(z, drape=T,  
           col.regions=rainbow(100))  
contour(x,y,z)
```



**SLIPS and FALLS**

**Gradient descent** is an optimization algorithm for finding a local minimum of a differentiable function.  $f : \mathbb{R}^n \supset D \rightarrow \mathbb{R}$ .

**Gradient boosting** is considered a gradient descent algorithm.

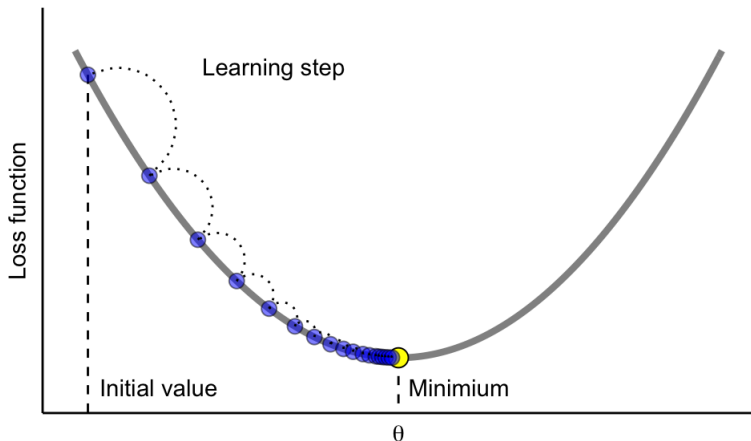
The general idea of **gradient descent** is to tweak parameter(s) iteratively in order to minimize a cost function.

The gradient descent measures the local gradient of the **loss (cost) function** for a given set of parameters and **takes steps in the direction of the descending gradient**.

- 1 select starting point  $x^0$ , determine the step (learning rate)  $h$
- 2  $k = 0$
- 3  $x^{k+1} = x^k - h \nabla f(x^k)$
- 4  $k = k + 1$
- 5 check the stop criterion:  $\|\nabla f(x^k)\| \leq \epsilon$ ,  $\|x^{k+1} - x^k\| \leq \epsilon$

## Gradient descent - Learning Rate

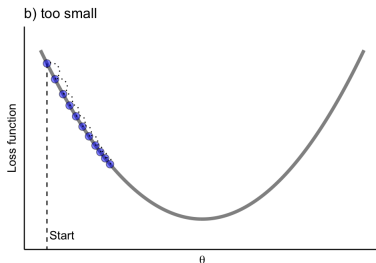
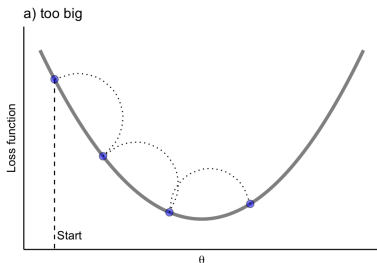
Gradient descent is the process of gradually decreasing the cost function (for instance MSE) by tweaking parameter(s) iteratively until you have reached a minimum.



## Gradient descent - Learning Rate

An important parameter in gradient descent is the size of the steps which is controlled by the **learning rate**. If the learning rate is **too small**, then the algorithm will take many iterations (steps) to find the minimum.

On the other hand, if the learning rate is **too high**, the algorithm might jump across the minimum and end up further away than when it started.



# Gradient Boosting Method

The main idea behind this algorithm is to build models sequentially and these subsequent models try to reduce the errors of the previous model. Minimizing the error is achieved by building a new model on the errors or residuals of the previous model.

- When the target variable is continuous, we use Gradient Boosting Regressor whereas when it is a classification problem, we use Gradient Boosting Classifier.
- The only difference between regression and classification is the **cost(loss) function**.
- The objective here is to minimize this loss function by adding weak learners using gradient descent.
- loss function for regression problems
  - **for regression problems** - we'll have different loss functions like Mean squared error (MSE)
  - **for classification problems** - we will have different for e.g log-likelihood.

gradient boosting builds a series of trees, where each tree is fit on the error—the difference between the label and the predicted value—of the previous tree  $y - \hat{y}$ . In each round, the tree ensemble improves as we are nudging each tree more in the right direction via small updates. These updates are based on a loss gradient, which is how gradient boosting got its name.



- ❶ first model  $F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$
- ❷ **for**  $k = 1$  **to**  $N$
- ❸ calculate the residua  $r_{ik} = -\eta \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{k-1}(x)}$ , for  
 $i = 1, \dots, n$ ,  $\eta$  value of learning rate
- ❹  $h_k$  is model build on  $r_{ik}$
- ❺ we calculate the rate of step  $\gamma_k$
- ❻  $F_k = F_{k-1} + \gamma_k h_k$

## Gradient boosting algorithm for binary classification

We will be using the **logistic loss** function for a single training example  $i$ , we can specify the logistic loss as follows:

$$L_i = -y_i \log p_i + (1 - y_i) \log(1 - p_i)$$

and  $\log(\text{odds})$ :

$$\hat{y} \log(\text{odds}) = \log \left( \frac{p}{1 - p} \right)$$

Logit function, logit is a function used in statistics, in the logistic regression method, to transform probability into the logarithm of the odds

$$\text{logit}(p) = \log \frac{p}{1 - p} = \log(p) - \log(1 - p)$$

The inverse transformation is the standard logistic function

$$p = \frac{e^{\text{logit}(p)}}{1 + e^{\text{logit}(p)}}$$

The loss function can also be represented as

$$L_i = \log(1 + e^{\hat{y}_i}) - y_i \hat{y}_i$$

then

$$\frac{\partial L_i}{\partial \hat{y}_i} = \frac{e^{\hat{y}_i}}{1 + e^{\hat{y}_i}} - y_i = p_i - y_i$$

So

$$\gamma_k = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jk}}^n L(y_i, F_{k-1}(x_i) + \gamma) = \log(1 + e^{\hat{y}_i + \gamma}) - y_i(\hat{y}_i + \gamma)$$

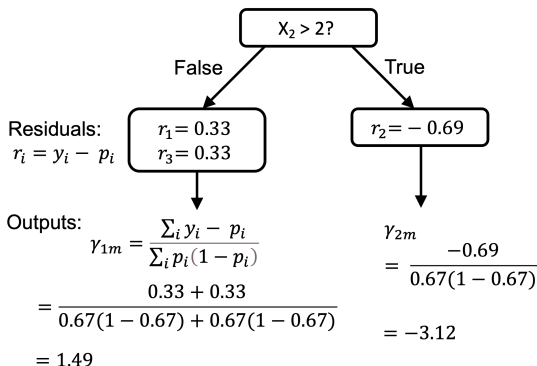
and

$$\gamma_k = \frac{\sum_i (y_i - p_i)}{\sum_i p_i(1 - p_i)}$$


## Gradient Boosting dla klasyfikacji binarnej

	Feature $x_1$	Feature $x_2$	Class label $y$
1	1.12	1.4	1
2	2.45	2.1	0
3	3.54	1.2	1

	Feature $x_1$	Feature $x_2$	Class label $y$	Step 1: $\hat{y} = \log(\text{odds})$	Step 2A: $p = \frac{1}{1 + e^{-\hat{y}}}$	Step 2A: $r = y - p$
1	1.12	1.4	1	0.69	0.67	0.33
2	2.45	2.1	0	0.69	0.67	-0.67
3	3.54	1.2	1	0.69	0.67	0.33

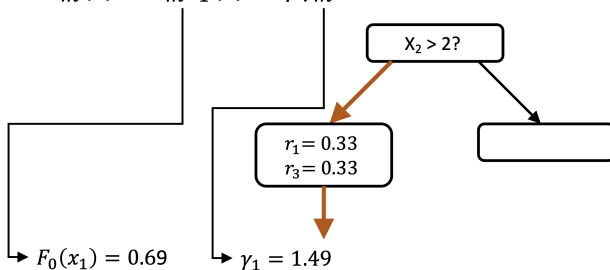


# Gradient Boosting dla klasyfikacji binarnej



	Feature $x_1$	Feature $x_2$	Class label $y$
1	1.12	1.4	1
2	2.45	2.1	0
3	3.54	1.2	1

**Step 2D:**  $F_m(x) = F_{m-1}(x) + \eta \gamma_m$



$$F_1(x_1) = 0.69 + 0.1 \times 1.49 = 0.839$$

# Gradient Boosting dla klasyfikacji binarnej

$\mathbf{x_1 \quad x_2 \quad y}$				Step 1: $F_0(x) = \hat{y}$ $= \text{log(odds)}$	Step 2A: $p = \frac{1}{1 + e^{-\hat{y}}}$	Step 2A: $r = y - p$	New log(odds) $\hat{y} = F_1(x)$	Step 2A: $p$	Step 2A: $r$
1	1.12	1.4	1	0.69	0.67	0.33	0.839	0.698	0.302
2	2.45	2.1	0	0.69	0.67	-0.67	0.378	0.593	-0.593
3	3.54	1.2	1	0.69	0.67	0.33	0.839	0.698	0.302

Round  $m = 1$

Round  $m = 2$

## Advantages of the GBM model

- ① flexibility - it can be tuned with different parameters;
- ② handles missing data - imputation is not required;
- ③ uses gradient methods.



- **XGBoost (Extreme Gradient Boosting)**, is the most popular boosting technique in recent times.
- XGBoost is a version of the algorithm **gbm**
- XGBoost has regularization introduced in the objective function, which **penalizes the model when it gets more complicated** in a training iteration.

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

- $L(\theta)$  - cost(loss) function on the training set - evaluates the predicted values calculated on the training data;
- $\Omega(\theta)$  - regularization function - controls the complexity of the model to avoid model overfitting.
- MSE ( mean squared error)

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

- loss function for logistic regression

$$L(\theta) = \sum_i \left( y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i}) \right)$$

The main advantages:

- **parallel computing** - this package is enabled with parallel processing using OpenMP, which then uses all the cores of the computing machine;
- **regularization** - this helps in circumventing the problem of overfitting by incorporating the regularization ideas;
- **cross-validation** - no extra coding is required for carrying out cross-validation;
- **pruning** - this grows the tree up to the maximum depth and then prunes backward;
- **missing values** - missing values are internally handled
- **saving and reloading** - this has features that not only help in saving an existing model, but can also continue the iterations from the step where it was last stopped
- **cross platform** - this is available for Python, Scala, and so on

Thank you for your attention!!!