Machine Learning

Ensemble methods

Karol Przystalski

April 9, 2025

Department of Information Technologies, Jagiellonian University

Agenda

1. Introduction
2. Bagging
3. Boosting
4. Quality metrics
5. Stacking
6. Random forest
7. xgboost
8. Miscellaneous

Introduction

Ensemble methods

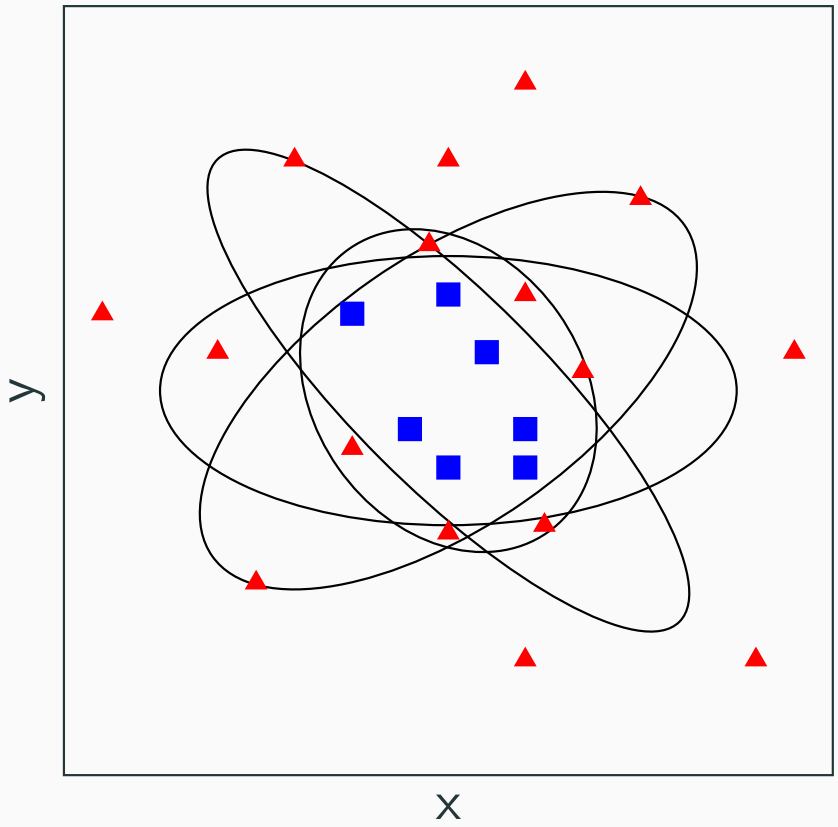
Ensemble methods are about combining classifiers to get better results than each classifier on its own. The classifiers are built on the same data sets. Depending on the way how an ensemble method is built, we have few types:

- 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.

Ensemble classifiers taxonomy

We can divide the ensemble methods by:

- combining strategy,
- optimization strategy,
- trainability.

Fusion and selection

We have two main strategies:

- fusion – each ensemble member is supposed to have knowledge of the whole feature space,
- selection – each ensemble member is supposed to know well a part of the feature space and be responsible for objects in this part.

Synonims:

- fusion – selection,
- competitive classifiers – cooperative classifiers,
- ensemble approach – modular approach,
- multiple topology – hybrid topology.

Fusion and selection

In the fusion approach we apply combiners such as the average or majority vote.

In selection approach we usually select one classifier to label the input.

Between fusion and selection

Some methods are between the two strategies. For example, if a classifier is taking the average of the outputs with coefficients that depends on the input. Then more than one classifier is responsible for the given input.

Bagging

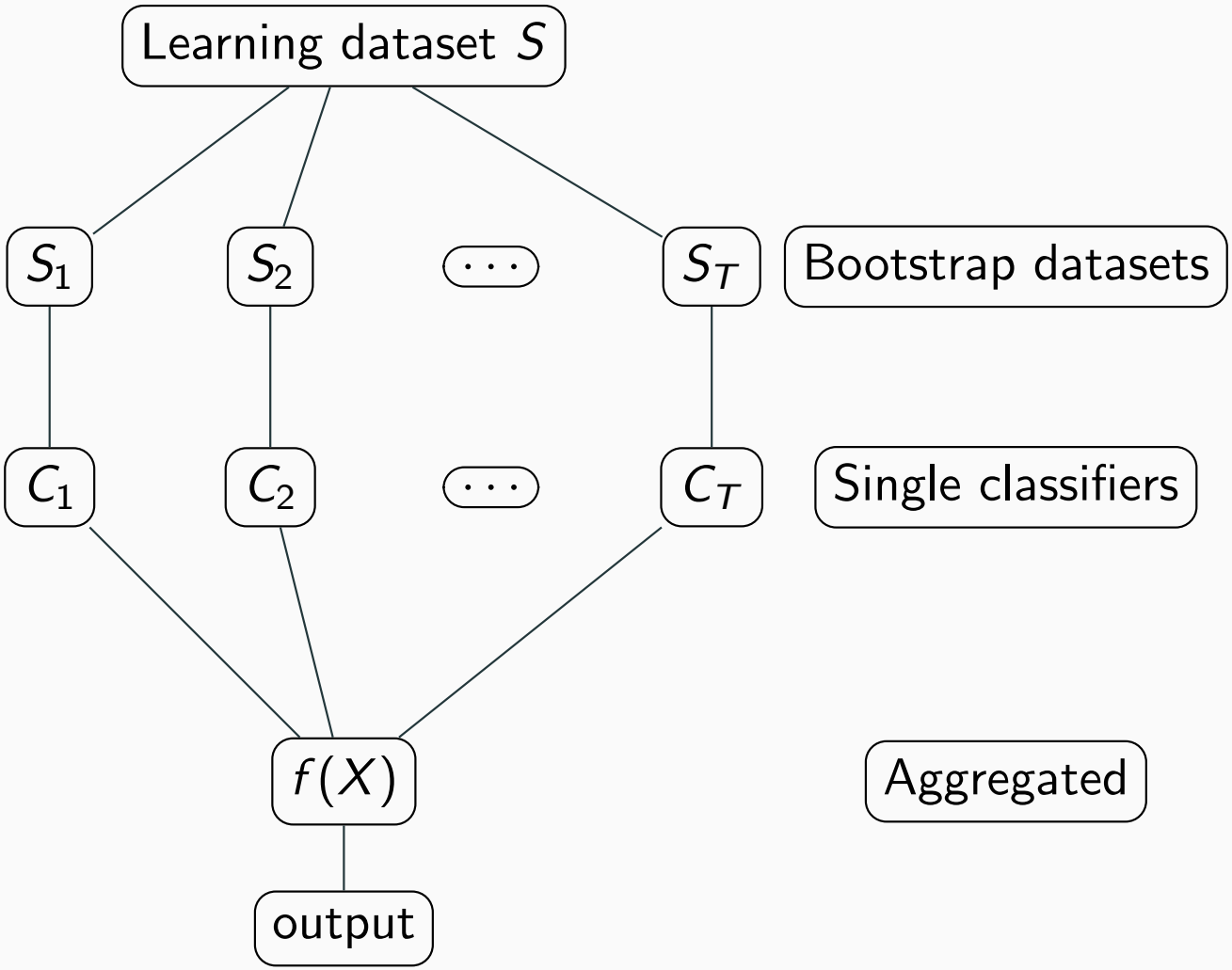
Bagging – aggregate bootstrap

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:

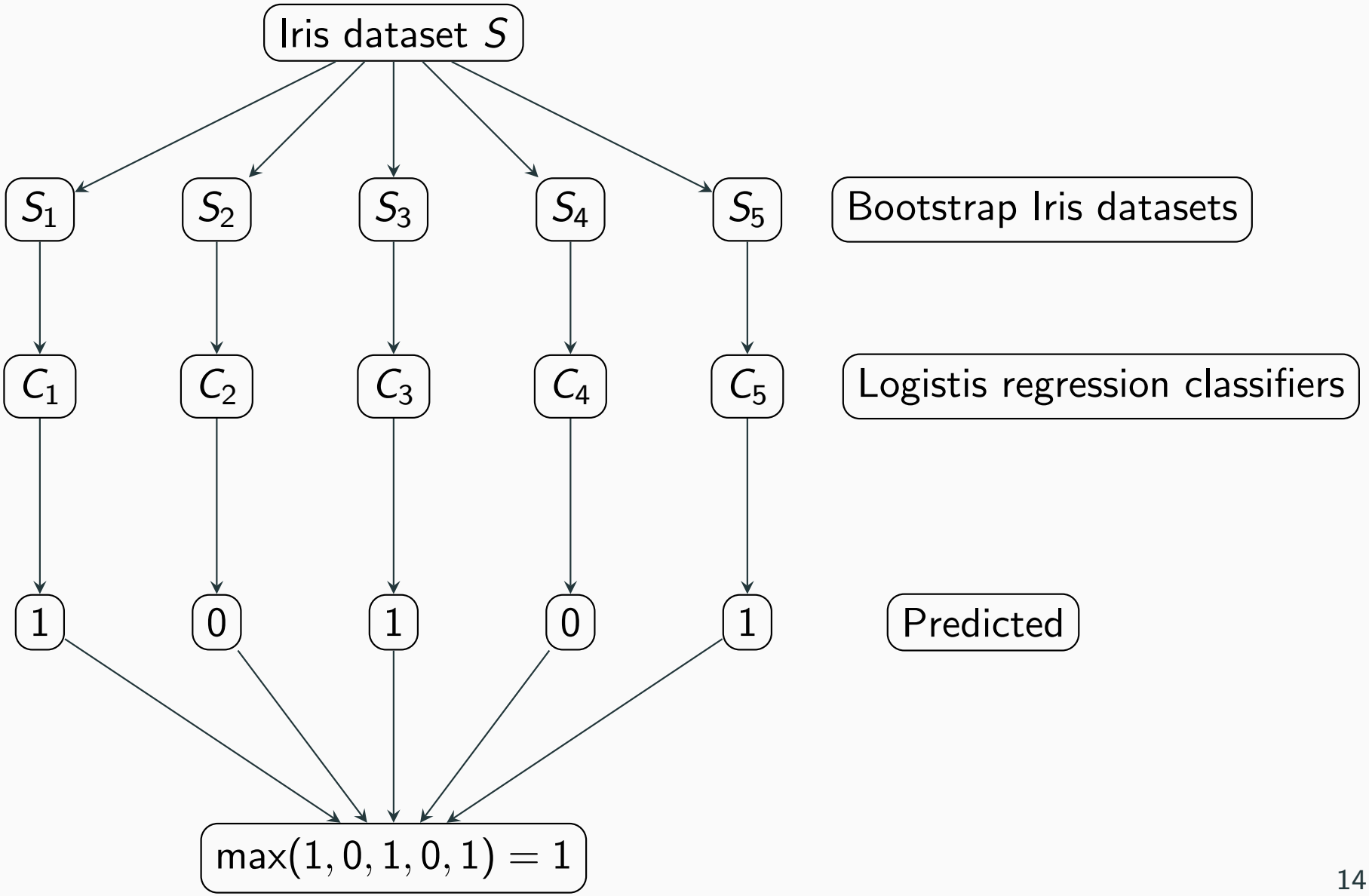
$$f(x) = \arg\max \sum_{i}^{T} (f_i(X) = y) \tag{2}$$

Bagging – steps

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

$$f(x) = \arg\max \sum_{i}^{T} (f_i(X) = y) \tag{3}$$

Bagging – example



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
$$\varepsilon_t < \frac{1}{2} \tag{4}$$
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
$$\varepsilon_t = \sum_{n=1}^{N} w_n^{(t)} I(y_n \neq h_t(x_n)), \tag{5}$$
5. set
$$\alpha_t = \log(\frac{1-\varepsilon_t}{\varepsilon_t}), \tag{6}$$

AdaBoost – steps

6. update weights:

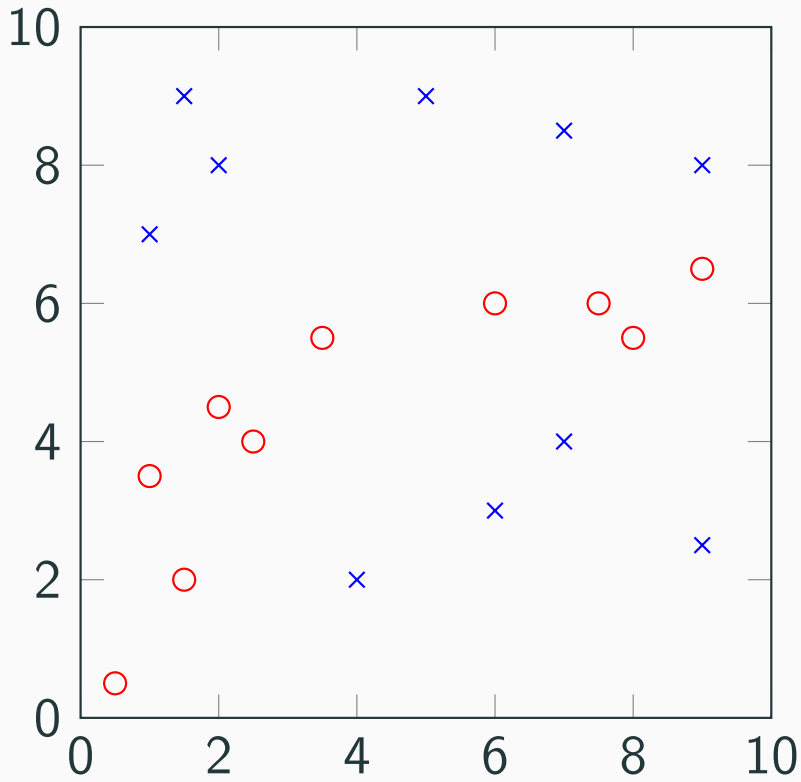
$$w_n^{(t+1)} = \frac{w_n^{(t)} \exp \alpha_t I(y_n \neq h_t(x_n))}{Z_t}, \tag{7}$$

   where Z_t is a normalization constant,

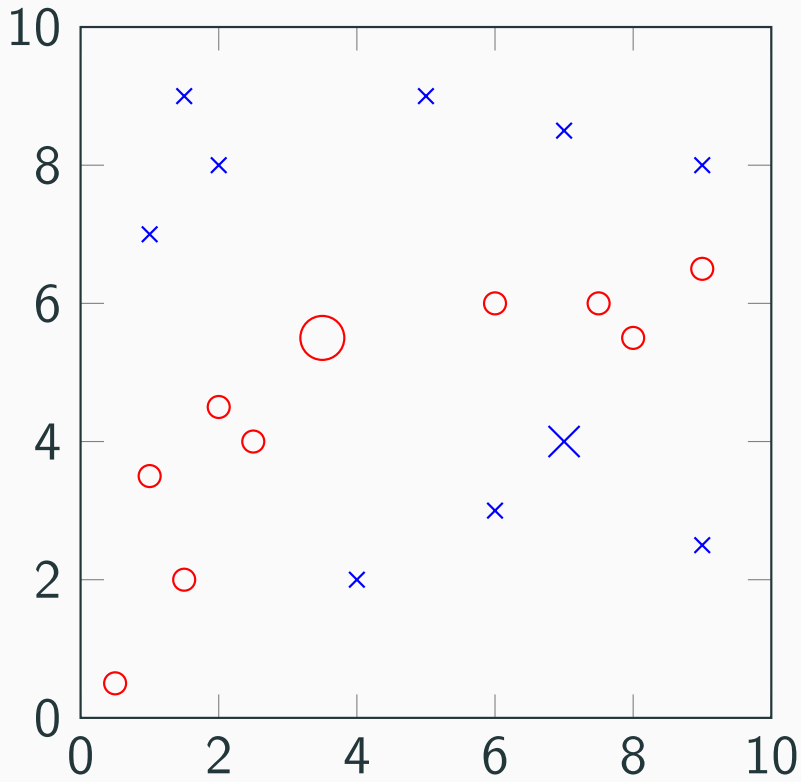
7. output

$$f(X) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{8}$$

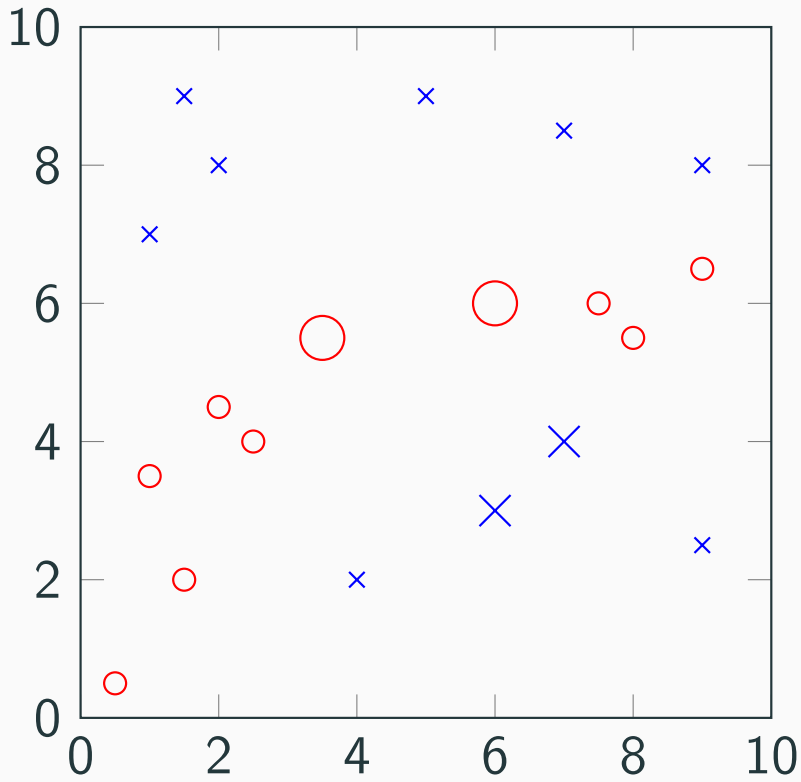
AdaBoost – overview



AdaBoost – overview



AdaBoost – overview



Arcing – main concept

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

$$\varepsilon_{(t+1)}(i) = \frac{1}{Z} \cdot (1 + \sum_{n=1}^{N} I(y_n \neq h_t(x_n))) \tag{9}$$

Finally, we vote for the label:

$$f(X) = \arg\max_i \sum_{i}^{T} (f_i(X) = y). \tag{10}$$

A comparison between AdaBoost and Arcing can be found in R. Khanchel and M. Limam, *Empirical comparison of Arcing algorithms*, 2000.

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:

$$w_i(x_i) = \frac{1}{T} \sum_{i=1}^{T} kNN(K, C_i, x_i, y_i), \tag{11}$$

where

$$kNN(K, C_i, x_i, y_i) = \frac{1}{K} [\sum_{x_s \in N(K,X)} I(f(x_s = y_s))] \tag{12}$$

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 (single decision tree).

Quality metrics

Boosting vs. bagging

Similarities:

- data sampling,
- both use voting,
- build N classifiers.

Differences:

- bagging use same classification methods,
- bagging use same weights for classifier,
- bagging consists of independent models.

Boosting vs. bagging

Based on Kuncheva, the general consensus is that boosting reaches lower testing error. Boosting algorithms have been crowned as the "most accurate available off-the-shelf classifiers on a wide variety of data sets". However, it seems that they are sensitive to noise and outliers, especially for small data sets. Random forest (bagging) have been found to be comparable to AdaBoost.

Voting limitation

The majority vote does not guarantee to do better than a single member of the combine method. There are two scenarios know as "pattern of success" and "pattern of failure". Let's assume that we have 10 objects and each classifier predicts correctly 6 of 10 objects. The accuracy of a single classifier is equal to 60%.

Pattern of success

Classifier	111	101	011	001	110	100	010	000	P_{maj}	$P_{maj} - p$
1	0	3	3	0	3	0	0	1	0.9	0.3
2	2	2	2	0	2	0	0	2	0.8	0.2
3	1	2	2	1	3	0	0	1	0.8	0.2
4	0	2	3	1	3	1	0	0	0.8	0.2
5	0	2	2	2	4	0	0	0	0.8	0.2
6	4	1	1	0	1	0	0	3	0.7	0.1
7	3	1	1	1	2	0	0	2	0.7	0.1
8	2	1	2	1	2	1	0	1	0.7	0.1
9	2	1	1	2	3	0	0	1	0.7	0.1
10	1	2	2	1	2	1	1	0	0.7	0.1
11	1	1	2	2	3	1	0	0	0.7	0.1
12	1	1	1	3	4	0	0	0	0.7	0.1
13	6	0	0	0	0	0	0	4	0.6	0.0
14	5	0	0	1	1	0	0	3	0.6	0.0

Pattern of failure

Classifier	111	101	011	001	110	100	010	000	P_{maj}	$P_{maj} - p$
15	4	0	1	1	1	1	0	2	0.6	0.0
16	4	0	0	2	2	0	0	2	0.6	0.0
17	3	1	1	1	1	1	1	1	0.6	0.1
18	3	0	1	2	2	1	0	1	0.6	0.0
19	3	0	0	3	3	0	0	1	0.6	0.0
20	2	1	1	2	2	1	1	0	0.6	0.0
21	2	0	2	2	2	2	0	0	0.6	0.0
22	2	0	1	3	3	1	0	0	0.6	0.0
23	2	0	0	4	4	0	0	0	0.6	0.0
24	5	0	0	1	0	1	1	2	0.5	-0.1
25	4	0	0	2	1	1	1	1	0.5	-0.1
26	3	0	1	2	1	2	1	0	0.5	-0.1
26	3	0	1	2	1	2	1	0	0.5	-0.1
27	3	0	0	3	2	1	1	0	0.5	-0.1
28	4	0	0	2	0	2	2	0	0.4	-0.2

Weighted voting

Let's assume we have five classifiers with different accuracies: $(0.9, 0.9, 0.6, 0.6, 0.6)$. The majority vote for the five independent classifers is:

$$P_{maj} = 3 * 0.9^2 * 0.4 * 0.6 + 0.6^3 + 6 * 0.9 * 0.1 * 0.6^2 * 0.4 \approx 0.877$$

If we set different weights for each classifier like: $(\frac{1}{3}, \frac{1}{3}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9})$ we get:

$$P^w_{maj} = 0.9^2 + 2 * 3 * 0.9 * 0.1 * 0.6^2 * 0.4 + 2 * 0.9 * 0.1 * 0.6^3 \approx 0.927.$$

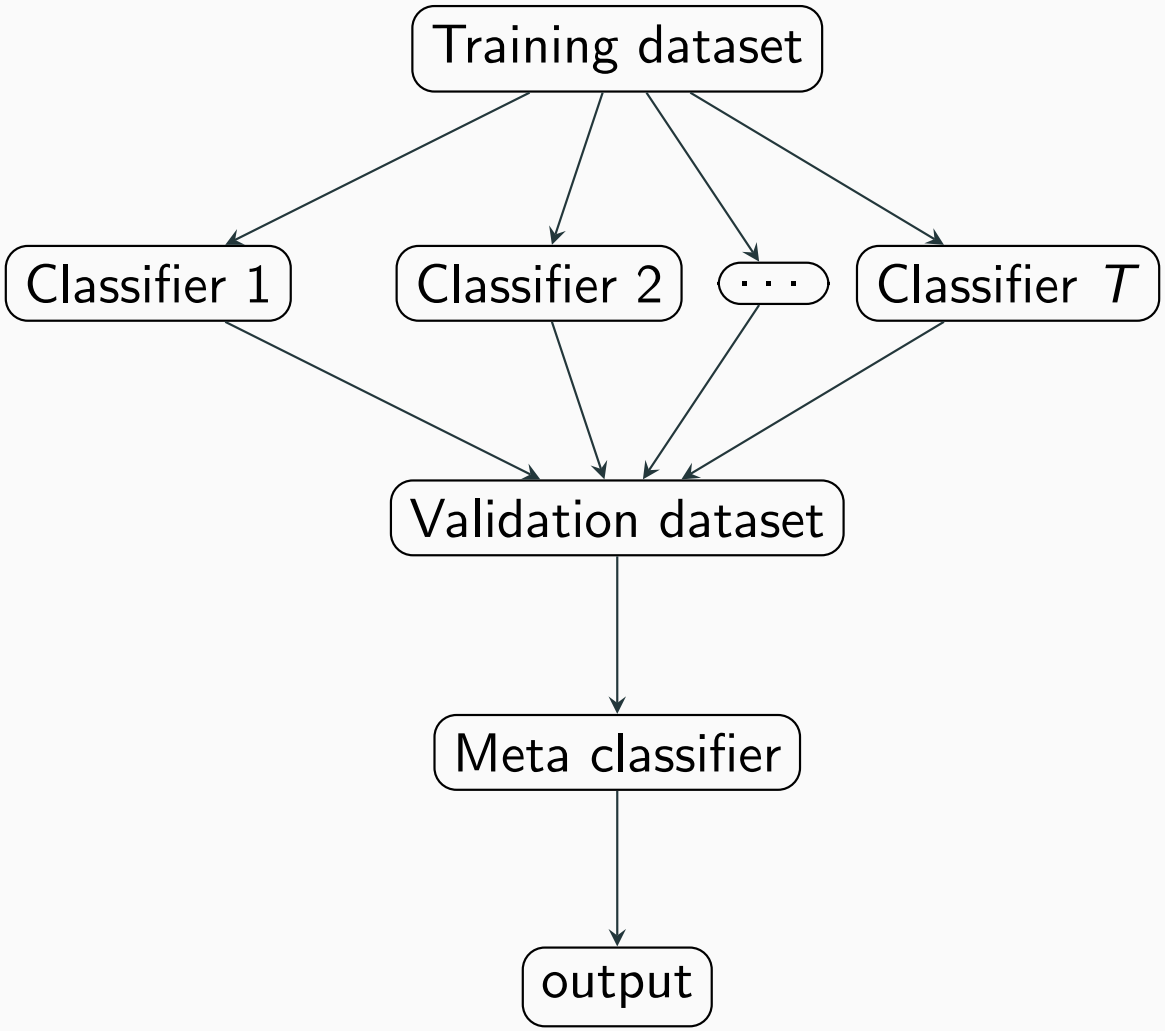
Other methods used in fusion approach

Apart from majority voting and weighted majority voting we have also other methods that can be used. The most popular are:

- Naive Bayes,
- Behavior knowledge space (BKS),
- Wernecke,
- First-order dependence tree,
- SVD (Single Value Decomposition) combination.

Stacking

Stacking – main concept

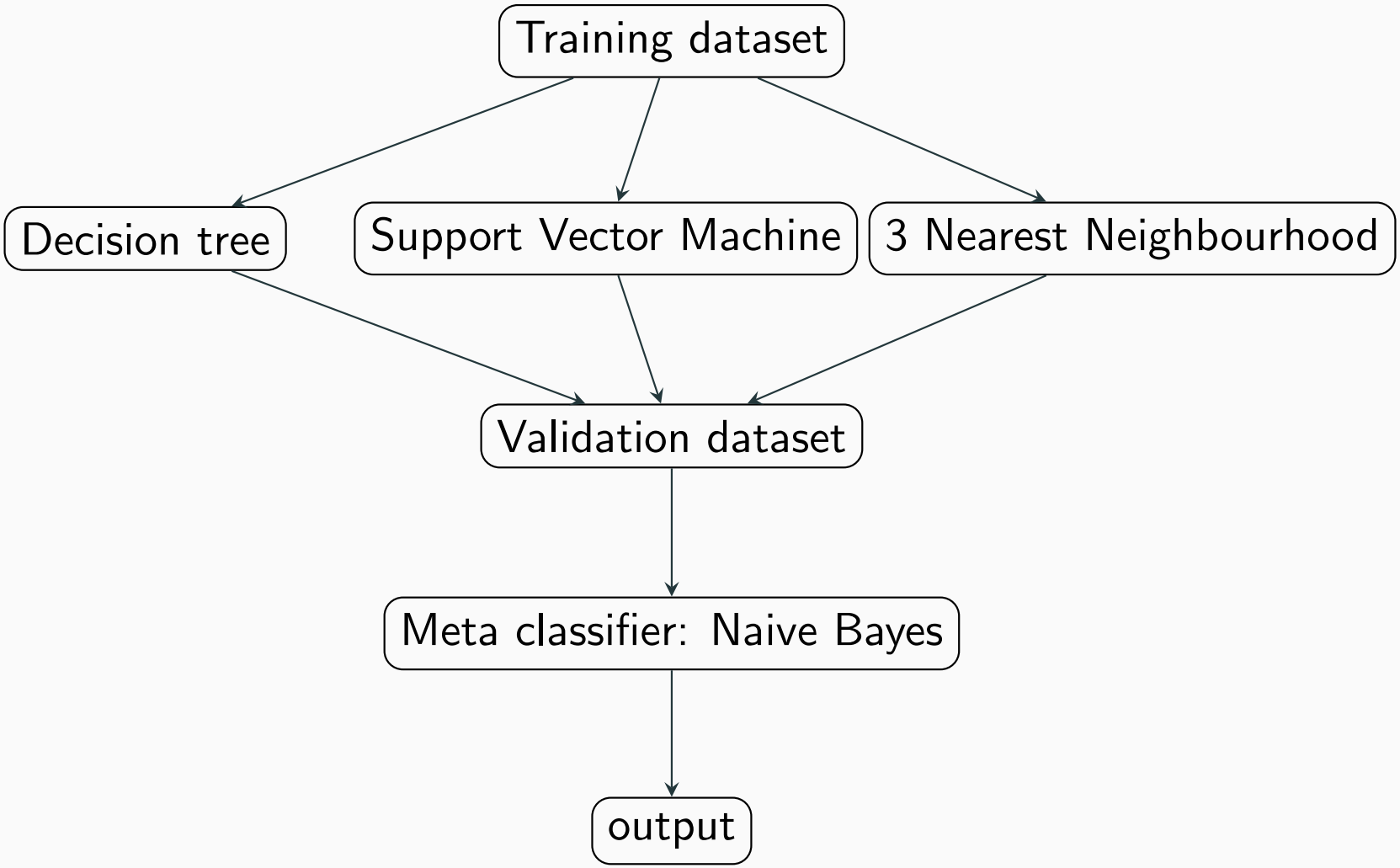


Stacking – steps (simplified)

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 – example



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.

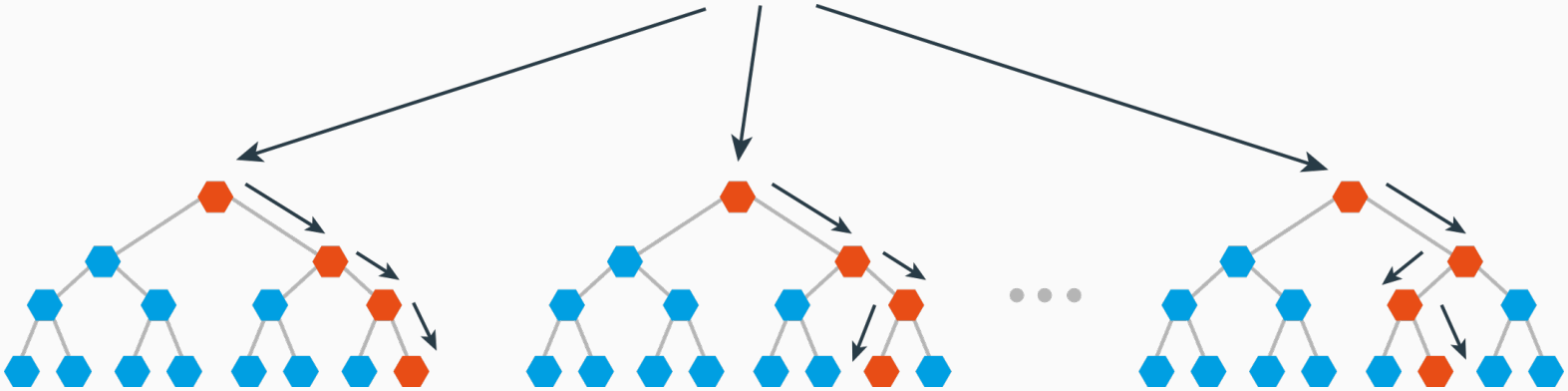
Random forest

Random forest

This method is similar to boosting. 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

The algorithm consists of the following 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 deep 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.

xgboost – features

Based on the original paper: T. Chen and C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, 2016, the classifier has a few important features that differs it from the other ensemble methods. This includes:

- regularized learning,
- gradient tree boosting,
- shrinkage and column subsampling.

Regularized learning

As in linear regression, we can penalize the classifiers and regularize the bias and variance.

$$L(\omega) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \tag{13}$$

where

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2. \tag{14}$$

The l is a differentiable convex loss function that measures the difference between prediction $\hat{y}_i$ and the target y_i. The Ω penalizes the complexity of the model. The λ helps to smooth the final learnt weights to avoid overfitting. T is the number of leaves in the tree, f_k corresponds to and independent tree structure.

Gradient tree boosting

We train the ensemble model using the gradient descent method. The model is trained in the addictive manner. It means that we add a f_k that suits better (greedly addition). We want to optimize the following objective:

$$L^{(t)} = \sum_i l(y_i, \hat{y}_i^{(t-1)}(f_t(x_i)) + \Omega(f_t), \tag{15}$$

where the second-order approximation can be used to quickly optimize the objective:

$$L^{(t)} \simeq \sum_i [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i] + \Omega(f_t), \tag{16}$$

where

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}^{(t-1)}, \tag{17}$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}^{(t-1)}. \tag{18}$$

Shrinkage and column subsampling

Beside the regularized objective two additional techniques are used to further prevent overfitting

Shrinkage scales newly added weights by a factor η after each step of tree boosting (similar to learning rate).

Column subsampling is the same technique that is used in random forest method.

xgboost comparison

Method	xgboost	pGBRT	Spark MLLib	H2O	scikit-learn	R GBM
Exact greedy	yes	no	no	no	yes	yes
Approximate global	yes	no	yes	yes	no	no
Approximate loacal	yes	yes	no	no	no	no
Out-of-core	yes	no	no	no	no	no
Sparsity aware	yes	no	partially	partially	no	partially
Parallel	yes	yes	yes	yes	no	no

Based on the original from 2016

Miscellaneous

Evaluate clustering methods – Model explorer algorithm

The algorithm is divided into steps:

1. pick the number of clusters c, the number of pairs of the subsamples L, and a similiarity measure $S(A, B)$ between two partitions A and B. Specify the proportion of objects f, to be sampled from Z without replacement.
2. Generate two subsamples from Z, S_1 and S_2, of size N each.
3. Cluster S_1 into c clusters; denote this partition by $A(S_1)$.
4. Cluster S_2 into c clusters; denote this partition by $B(S_2)$.
5. Find the objects present in both subsamples, i.e., $S_{12} = S_1 \cap S_2$.
6. Calculate and store the similarity between the restricted partitions $S(A(S_{12}), B(S_{12}))$.
7. Repeat 2–6 L times.

Cluster ensembles

Use cases of cluster ensembles:

- Knowledge reuse,
- Distributed clustering.

A typical cluster ensemble algorithm consists of:

1. Given is a data set Z with N elements. Pick the ensemble size L and the number of clusters c.
2. Generate L partitions of Z in c clusters.
3. Form a co-association matrix of each partition, $M^{(k)} = m_{ij}^{(k)}$, of size $N \times N, k = 1, \ldots, L$, where:

$$m_{ij} = \begin{cases} 1, & \text{if } z_i \text{ and } z_j \text{ are in the same cluster in partition } k, \\ 0, & \text{if } z_i \text{ and } z_j \text{ are in different cluster in partition } k \end{cases}.$$

4. Form a final co-association matrix M (consensus matrix) from $M^{(k)}, k = 1, \ldots, L$, and derive the final clustering using this matrix.

Questions?