Machine Learning

Ensemble methods

Karol Przystalski

April 8, 2020

Department of Information Technologies, Jagiellonian University

Agenda

1. Introduction
2. Bagging
3. Boosting
4. Stacking
5. Random forest
6. xgboost

Introduction

Ensemble methods

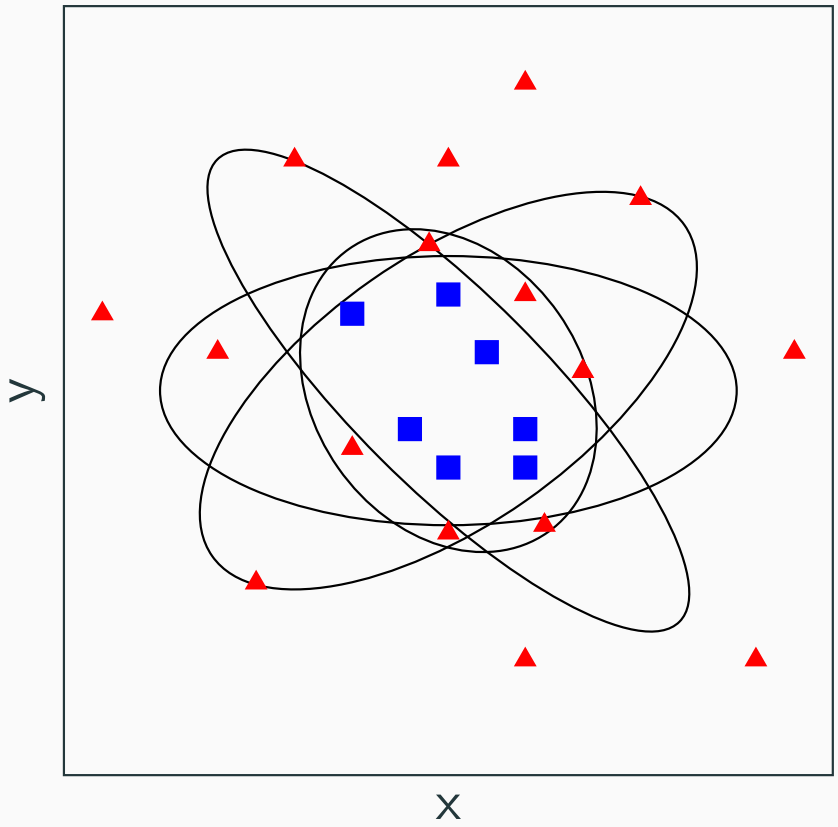
Ensemble methods are about combining classifiers to get better results than each classifier on its own. The classifiers are built on the same data sets. Depending on the way how an ensemble method is built, we have few types:

- boosting,
- bagging,
- arcing,
- grading,
- stacking,
- other ....

Combining identical classifiers is useless!

Ensemble methods – overview

A comparison of one classifier against many classifiers.



Ensemble methods – formulas

A general formula of ensemble methods looks like following:

$$\bar{C}(X) = \sum_{i=1}^{T} w_i C_i(X). \tag{1}$$

Ensemble methods – more

Collecting many poor classifiers into one ensemble classifier can result in a classifier that performs well.

In other words, we need many low-quality classifiers and a way to combine them together, so we get a classifier that do very well.

Each classifier needs to be better than guessing.

Ensemble methods are also known as combined methods.

Bagging

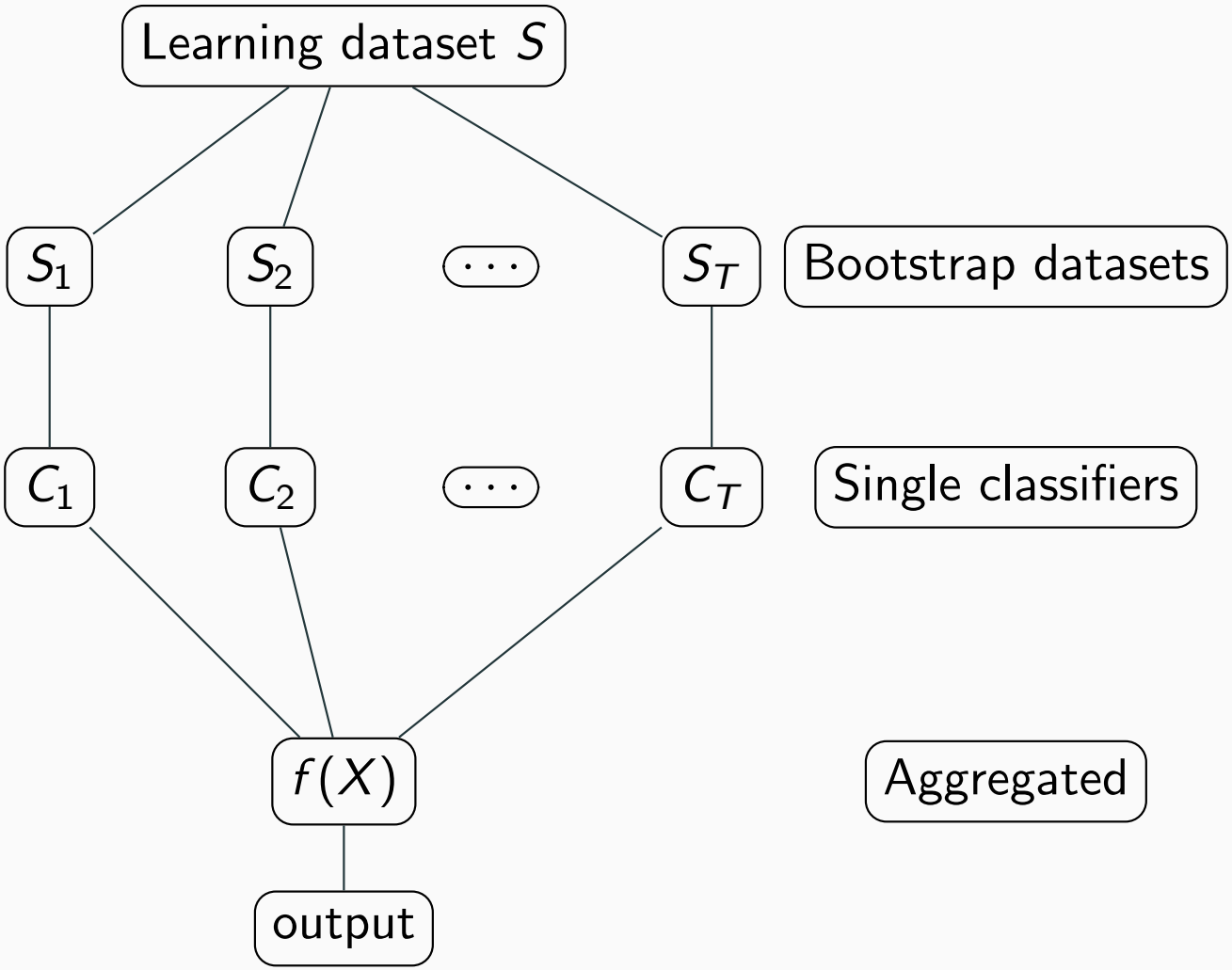
Bagging – aggregate bootstrap

Bagging is a short name for bootstrap aggregation. Generates individual classifiers on bootstrap samples of the training set.

A bootstrap sample is a sample taken from the original dataset with replacement, so that we may get some data several times and others not at all. The bootstrap sample is the same size as the original dataset.

Bagging traditionally uses component classifiers of the same type and combines prediction by a simple majority voting across.

Bagging – overview



Bagging – steps

1. create T bootstrap samples S_i,
2. for each sample S_i train a classifier,
3. vote:

$$f(x) = \arg\max \sum_{i}^{T} (f_i(X) = y) \tag{2}$$

Boosting

Boosting

Boosting methods take a different weighting schema of resampling than bagging

The component classifiers are built sequentially, and examples that are misclassified by previous components are chosen more often than those that are correctly classified.

So, new classifiers are influenced by performance of previously built ones. New classifier is encouraged to become expert for instances classified incorrectly by earlier classifier.

There are few methods of boosting type:

- AdaBoost,
- Arcing,
- RegionBoost,
- Stumping.

AdaBoost – steps

1. initialize weights to $\frac{1}{N}$, where N is the number of datapoints,
2. loop until
$$\varepsilon_t < \frac{1}{2} \tag{3}$$
or maximum number of iteration is reached,
3. train classifier on $S, w^{(t)}$ and get a hypothesis $h_t(x_n)$ for datapoints x_n,
4. compute error
$$\varepsilon_t = \sum_{n=1}^{N} w_n^{(t)} I(y_n \neq h_t(x_n)), \tag{4}$$
5. set
$$\alpha_t = \log(\frac{1-\varepsilon_t}{\varepsilon_t}), \tag{5}$$

AdaBoost – steps

6. update weights:

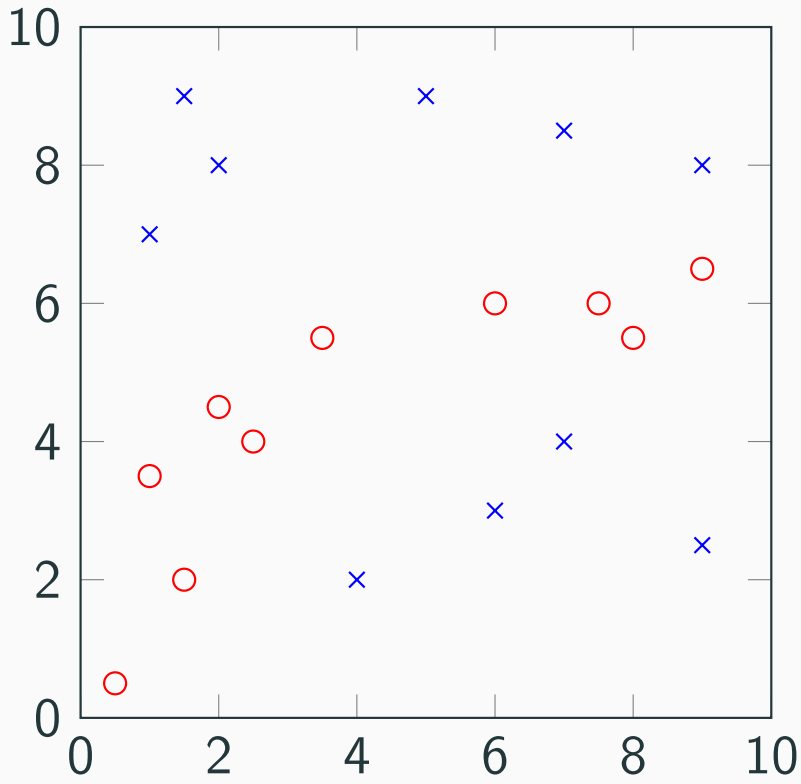
$$w_n^{(t+1)} = \frac{w_n^{(t)} \exp \alpha_t I(y_n \neq h_t(x_n))}{Z_t}, \tag{6}$$

   where Z_t is a normalization constant,

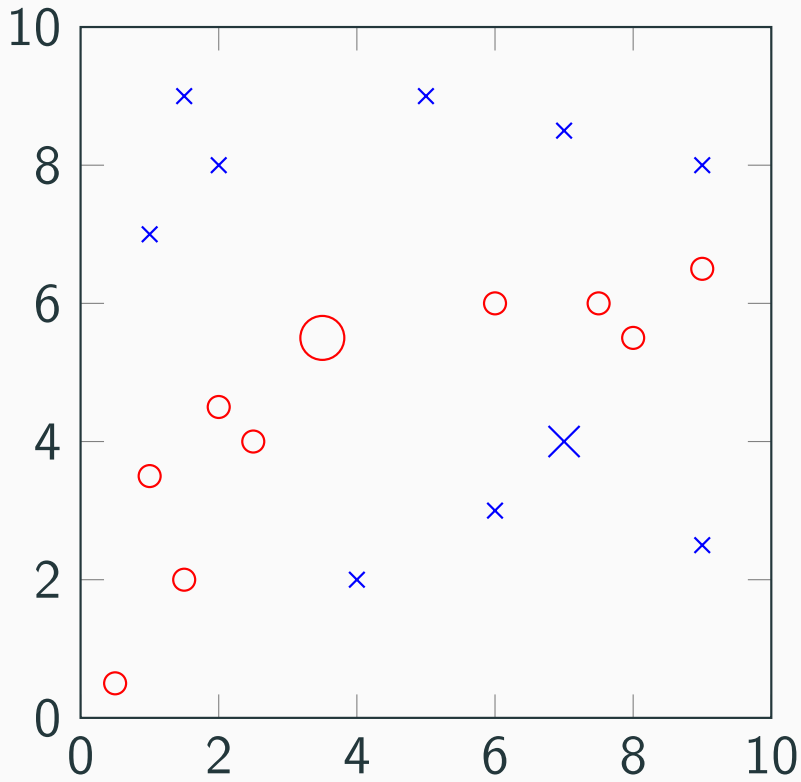
7. output

$$f(X) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{7}$$

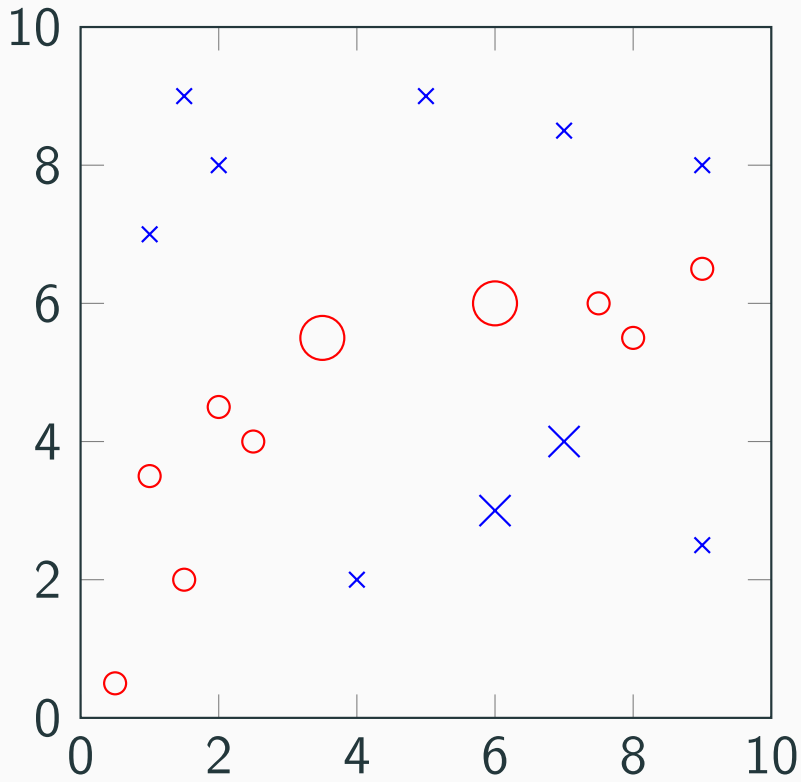
AdaBoost – overview



AdaBoost – overview



AdaBoost – overview



Arcing – main concept

Arcing is similar to AdaBoost, but it differs how it calculates the error:

$$\varepsilon_{(t+1)}(i) = \frac{1}{Z} \cdot (1 + \sum_{n=1}^{N} I(y_n \neq h_t(x_n))) \tag{8}$$

Finally, we vote for the label:

$$f(X) = \arg\max_i \sum_{i}^{T} (f_i(X) = y). \tag{9}$$

RegionBoost

It is a modification of AdaBoost. The difference is that the weights of each object depends locally on the importance of other k closest neighbourhood objects:

$$w_i(x_i) = \frac{1}{T} \sum_{i=1}^{T} kNN(K, C_i, x_i, y_i), \tag{10}$$

where

$$kNN(K, C_i, x_i, y_i) = \frac{1}{K} [\sum_{x_s \in N(K,X)} I(f(x_s = y_s))] \tag{11}$$

Stumping

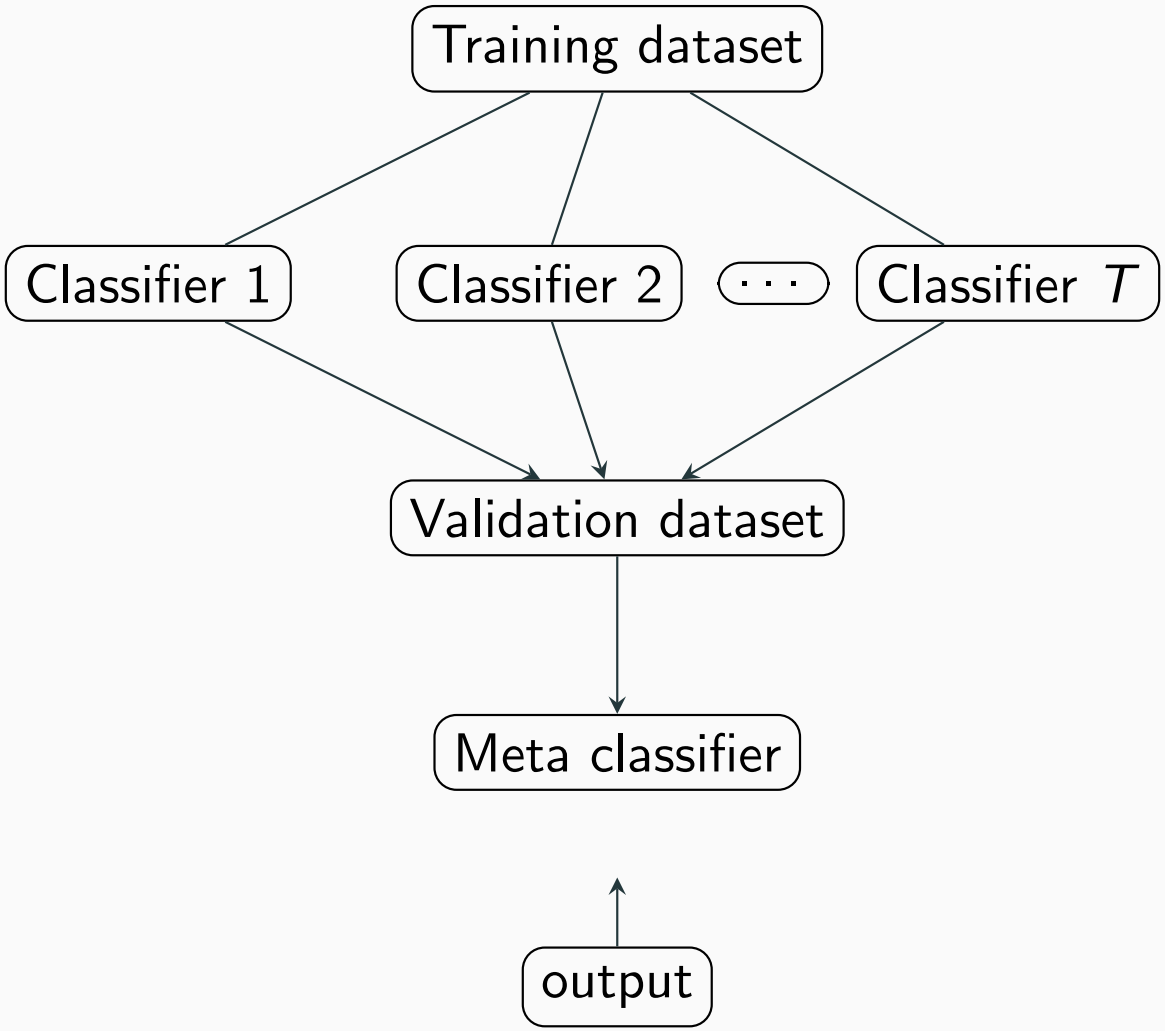
It's a type of boosting method that is applied to trees. A stump of a tree is a tine piece that is left over when you chop off the rest.

Stumping consists of simply taking the root of the tree and using that as the decision maker.

For each classifier you use the very first question that makes up the root of the tree and that is it.

Stacking

Stacking – main concept



Stacking – steps (simplified)

We have the following steps:

1. create T classifiers and learn each to get m predictions (hypothesis h_t,
2. construct data set of predictions and construct a new classifier C_m for each dataset,
3. construct a C_h classifier that combines all C_m classifiers.

Stacking – prediction matrix

Predictions				
C_1	C_2	C_3	C_T	$\bar{C}$
1	1	0	1	1
0	0	0	1	0
...	...	...	...	
0	1	1	1	1

Grading

Grading is similar to stacking, but the main difference is in the way how the data for the meta classifier is given. For stacking we had the data as following:

Predictions				
C_1	C_2	C_3	C_T	**Label**
1	1	0	1	1
0	0	0	1	0
...	...	...	...	
0	1	1	1	1

Grading

For grading we have the training set as following:

Attributes				
x_1	x_2	x_3	x_n	**Graded predictions**
0.2	0.5	-0.1	0.4	+
-0.1	0.15	-0.7	0.5	+
...	...	...	...	-
0.8	0.2	-0.24	0.6	+

Graded predictions are the values if a prediction of a given classifier is done properly or not.

Random forest

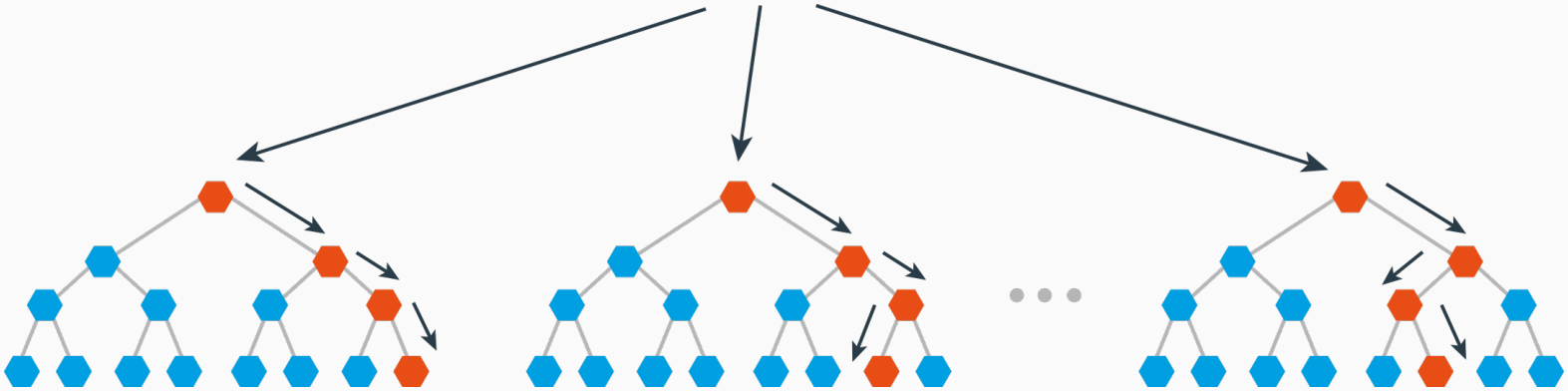
Random forest

This method is similar to boosting, but this method tells more.

Boosting test on the whole feature set at each stage. It means that boosting is running sequential while random forest can work in parallel.

Since the algorithm only searched a small subset of the data at each stage, it cannot be expected to be as good as boosting for the same number of trees. However, since the trees are cheaper to train, we can make more of them in the same computational time.

Random forest



Random forest – steps

1. for each tree of N we create a new bootstrap dataset and train it,
2. at each node of the decision tree, randomly select m features, and compute the information gain only on that set of features, selecting the optimal one,
3. repeat until the tree is complete.

xgboost

xgboost

In a few words it can summarized as a AdaBoost modification where the loos function is calculated with gradient descent algorithm. Xgboost stands for extreme gradient boosting decision trees. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

Questions?