Multi-classifiers

Department of Computer Languages and Systems

Terminologies

- Lots of terms are used to refer to multi-classifiers:
  - ensemble of classifiers
  - combining classifiers
  - decision committee
  - multiple classifier system
  - mixture of experts
  - committee-based learning
  - etc.

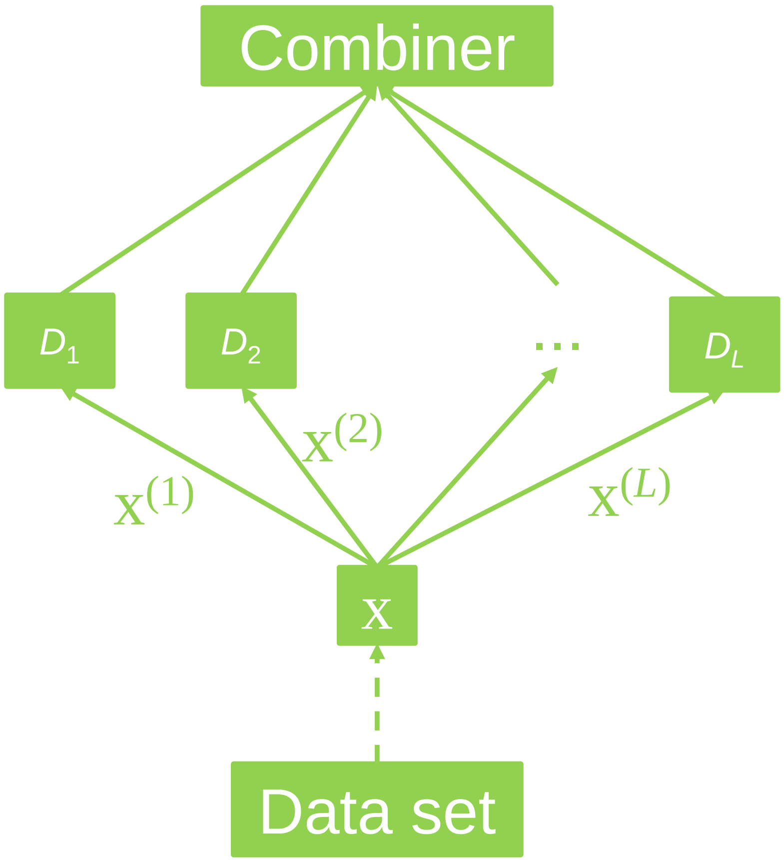
Introduction: motivation

- When you have to face a complex classification problem:
  - which learning algorithm to use?
  - which parameters to choose?
  - how to use the training data?
  - which vector space to map the data onto? What is the most discriminating representation?

Introduction: motivation

- Different models may appear while searching for a solution, but often none of them is better than the rest
  - In this case, a reasonable choice is to keep them all and create a final system integrating the pieces
  - The core idea behind this is to aggregate multiple models to obtain a combined model D that outperforms every single model D_i in it
  - Each single model D_i is called base learner (classifier) or individual learner (classifier)

Strategies to build a multi-classifier



- Combination level: design different combiners
- Classifier level: use different base classifiers
- Data level: use different data subsets
- Feature level: use different feature subsets

Source: http://www.ccas.ru/voron/download/books/machlearn/kuncheva04combining.pdf

Combination level: fusion vs. selection

Fusion

- each ensemble member is supposed to have knowledge of the whole feature space
- some combiner such as the average and majority vote is applied to label the input object x

Selection

- each ensemble member is supposed to know well a part of the feature space and to be responsible for objects in this part
- one member is chosen to label the input object x

Combination level (ii): fusion vs. selection

Fusion

- competitive classifiers
- ensemble approach
- multiple topology

Selection

- cooperative classifiers
- modular approach
- hybrid topology

Fusion: Majority vote

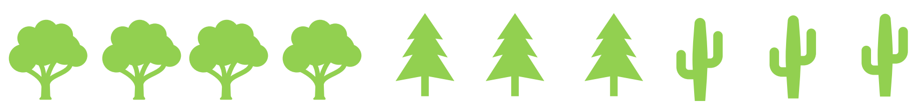
Decision rule: to choose the class most voted by the base classifiers

Three consensus patterns:

- Unanimity (all agree)

- Simple majority (50%+1)

- Plurality (most votes)



Fusion: Majority vote (ii)

Let it be

- $\left[d_{i,1}, \dots, d_{i,C}\right]^{\mathrm{T}} \in \{0,1\}^{C}, i = 1, \dots, L$, where $d_{i,j} = 1$ if D_i labels x in class ω_j, and 0 otherwise

Then, the plurality vote rule will result in an ensemble decision for class ω_k if

$$\sum_{i=1}^{L} d_{i,k} = \max_{j=1,\dots,c} \sum_{i=1}^{L} d_{i,j}$$

This rule coincides with the simple majority rule if $C = 2$

Fusion: Majority vote (iii)

A thresholded plurality vote: we increase the set of classes with one more class ω_{c+1}, for objects for which the ensemble does not determine a class label with a sufficient confidence. Now, the decision is

$$\begin{cases} \omega_k, & \text{if } \sum_{i=1}^{L} d_{i,k} \geq \alpha \cdot L \\ \omega_{c+1}, & \text{otherwise} \end{cases}$$

where $0 < \alpha \leq 1$. If $\alpha = 1$, this becomes the unanimity vote rule

Fusion: Majority vote (iv)

Weighted majority vote:

- an adequate option when the base classifiers are not of very similar accuracy
- it attempts to give the more competent classifiers more power in making the final decision

Fusion: Majority vote (v)

Weighted majority vote:

- we can represent the outputs as

$$d_{i,j} = \begin{cases} 1 & \text{if } D_i \text{ labels x in } \omega_j \\ 0 & \text{otherwise} \end{cases}$$

- then, the decision is ω_k if

$$\sum_{i=1}^{L} w_i d_{i,k} = \max_{j=1,\dots,c} \sum_{i=1}^{L} w_i d_{i,j}$$

where $w_i \geq 0$ ($\sum_{i=1}^{c} w_i = 1$) is a weight for classifier D_i

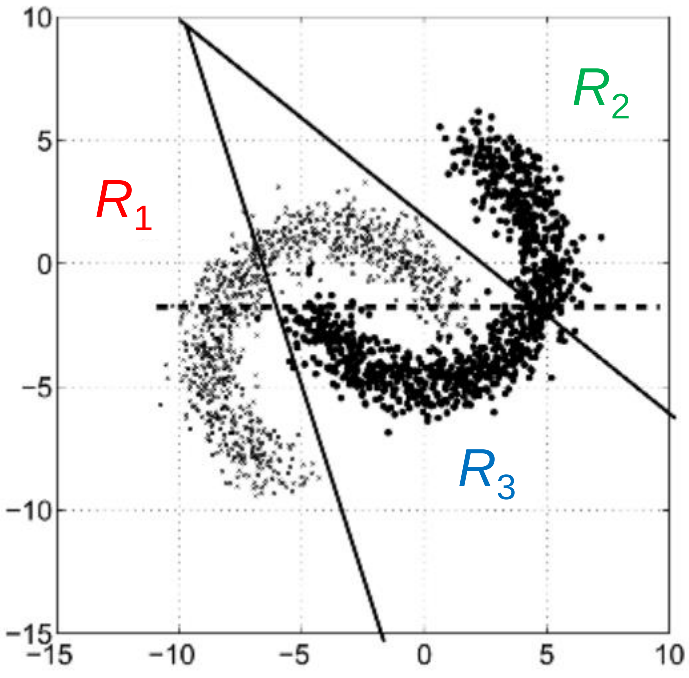
Selection

Suppose an ensemble $D = \{D_1, \ldots, D_L\}$ of classifiers already trained. Then, the feature space $\mathbb{R}^d$ is divided into $K > 1$ selection regions (or regions of competence), which are denoted by $R_1, \ldots, R_K$

- usually, $K = L$
- each region R_i is associated with a classifier, which will be responsible for deciding on the input objects in this part of the space
- these regions are not associated with specific classes, nor do they need to be of a certain shape or size

Selection (ii)

Example: suppose a data set with 2000 points and two classes ω_1 and ω_2, and we have an ensemble with three classifiers D_1, D_2, D_3, each one associated with regions R_1, R_2, R_3



- D_1 always predicts ω_1
- D_2 always predicts ω_2
- D_3 is a linear classifier whose discriminant function is shown as a dashed line

- Accuracy of the individual classifiers or that of a majority vote (fusion) is approximately 0.5
- Accuracy of the selection combiner will be close to 1

Source: http://www.ccas.ru/voron/download/books/machlearn/kuncheva04combining.pdf

Classifier level: stacking

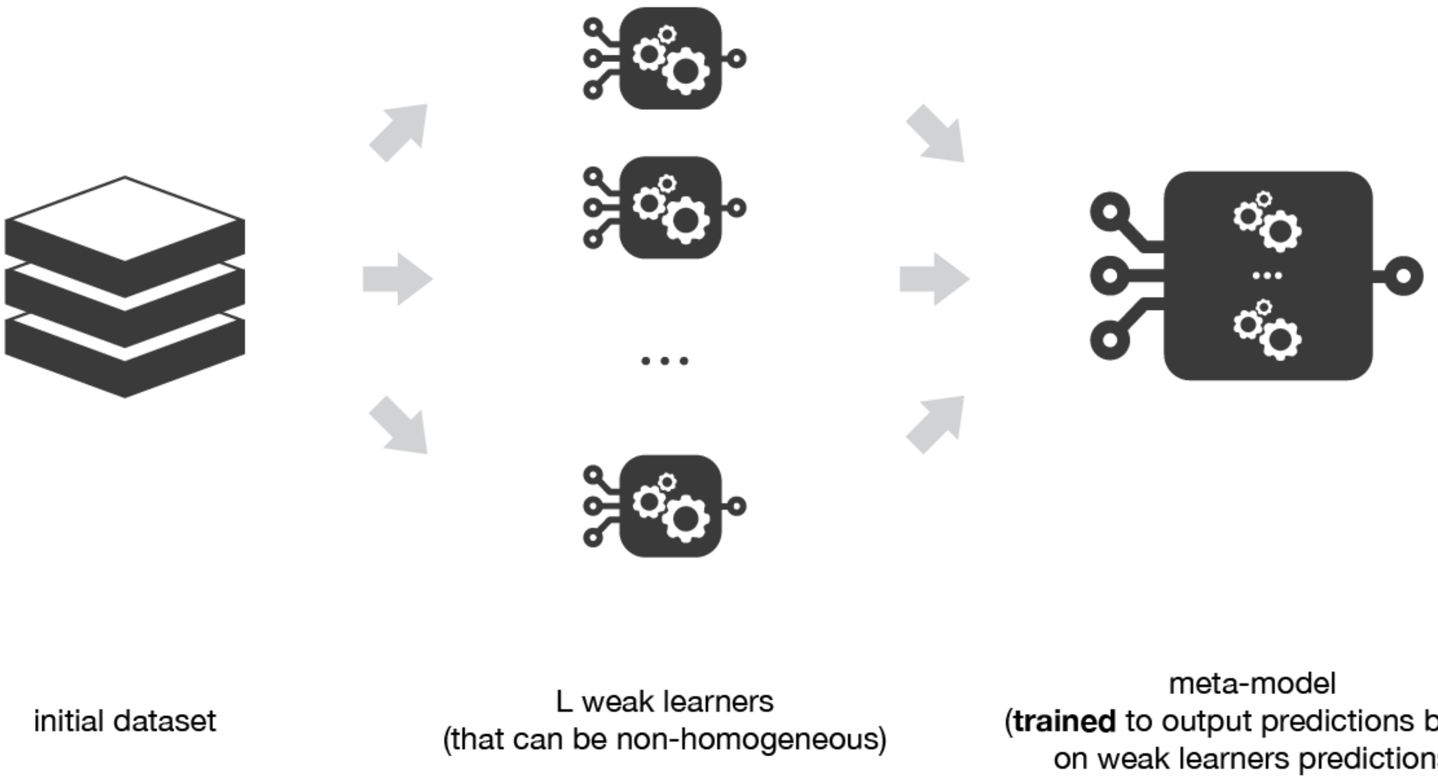
Idea:

- learn various different weak learners (base learners)
- combine the base learners by training a meta-model

Comments:

- we need to define two things in order to build our stacking model: the L base learners we want to fit and the meta-model that combines them
- for example, we can choose as weak learners a k-NN classifier, a decision tree and a SVM, and decide to learn a neural network as meta-model. Then, the neural network will take as inputs the outputs of our three weak learners and will learn to return final predictions based on it

Classifier level: stacking (ii)



Classifier level: stacking (iii)

1. Initialize the parameters

   L, the number of weak learners

2. Split the data into two folds

3. For $l = 1, \ldots, L$

   Train the weak learner to data of the first fold

   Make predictions for data in the second fold

4. Train the meta-model on the second fold, using predictions made by the weak learners as inputs

Data level: bagging

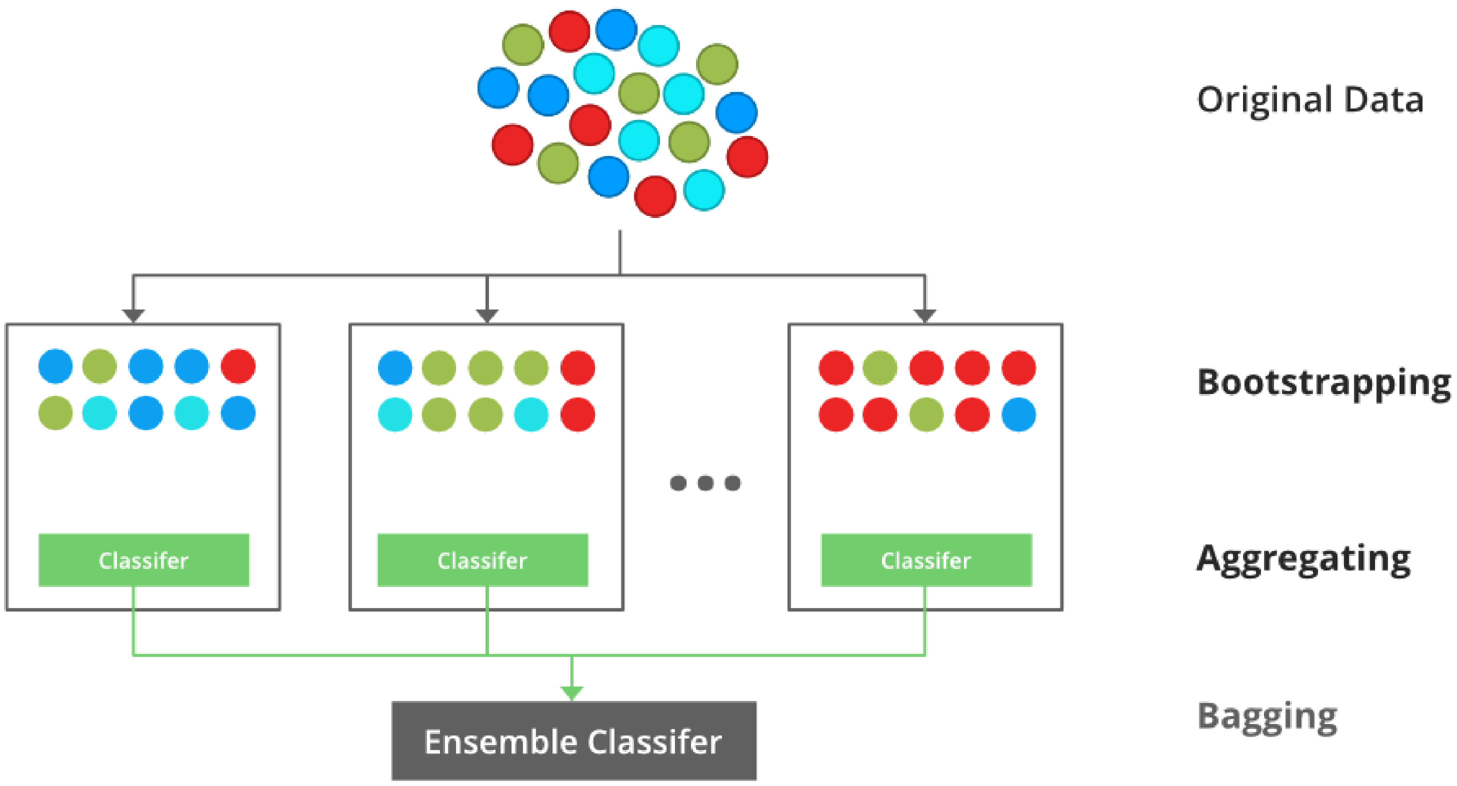
Idea:

- the ensemble is made of classifiers built on bootstrap replicates of the training set $T_{tra} = \{x_1, \ldots, x_n\}$
- the classifier outputs are combined by the plurality vote

Comments:

- we sample with replacement from the original T_{tra} to create L new training sets (often, also of size n)
- all L base classifiers are the same classification model
- the base classifier should be unstable (small changes in T_{tra} lead to large changes in the classifier output (neural networks and decision trees are unstable, *k*-NN is stable)
- this is a parallel algorithm in both its training and operational phases

Data level (ii): bagging



Data level (iii): bagging

Training phase

1. Initialize the parameters
   $D = \emptyset$, the ensemble
   L, the number of classifiers to train
2. For $l = 1, \ldots, L$

   Take a bootstrap sample S_l from the original training set T_{tra}
   Build a classifier D_l using S_l as the training set
   Add the classifier to the current ensemble, $D = D \cup D_l$
   Return D

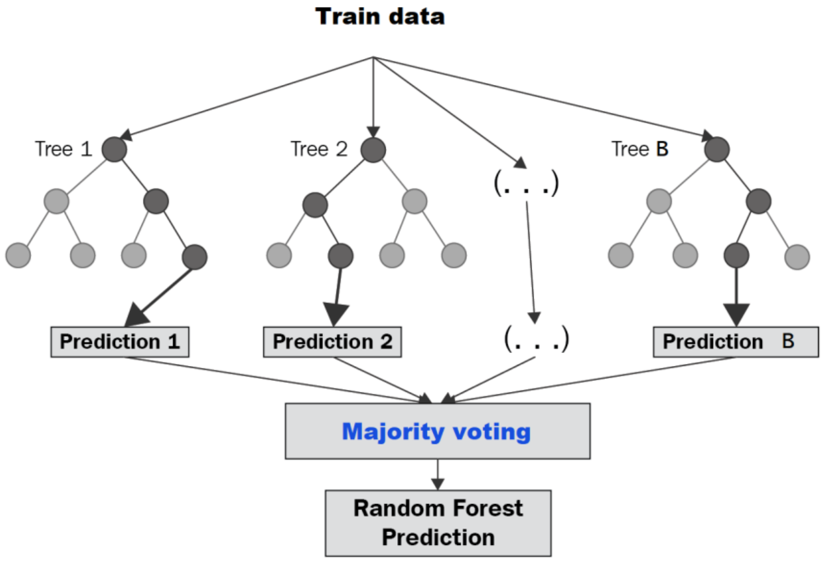
Classification (regression) phase

1. Run $D_1 \cdots D_L$ on the input $\mathbf{x}$
2. Assign $\mathbf{x}$ to the class with the maximum number of votes (simple majority voting, for classification)
   Assign $\mathbf{x}$ with the average of the estimated values (simple average, for regression)

Data level (iv): variants of bagging

Random forest

- a collection of full decision trees built in parallel from random bootstrap sample of the data set
- the final prediction is an average of all of the decision tree predictions



Data level (vi): boosting

Idea:

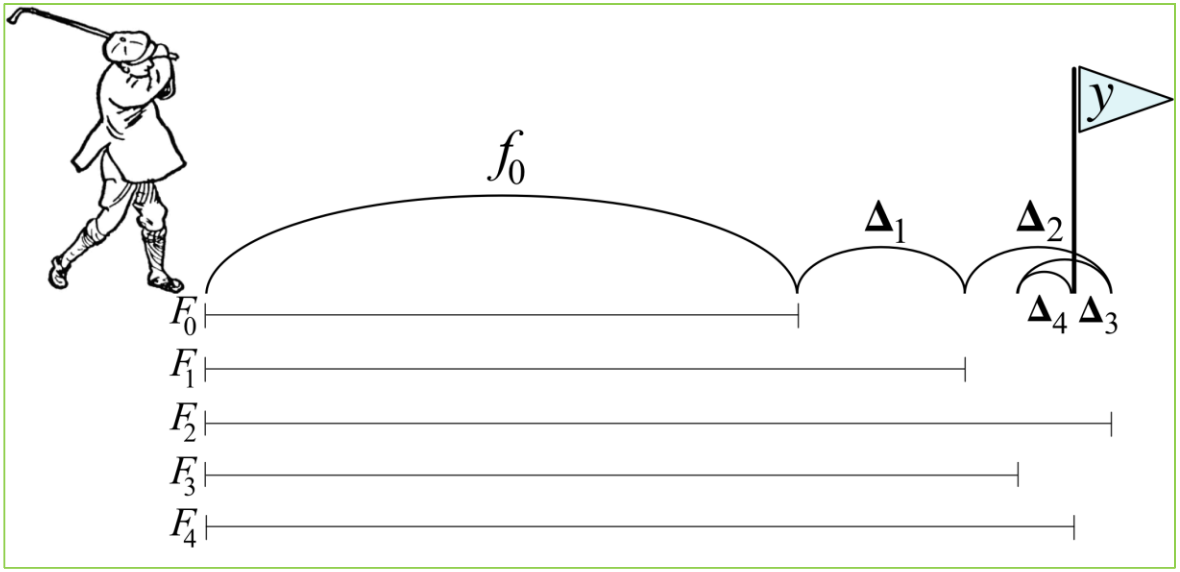
- to develop the ensemble D incrementally, adding one base classifier at a time
- some classifiers have more say in the classification than others
- the classifier D_i is made by taking the errors of the classifier D_{i-1} into account

Comments:

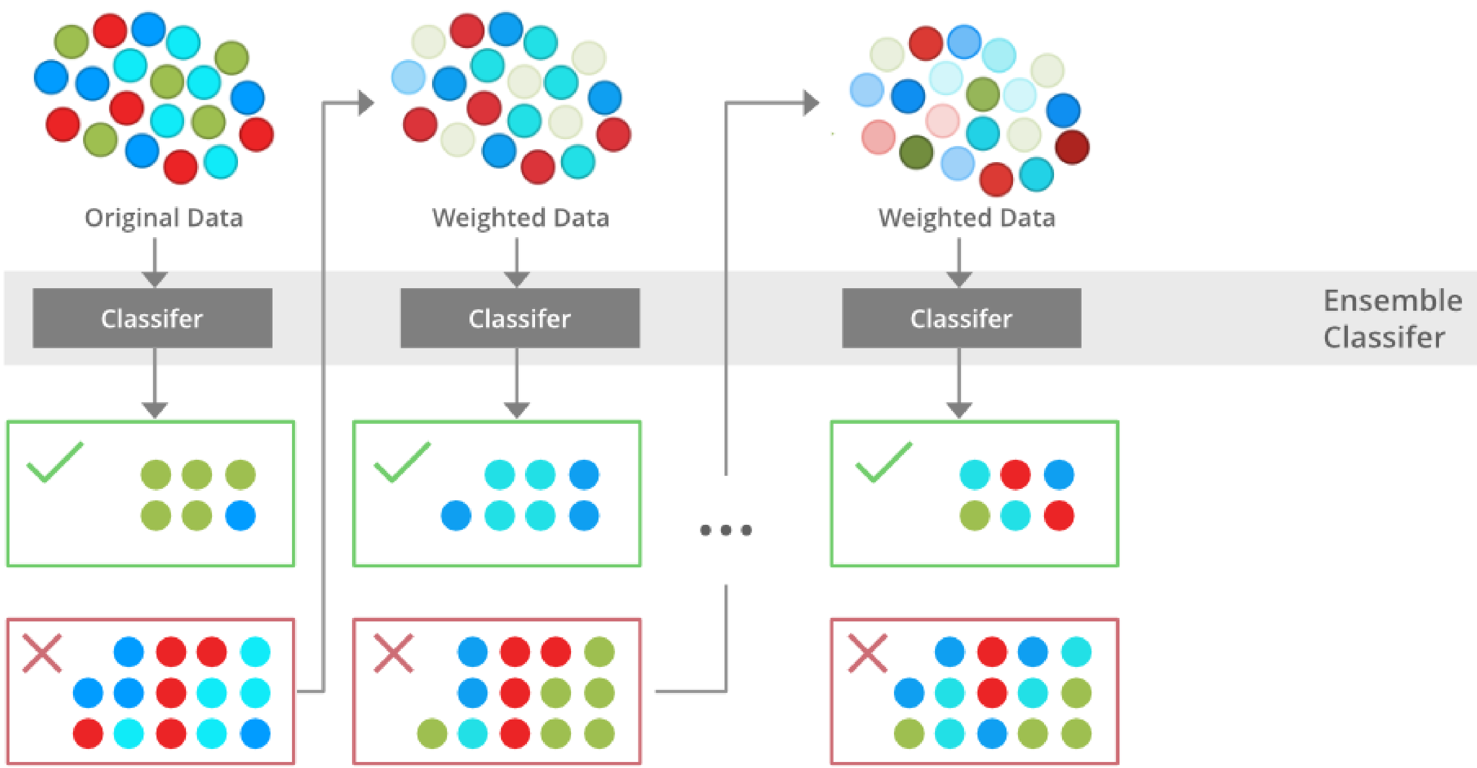
- this is a sequential algorithm
- the errors that the first classifier makes influence how the second classifier is made, and so on

Data level (vii): boosting

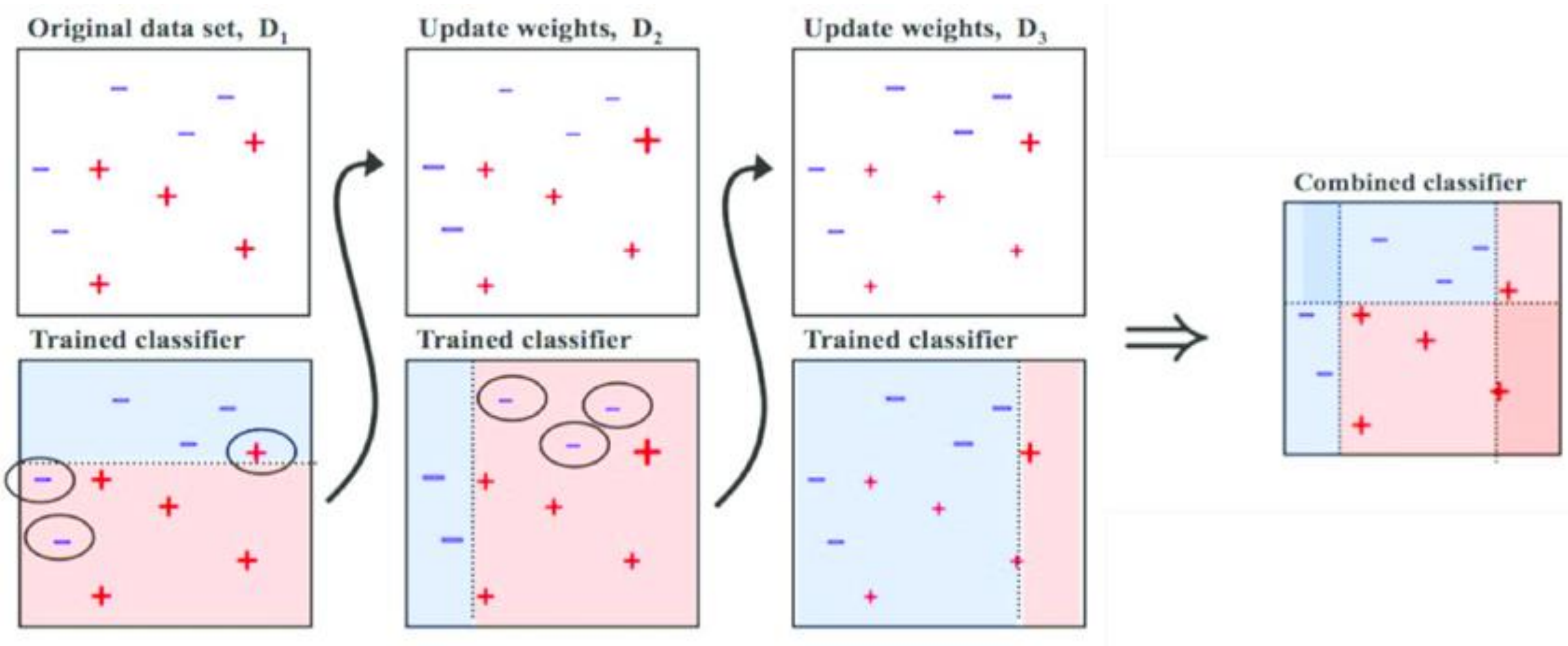
The idea of boosting could be seen as a golfer who initially hits a golf ball towards the hole at position y, but only goes as far as f_0. The golfer then repeatedly hits the ball more gently, moving it toward the hole a little at a time and after reassessing the direction and distance to the hole with each shot.



Data level (viii): boosting



Data level (ix): boosting



Data level (x): boosting (AdaBoost)

Training phase

1. Initialize the parameters

   Set the weights $w^i = 1/n$ (equal weights to each data point)

   $D = \emptyset$, the ensemble

   L, the number of classifiers

2. For $l = 1, \ldots, L$

   Build a classifier D_l with the training data using w^i for $i = 1, \ldots, n$

   Calculate the proportion of errors in classification e_l

   Compute $S_l = \log\big((1 - e_l)/e_l\big)$

   Update the weights w^i (weights of correctly classified samples do not change; incorrectly classified samples are given more weight by multiplying their previous weight by $(1 - e_l)/e_l$

Data level (xi): boosting (AdaBoost)

Classification phase

Given a sample $\mathbf{x}$, if we denote $\hat{y}_l(\mathbf{x})$ its classification using classifier D_l, then

$$\hat{y}(\mathbf{x}) = sign\left(\sum_l S_l \hat{y}_l(\mathbf{x})\right)$$

(if the sum is positive, the observation is classified as belonging to class $+1$, otherwise to class -1)

Data level (xii): variants of boosting

Gradient boosting

- it involves three elements:
  - a loss function to be optimized (e.g., regression may use mean squared error and classification may use logarithmic loss)
  - a weak learner to make predictions (usually, decision tress)
  - an additive model that minimizes the loss function when adding trees (gradient descent is used to minimize the loss)

Data level (xiii): variants of boosting

Extreme gradient boosting (XGBoost)

- an efficient and effective implementation of gradient boosting
- it is highly scalable and can handle large data sets
- trees are built in parallel, instead of sequentially like gradient boosting
- it implements early stopping so we can stop model evaluation when additional trees offer no improvement

Data level (xiv): variants of boosting

Categorical boosting (CatBoost)

- it is designed to work on heterogeneous data (categorical, numerical, logical, …)
- it works well with less data
- improved accuracy by reducing overfitting

Feature level: random subspace

Idea:

- the ensemble is made of classifiers built on random subsets of features (with replacement) of predefined size d_{rs} ($d_{rs} < d$)
- the classifier outputs are combined by the plurality vote

Comments:

- an attractive choice for high-dimensional problems where the number of features (d) is much larger than the number of training points (n)
- it works best when the discriminative information is "dispersed" across all the features

Feature level (ii): random subspace

Training phase

1. Initialize the parameters

   $D = \varnothing$, the ensemble

   L, the number of classifiers to train

2. For $k = 1, \ldots, L$

   Pick up d_{rs} features from d with replacement

   Build a classifier D_k using the subspace sample

   Add the classifier to the current ensemble, $D = D \cup D_k$

3. Return D

Classification phase

1. Run $D_1, \ldots, D_L$ on the input x
2. Assign x to the class with the maximum number of votes