Ensemble learning

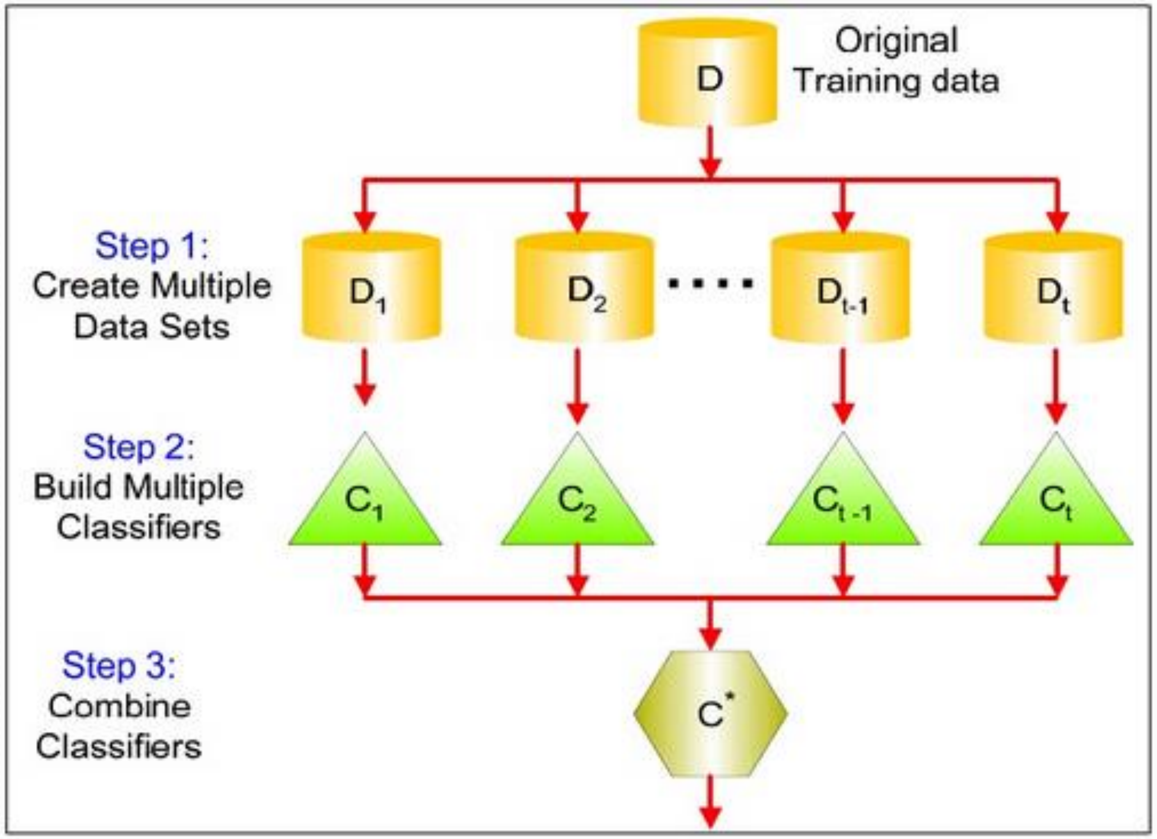
Basic idea

- Concordet's jury theorem (1785): imagine that a group of people has to select between two choices (from which only one is correct). They vote independently, and the probability that they vote correctly is *p*. The votes are combined by the majority rule. Let M denote the probability that the majority vote is correct.
  - Concordet's theorem says that if $p>0.5$ then $M \rightarrow 1$ if the number of votes goes to infinity
- This means that the crowd is more clever than the individuals under relatively weak assumptions
  - Each individual must be correct with $p>0.5$ (better than random guessing)
  - Their should make independent decisions
- Now, how can we apply this idea in machine learning?

Strong vs. weak learners

- Strong learner: we seek to produce one classifier for which the classification error can be made arbitrarily small
    - So far we were looking for such methods
- Weak learner: a classifier which is just better than random guessing
    - Now this will be our only expectation
- Ensemble learning: instead of creating one strong classifier, we create a huge set of weak classifiers, then we combine their outputs into one final decision
    - According to Concordet's theorem, under proper conditions we can expect that the ensemble model can attain an error rate that is arbitrarily close to zero
    - While creating a lot of weak classifiers is hopefully a much easier task than to create one strong classifier

Ensemble learning


D
Original Training data
Step 1: Create Multiple Data Sets
D_1
D_2
....
D_{t-1}
D_t
Step 2: Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3: Combine Classifiers
C^*

Conditions

- Training the same classifier on the same training data several times would give the same result for most machine learning algorithms
  - Exception: methods where the training involves some randomness
- Combining these classifiers would make no sense
- Classifier combination gives the best result if
  - The classifier outputs for the same input are diverse
  - The classifiers operate independently (or at least partially independently)
  - The classifiers have some unique or “local” knowledge
    - E.g. they are trained on different (or at least partially different) training data sets
- We also must have some formalism to combine (aggregate) the opinions of the various classifiers

How to produce diverse classifiers?

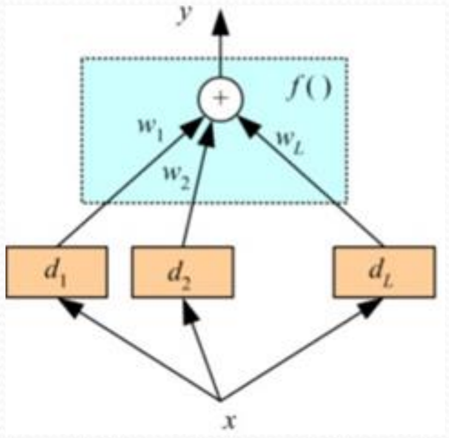
- We can combine *different learning algorithms* (“hybridization”)
  - E.g. we can train a GMM, an SVM, a k-NN,… over the same data, and then combine their output
- We can combine the same learning algorithm trained several times over the same data
  - This works only if there is some random factor in the training method
  - E.g.: neural networks trained with *different random initialization*
- We can combine the same learning algorithm trained over *different subsets of the training data*
  - We can also try using *different subsets of the features*
  - Or *different subsets of the target classes* (multi-class task, lot of classes)
- For certain algorithms we can use the same algorithm over the same data, but with a *different weighting over the data instances*

Randomization of decision trees

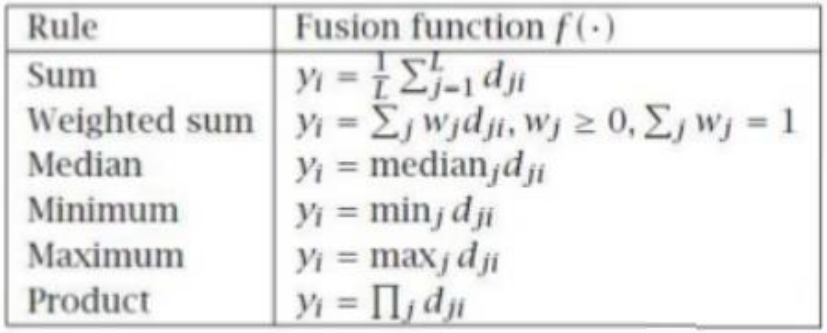
- The decision tree is a very popular choice for combination-based methods
  - (Small) decision trees are not really efficient classifiers
  - But their training is very simple and fast, so it is very easy to create an ensemble learner from a huge set of (small) decision trees
- The decision tree algorithm is deterministic, how can we modify it to produce different learners at different runs?
  - Data randomization – use different subsets of training data for each tree
  - "Random subspace" method – each tree is grown using a random subset of the features
  - Algorithm randomization – instead of choosing the best attribute for node splitting, we select an attribute randomly from the K best attributes

Aggregation methods

- There are several methods to combine (aggregate) the outputs of the various classifiers
- When the output is a class label:
  - Majority voting
  - Weighted majority voting (e.g we can weight each classifier by its reliability (which also has to be estimated somehow, of course…)



- When the output is numeric (e.g. a probability estimate for each class c_i):
  - We can combine the d_j scores by taking their (weighted) mean, product, minimum, maximum, …



Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

- Stacking
  - Instead of using the above simple aggregation rules, we can train yet another classifier on the output values of the base classifiers

Ensemble-based methods

- Classifier combination is a general concept that can be applied for any training task, and by using any set of classifiers that are diverse (according to the previous slides)
- But today we concentrate on those algorithms that were specially invented to exploit the ensemble approach
    - They will create a large set of classifiers, here called the “base learners”
    - These classifiers can be weak, as we expect that the power of these methods will be in the ensemble approach
    - We will create a lot from these base classifiers
    - We will try to force them to be diverse, or in better methods, to complement each other
    - During their combination we will seek maximum classification accuracy
- These methods will be:
    - Bagging, Boosting, AdaBoost, Random Forests

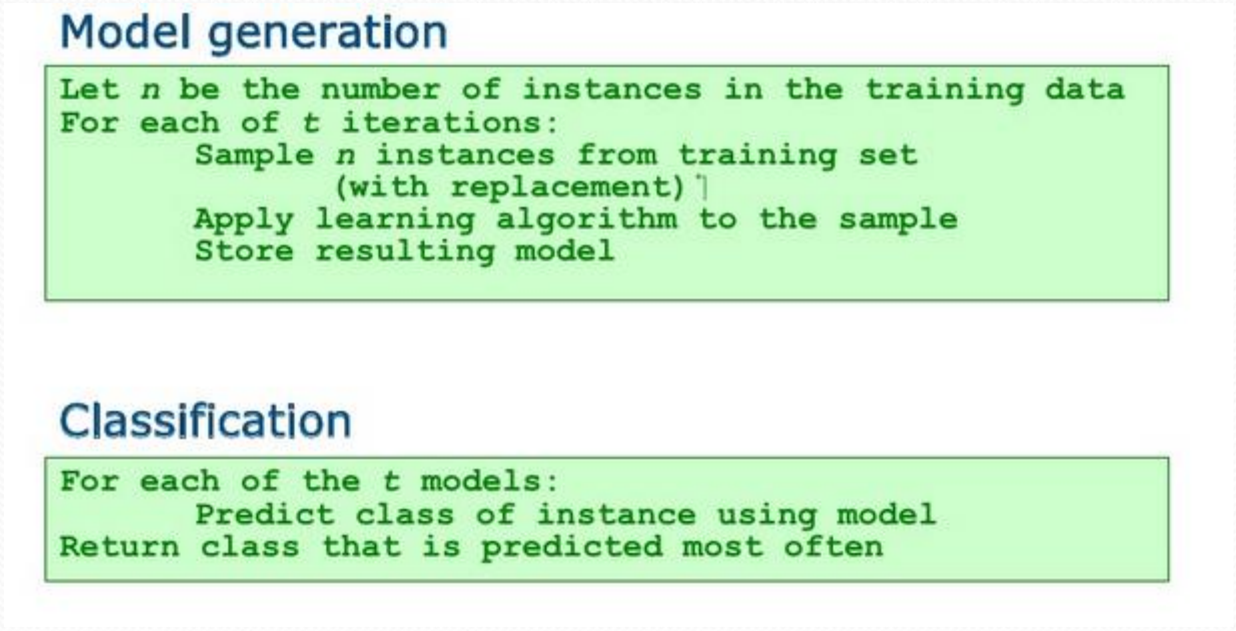
Bagging

- Bagging = **B**ootstrap + **agg**regat**ing**
- It uses bootstrap resampling to generate L different training sets from the original training set
- On the L training sets it trains L base learners
- During testing it aggregates the L learners by taking their average (using uniform weights for each classifiers), or by majority voting
- The diversity or complementarity of the base learners is not controlled in any way, it is left to chance and to the instability of the base learning method
- The ensemble model is almost always better than the unique base learners if the base learners are unstable (which means that a small change in the training dataset may cause a large change in the result of the training)

Bootstrap resampling

- Suppose we have a training set with n samples
- We would like to create L different training sets from this
- Bootstrap resampling takes random samples from the original set with replacement
- Randomness is required to obtain different sets for L rounds of resampling
- Allowing replacement is required to be able to create sets of size n from the original data set of size n
- As the L training sets are different, the result of the training over these set will also be more or less different, independent of what kind of training algorithm we use
  - Works better with unstable learners (e.g. neural nets, decision trees)
  - Not really effective with stable learners (e.g. k-NN, SVM)

Bagging -Summary



Model generation

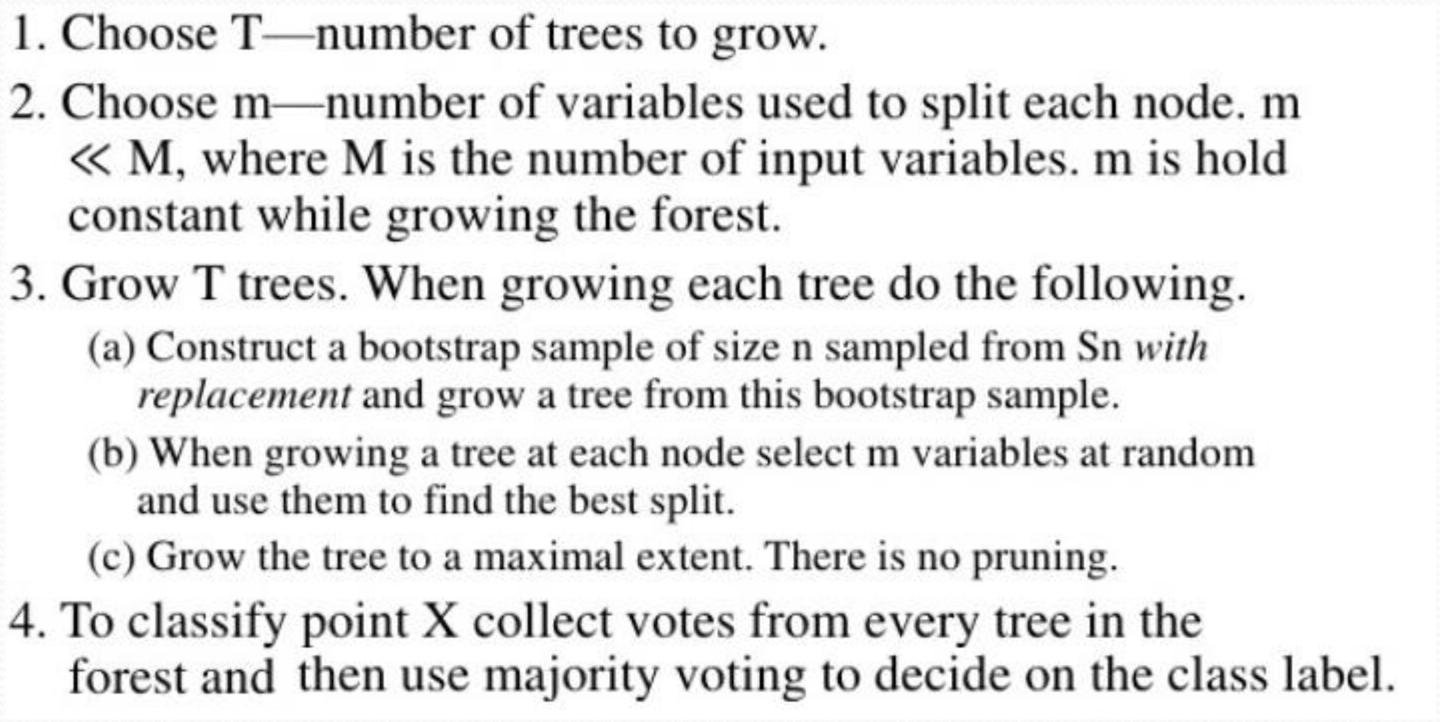
```
Let $n$ be the number of instances in the training data
For each of $t$ iterations:
      Sample $n$ instances from training set
            (with replacement)
      Apply learning algorithm to the sample
      Store resulting model
```

Classification

```
For each of the $t$ models:
      Predict class of instance using model
Return class that is predicted most often
```

Random Forests- an ensemble method for decision trees

- Input: training data set Sn, T, m



1. Choose T—number of trees to grow.
2. Choose m—number of variables used to split each node. m ≪ M, where M is the number of input variables. m is hold constant while growing the forest.
3. Grow T trees. When growing each tree do the following.
   (a) Construct a bootstrap sample of size n sampled from Sn *with replacement* and grow a tree from this bootstrap sample.
   (b) When growing a tree at each node select m variables at random and use them to find the best split.
   (c) Grow the tree to a maximal extent. There is no pruning.
4. To classify point X collect votes from every tree in the forest and then use majority voting to decide on the class label.

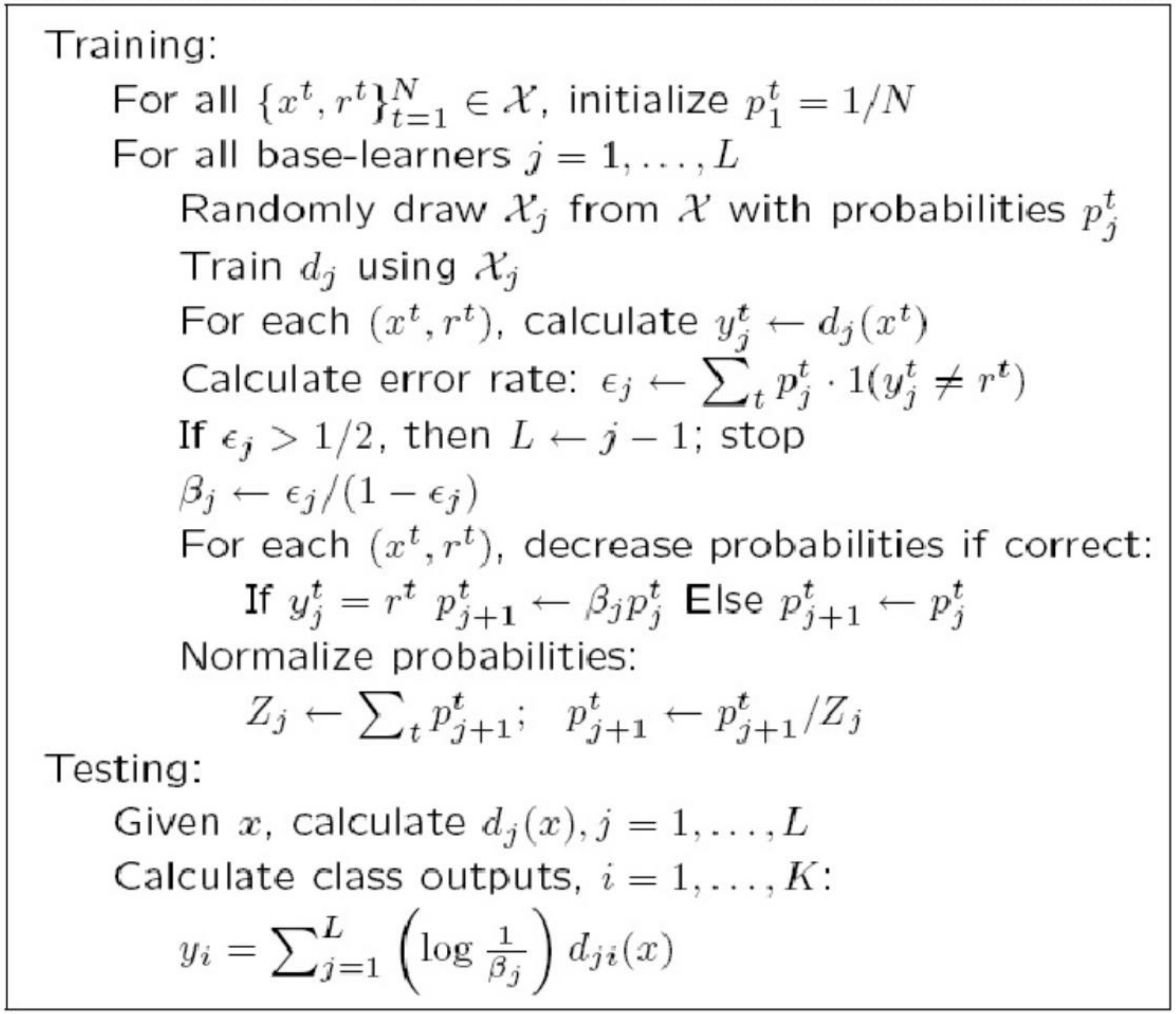
Boosting 1

- Bagging created a diversity of base learners by creating different variants of the training dataset randomly
  - However, we do not have direct control over the usefulness of the newly added classifiers
- We would expect a better performance if the learners also complemented each other
  - They would have “expertise” on different subsets of the data
  - So they would work better on different subsets
- The basic idea of boosting is to generate a series of base learners which complement each other
  - For this, we will force each learner to focus on the mistakes of the previous learner

Boosting 2

- We represent the importance of each sample by assigning weights to the samples
    - Correct classification → smaller weights
    - Misclassified samples → larger weights
- The weights can influence the algorithm in two ways
    - Boosting by sampling: the weights influence the resampling process
        - This is a more general solution
    - Boosting by weighting: the weights influence the learner
        - Works only with certain learners
- Boosting also makes the aggregation process more clever: We will aggregate the base learners using weighted voting
    - Better weak classifier gets a larger weight
    - We iteratively add new base learners, and iteratively increase the accuracy of the combined model

AdaBoost (Adaptive Boosting)



```
Training:
    For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$
    For all base-learners $j = 1, \ldots, L$
        Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities $p_j^t$
        Train $d_j$ using $\mathcal{X}_j$
        For each $(x^t, r^t)$, calculate $y_j^t \leftarrow d_j(x^t)$
        Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$
        If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop
        $\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$
        For each $(x^t, r^t)$, decrease probabilities if correct:
            If $y_j^t = r^t$ $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$
        Normalize probabilities:
            $Z_j \leftarrow \sum_t p_{j+1}^t$;  $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$
Testing:
    Given $x$, calculate $d_j(x), j = 1, \ldots, L$
    Calculate class outputs, $i = 1, \ldots, K$:
        $y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$
```

Start with equal weights

Random resampling

Estimated labels

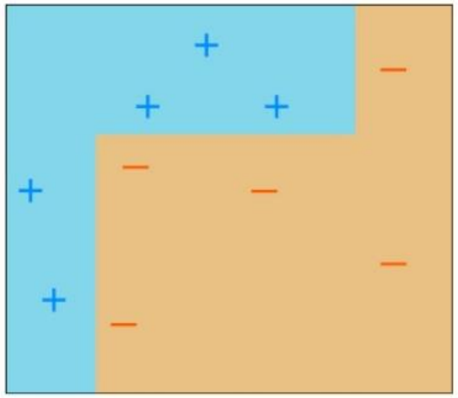
Error is the weighted sum of not hit samples

Not a weak learner, must stop

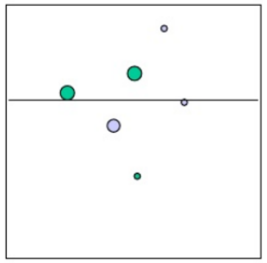
Weighted aggregation of the classifiers

Example

- We will have two classes with 5-5 training samples
- We will have two continuous features
- The training data with a possible correct decision boundary:

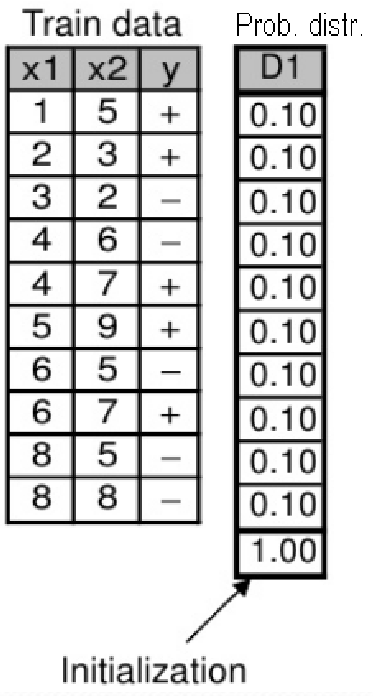


- As the base learners, we will use decision trees with depth=1
    - These are also known as "decision stumps"
    - In the case of continuous features these correspond to axis-parallel lines:
    - Outputs 1 on one side, -1 on the other side



Initialization + first iteration

- We initialize the weights to equal weights → D_1
- Then we train the first stump and calculate its error and combination weight
- Then we increase the weight of the missed samples → new prob. distr. D_2



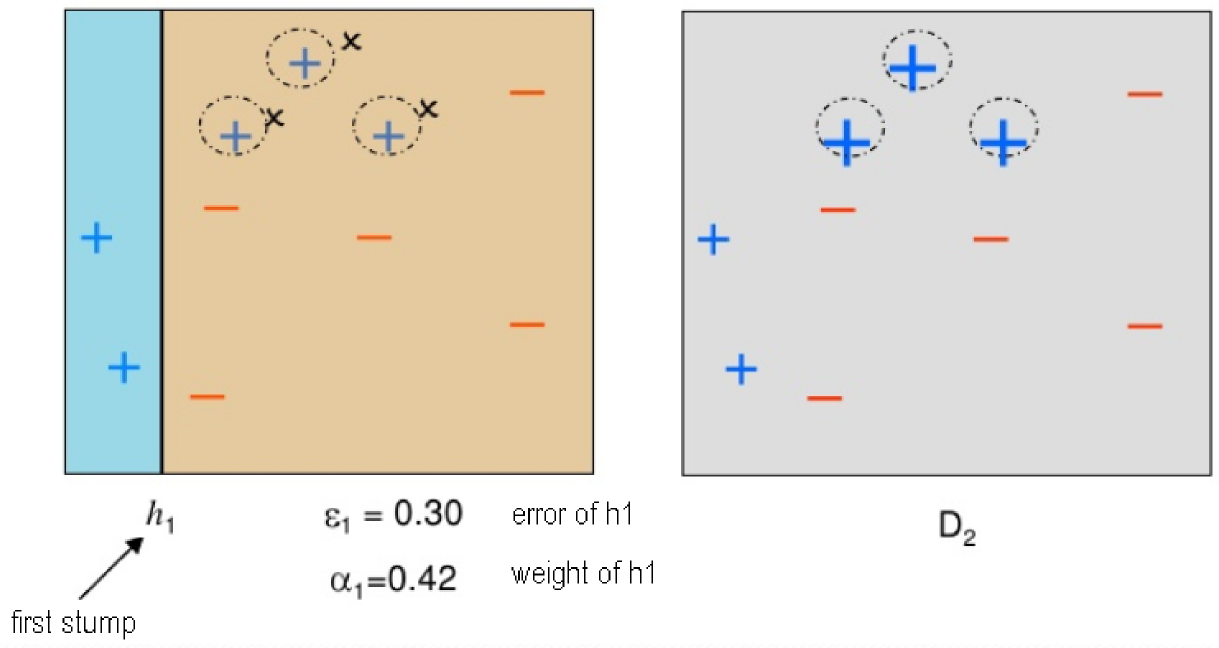
Train data

x1	x2	y
1	5	+
2	3	+
3	2	–
4	6	–
4	7	+
5	9	+
6	5	–
6	7	+
8	5	–
8	8	–

Prob. distr.

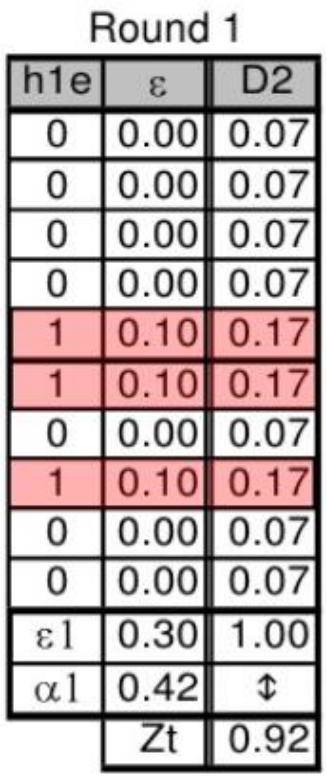
D1
0.10
0.10
0.10
0.10
0.10
0.10
0.10
0.10
0.10
0.10
1.00

Initialization



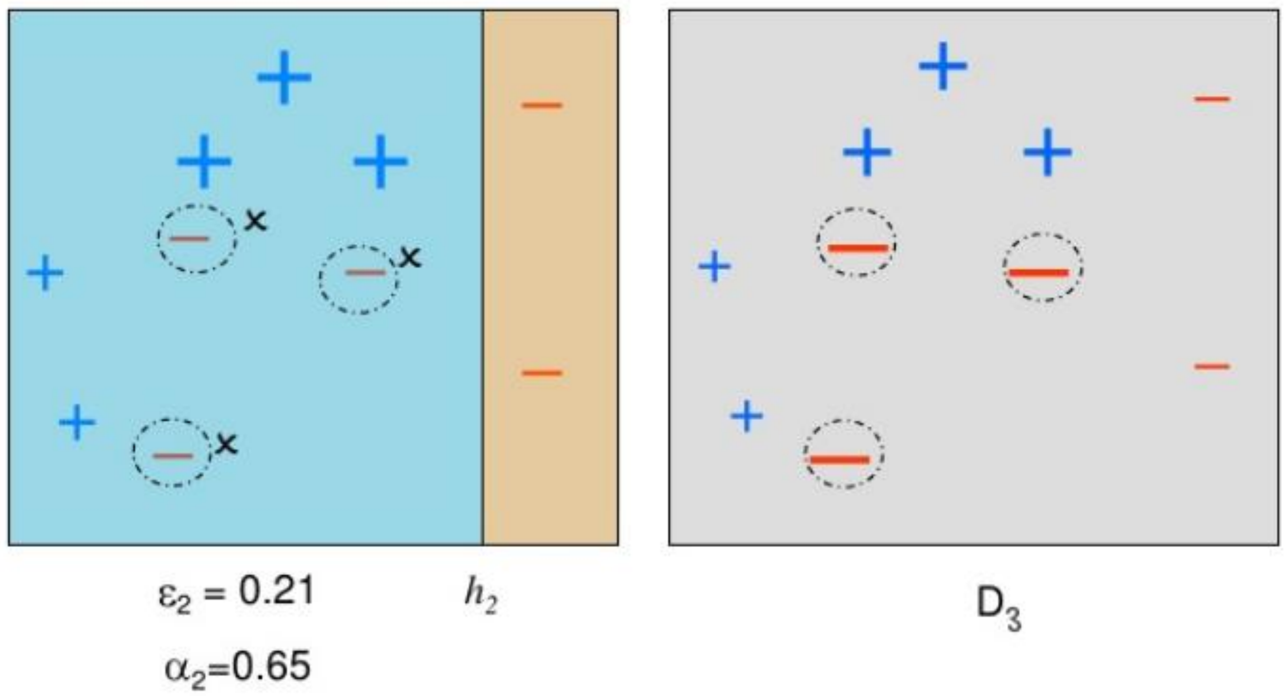
Second iteration

- We sample the data using the new weights D_2
- We train the second stump and calculate its error and combination weight
- We increase the weights of the missed samples $\rightarrow D_3$



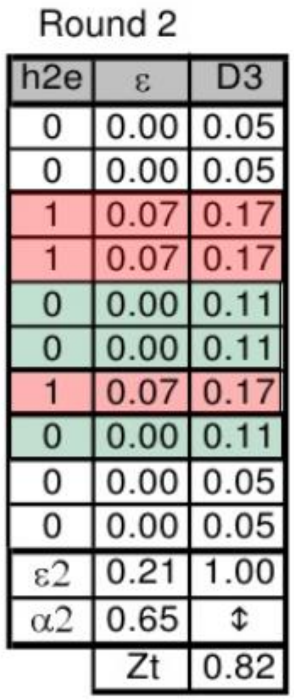
Round 1

h1e	ε	D2
0	0.00	0.07
0	0.00	0.07
0	0.00	0.07
0	0.00	0.07
1	0.10	0.17
1	0.10	0.17
0	0.00	0.07
1	0.10	0.17
0	0.00	0.07
0	0.00	0.07
ε1	0.30	1.00
α1	0.42	↕
	Zt	0.92



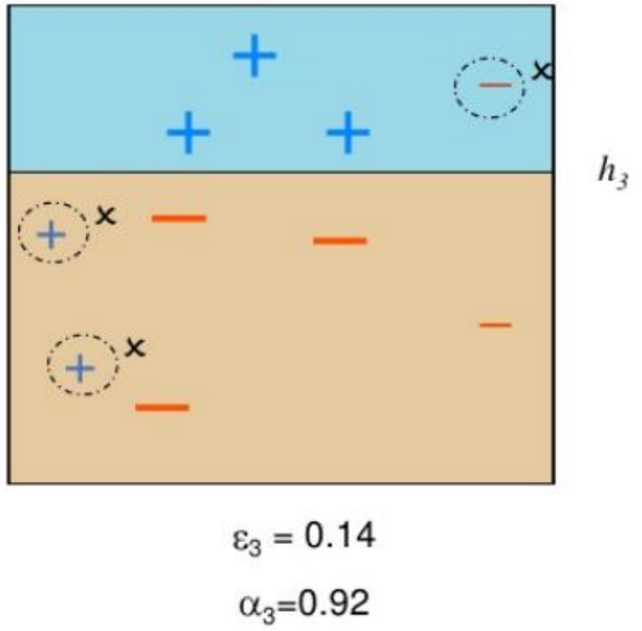
Third iteration

- We sample the data using the new weights D_3
- Then we train the third stump and calculate its error and weight
- Then we stop, because the combined error became 0



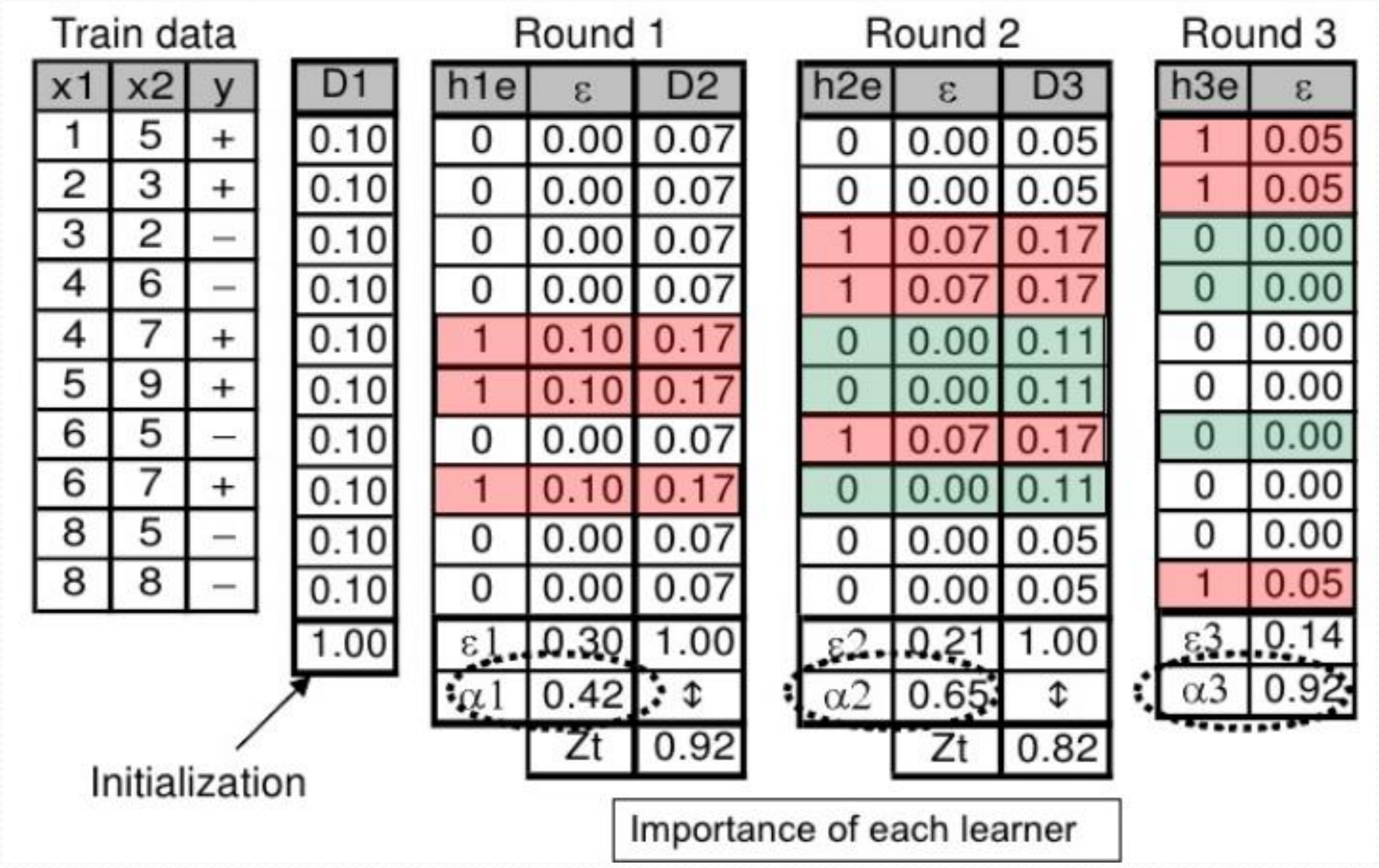
Round 2

h2e	ε	D3
0	0.00	0.05
0	0.00	0.05
1	0.07	0.17
1	0.07	0.17
0	0.00	0.11
0	0.00	0.11
1	0.07	0.17
0	0.00	0.11
0	0.00	0.05
0	0.00	0.05
ε2	0.21	1.00
α2	0.65	⇕
	Zt	0.82



Obtaining the final classifier

- We have to combine the 3 stumps using their weights



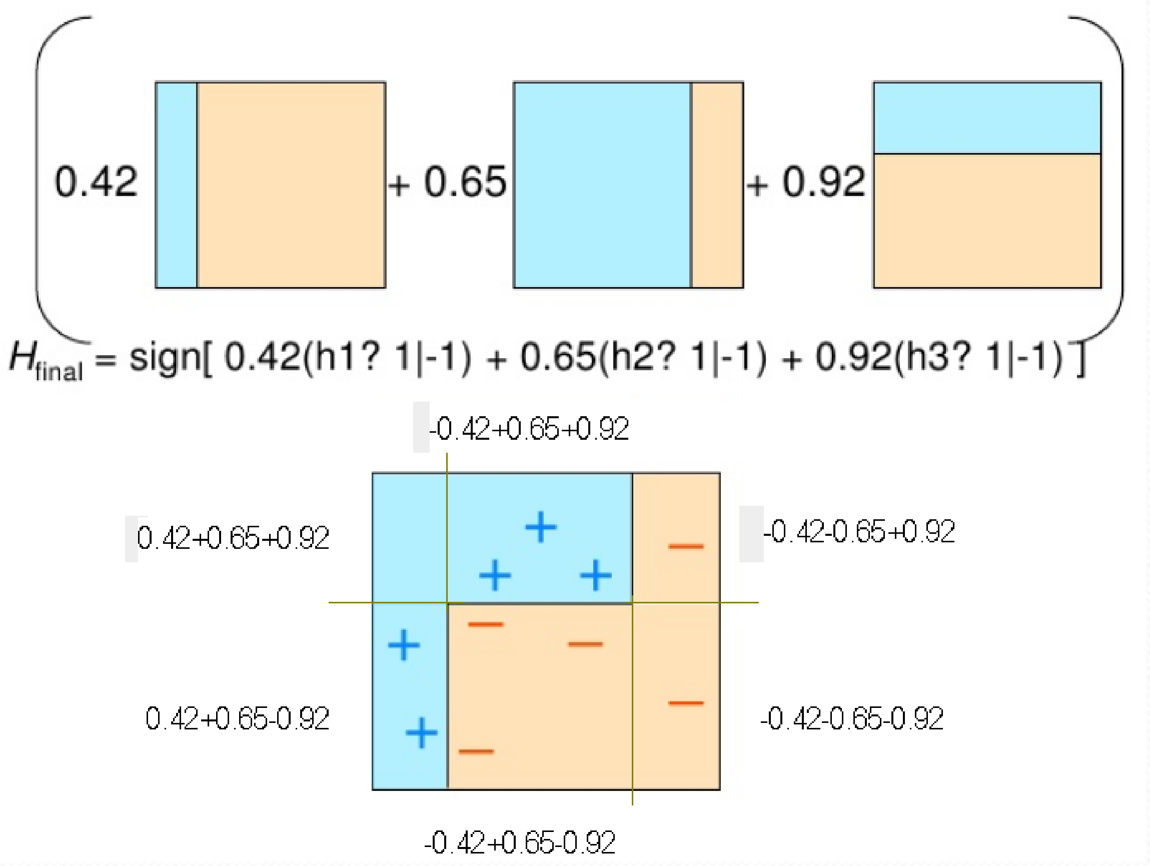
Train data				Round 1			Round 2			Round 3	
x1	x2	y	D1	h1e	ε	D2	h2e	ε	D3	h3e	ε
1	5	+	0.10	0	0.00	0.07	0	0.00	0.05	1	0.05
2	3	+	0.10	0	0.00	0.07	0	0.00	0.05	1	0.05
3	2	–	0.10	0	0.00	0.07	1	0.07	0.17	0	0.00
4	6	–	0.10	0	0.00	0.07	1	0.07	0.17	0	0.00
4	7	+	0.10	1	0.10	0.17	0	0.00	0.11	0	0.00
5	9	+	0.10	1	0.10	0.17	0	0.00	0.11	0	0.00
6	5	–	0.10	0	0.00	0.07	1	0.07	0.17	0	0.00
6	7	+	0.10	1	0.10	0.17	0	0.00	0.11	0	0.00
8	5	–	0.10	0	0.00	0.07	0	0.00	0.05	0	0.00
8	8	–	0.10	0	0.00	0.07	0	0.00	0.05	1	0.05
			1.00	ε1	0.30	1.00	ε2	0.21	1.00	ε3	0.14
				α1	0.42	⇕	α2	0.65	⇕	α3	0.92
					Zt	0.92		Zt	0.82		

Initialization

Importance of each learner

The ensemble classifier

- We have to combine the 3 stumps using their weights



Bias-Variance in Ensemble learning

- Bagging: There is both theoretical and empirical evidence that bagging reduces the variance part of the error (see percious discussion under "why bagging works"
  - The efficient variance reduction effect of ensemble learning is the main reason why there is no need to prune the trees in the Random Forest method
- AdaBoost: there is empirical evidence that AdaBoost reduces both the bias nad the variance part of the error. In particular, it seems that bias is mostly reduced in earlier iterations, while variance in later ones