

CS361

(Software Engineering Program)

Artificial Intelligence II - Applied Machine Learning

Lecture 7

A Basic Introduction to Ensemble Learning

[Bagging, Random Forests, Boosting, AdaBoost]

Amr S. Ghoneim

(Assistant Professor, Computer Science Dept.)

Helwan University

Fall 2019

Lecture is based on its counterparts in the following courses (and the following resources):

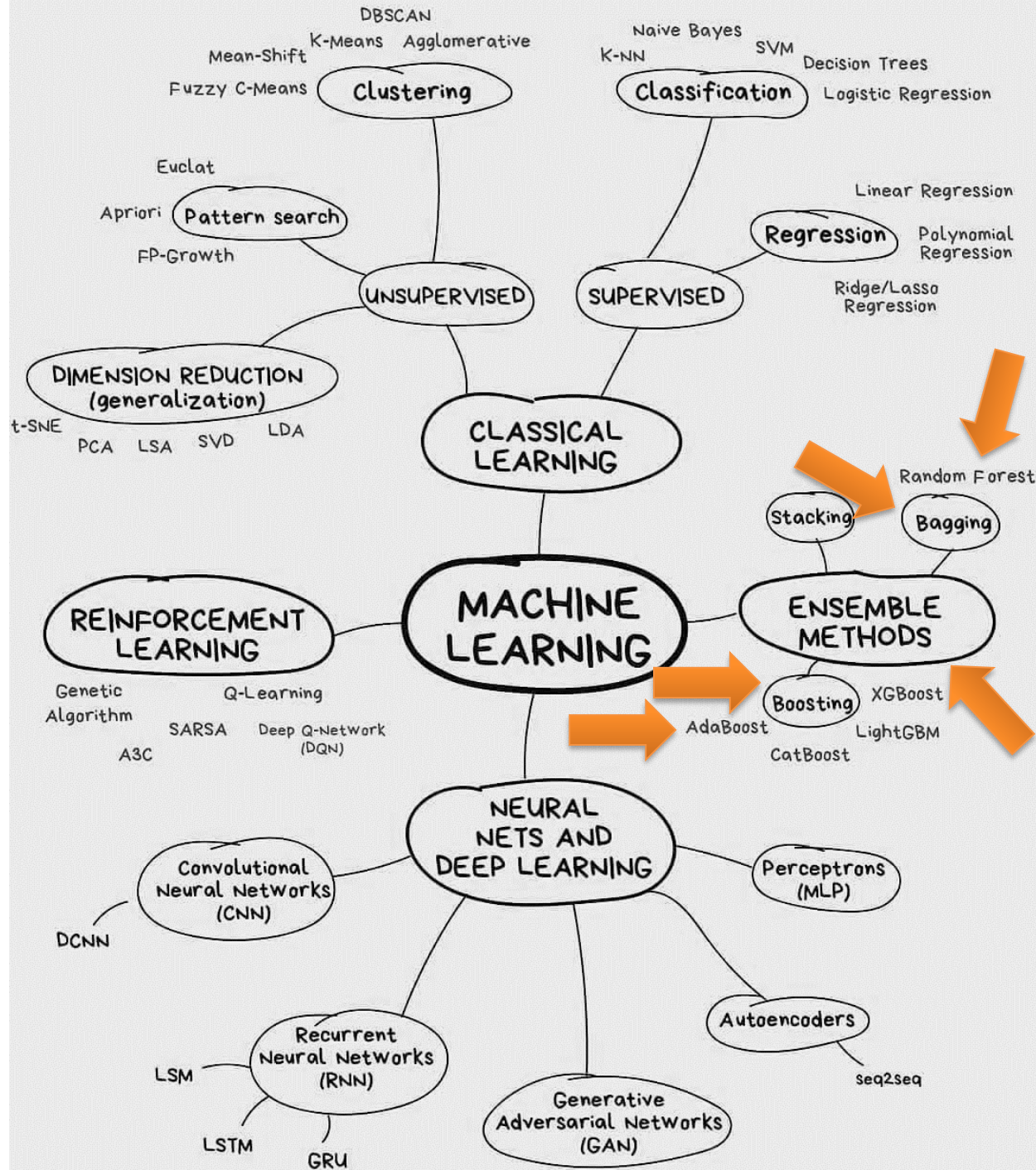
- *Ensemble Learning*, **University of Szeged "Szegedi Tudományegyetem" (Hungary)**, Institute of Informatics.
- *Web-Mining Agents: Classification with Ensemble Methods*, **Universität zu Lübeck (Germany)**, R. Möller, at the Institute of Information Systems.
- *Machine Learning CS165B*, **UCSB University of California Santa Barbara (California USA)**, Department of Computer Science.
- *Explaining AdaBoost*, **Princeton University (New Jersey USA)**, Rob E. Schapire, Department of Computer Science: <https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf>

Today's Key Concepts

- **Basic Idea**
 - **Condorcet's Jury Theorem**
 - **Strong versus Weak Learners**
 - **Ensemble Learning .. a Generic Approach**
 - **Conditions**
 - **How to produce Diverse Classifiers?**
 - **Randomization of Decision Trees**
 - **Random Forests**
- **Ensemble-Based Methods specifically invented for Ensemble Learning**
 - **Bagging**
 - **Bootstrap Resampling**
 - **Random Forests**
 - **Boosting**
 - **Boosting by Sampling**
 - **Boosting by Weighting**
 - **Adaboost (Adaptive Boosting)**

Machine Learning?

{Artificial
Intelligence}
Machine
Learning Map

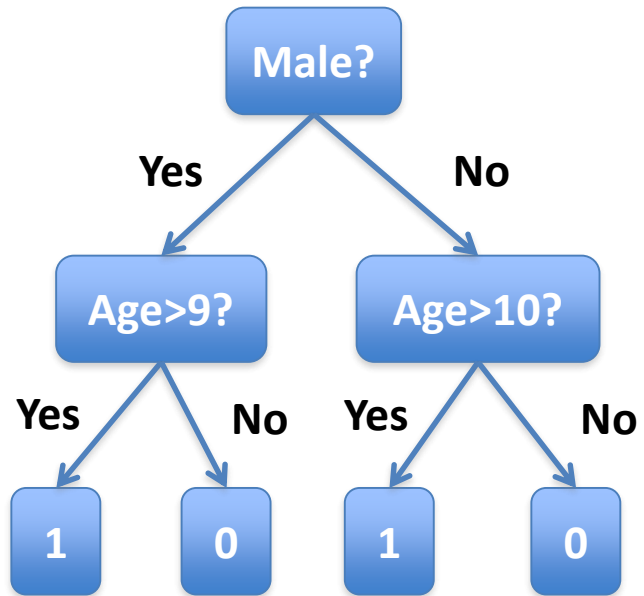


Recap: Supervised Learning

Goal: learn predictor $h(x)$:

- High accuracy (*low error*).
- Using training data $\{ (x_1, y_1), \dots, (x_n, y_n) \}$.

Recap: Supervised Learning



Person	Age	Male?	Height > 55"
Alice	14	0	1
Bob	10	1	1
Carol	13	0	1
Dave	8	1	0
Erin	11	0	0
Frank	9	1	1
Gena	8	0	0



$$x = \begin{bmatrix} \text{age} \\ 1_{[\text{gender}=\text{male}]} \end{bmatrix} \quad y = \begin{cases} 1 & \text{height} > 55'' \\ 0 & \text{height} \leq 55'' \end{cases}$$

Basic Idea .. Condorcet's Jury Theorem

Condorcet's jury theorem (1785) is a **political science** theorem about the relative probability of a given group of individuals arriving at a correct decision. The theorem was *first expressed by the Marquis de Condorcet in his 1785 work Essay on the Application of Analysis to the Probability of Majority Decisions.*

The assumptions of the simplest version of the theorem are that:

- A group wishes to reach a decision by majority vote (*i.e., the votes are combined by the majority rule*).
- One of the two outcomes of the vote is correct (*i.e., they have to select between two choices from which only one is correct*).
- Each voter has an independent probability p of voting for the correct decision (*i.e., They vote independently, and the probability that they vote correctly is p*).

The theorem asks how many voters we should include in the group.

Basic Idea .. Condorcet's Jury Theorem

The result depends on whether p is greater than or less than $1/2$:


















































- If p is greater than $1/2$ (*each voter is more likely to vote correctly*), then adding more voters increases the probability that the majority decision is correct. In the limit, the probability that the majority votes correctly approaches 1 as the number of voters increases.
- On the other hand, if p is less than $1/2$ (*each voter is more likely to vote incorrectly*), then adding more voters makes things worse: the optimal jury consists of a single voter.

That is, Let M denotes the probability that the majority vote is correct, thus if $p > 0.5$, then $M \rightarrow 1$ if the number of votes goes to infinity.

- The crowd is more clever than the individuals under relatively weak assumptions.
- Each individual must be correct: $p > 0.5$ (*better than random guessing*).
- They should make independent decisions.

Now, how can we apply this idea in machine learning?

Basic Idea .. An Example: Weather Forecast

Reality							
1							
2							
3							
4							
5							
Combine							

Strong *versus* Weak Learners

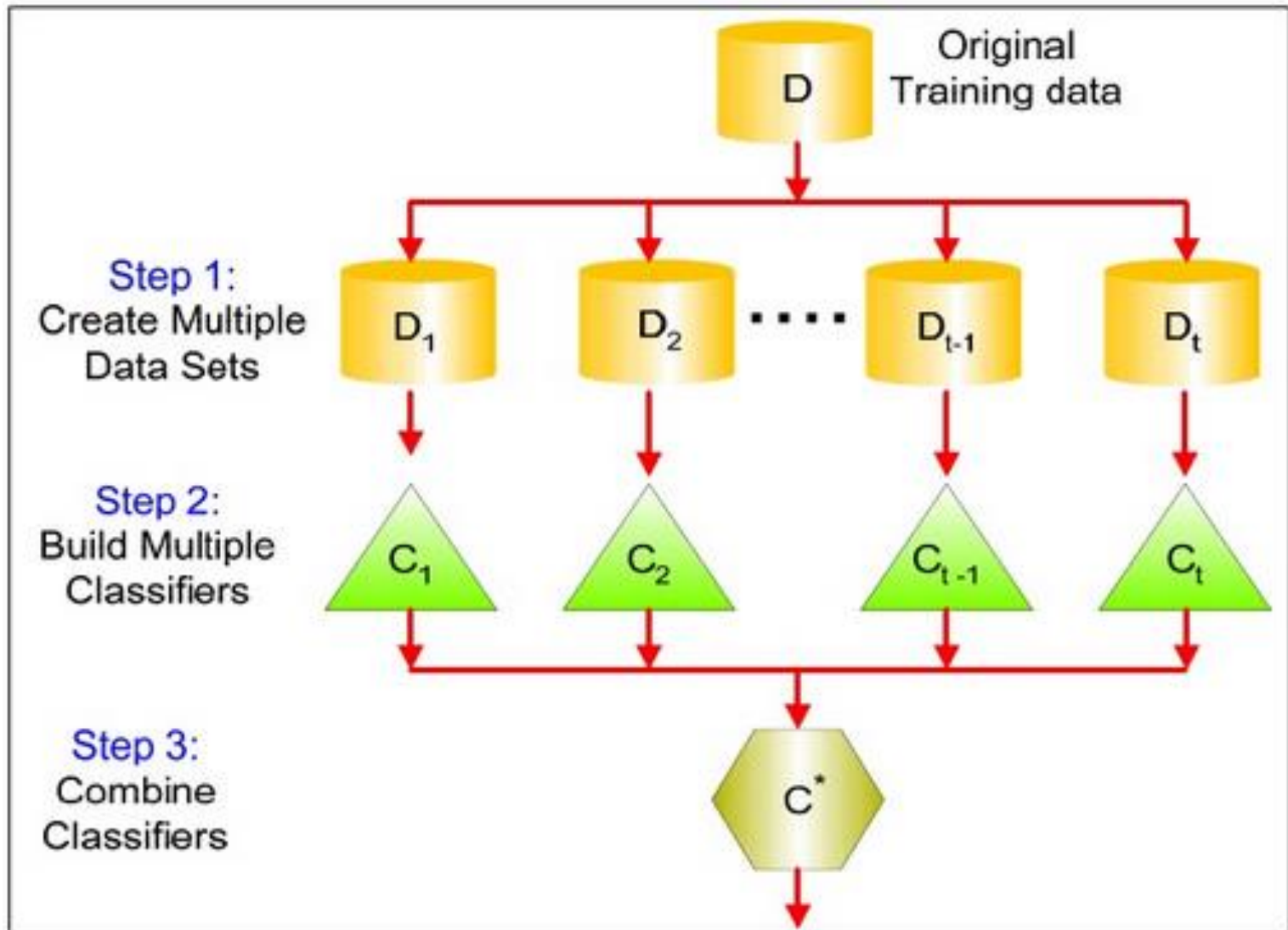
Strong learner: we seek to produce one classifier for which the classification error can be made arbitrarily small (*So far we've been looking for such methods*).

Weak learner: a classifier which is just better than random guessing (*thus, now, this will be mostly 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 .. a Generic Approach



Conditions

Training the same classifier on the same training data several times would give the same result for most machine learning algorithms (with the exception of 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.
 - For example, 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

Random Forests

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 (Combining Classifiers)

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).*

When the output is numeric (for example, a probability estimate for each class c_i):

- We can combine the d_j scores by taking their (weighted) mean, product, minimum, maximum, .. etc.

Stacking

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

Aggregation Methods (Combining Classifiers)

There are several methods to combine (aggregate) the outputs of the various classifiers.

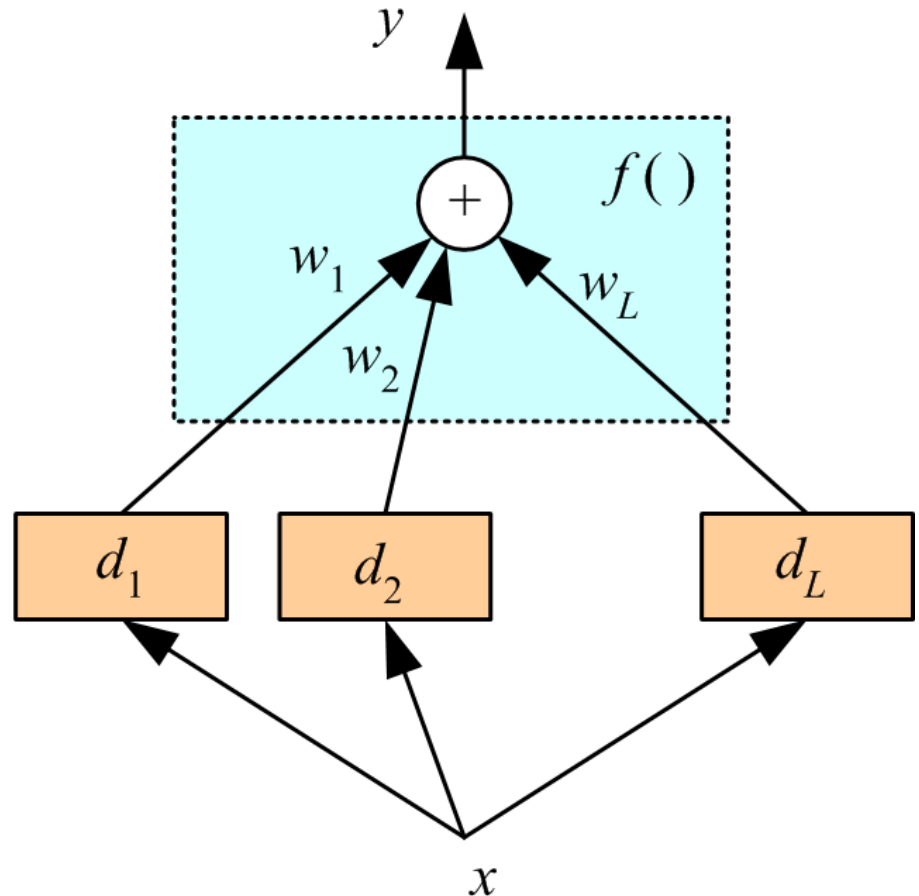
- **Linear combination**

$$y = \sum_{j=1}^L w_j d_j$$

$$w_j \geq 0 \text{ and } \sum_{j=1}^L w_j = 1$$

- **Classification**

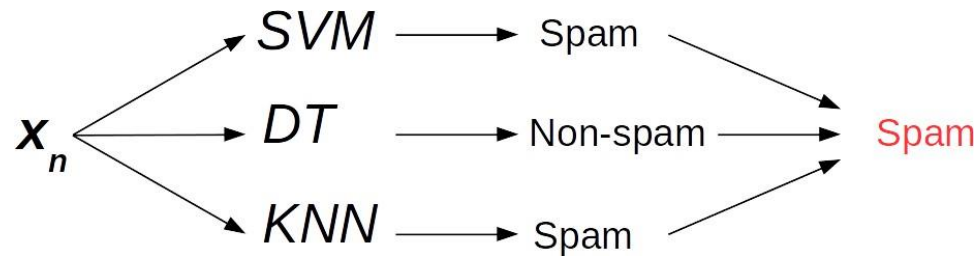
$$y_i = \sum_{j=1}^L w_j d_{ji}$$



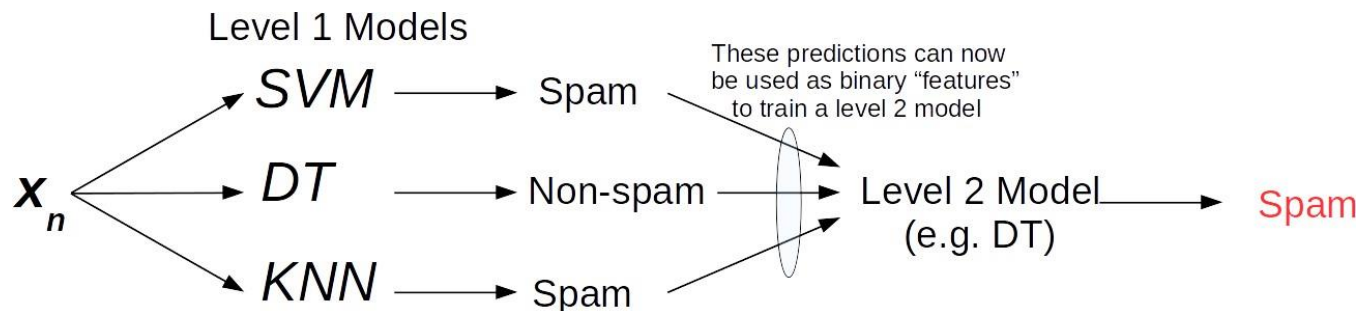
Aggregation Methods (Combining Classifiers)

Voting versus Stacking:

"Voting" or "Averaging" of predictions of multiple pre-trained models.



"Stacking": Use predictions of multiple models as "features" to train a new model and use the new model to make predictions on test data.



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.

In this lecture .. we will **concentrate on those algorithms that have been specially invented to exploit the ensemble approach:**

- They will create a large set of classifiers, 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, and Random Forests.

Bagging

– *based on* Bootstrap Resampling

Random Forests

– *based on* Decision Trees

Boosting

– Boosting by Sampling

– Boosting by Weighting

Adaboost (*Adaptive Boosting*)

Ensemble-Based Methods

specifically invented for

Ensemble Learning

Bagging .. Bootstrap + aggregating

- 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 it.
- 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 S_n , T , m ..

1. Choose T —number of trees to grow.
2. Choose m —number of variables used to split each node. $m \ll M$, 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 S_n *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.

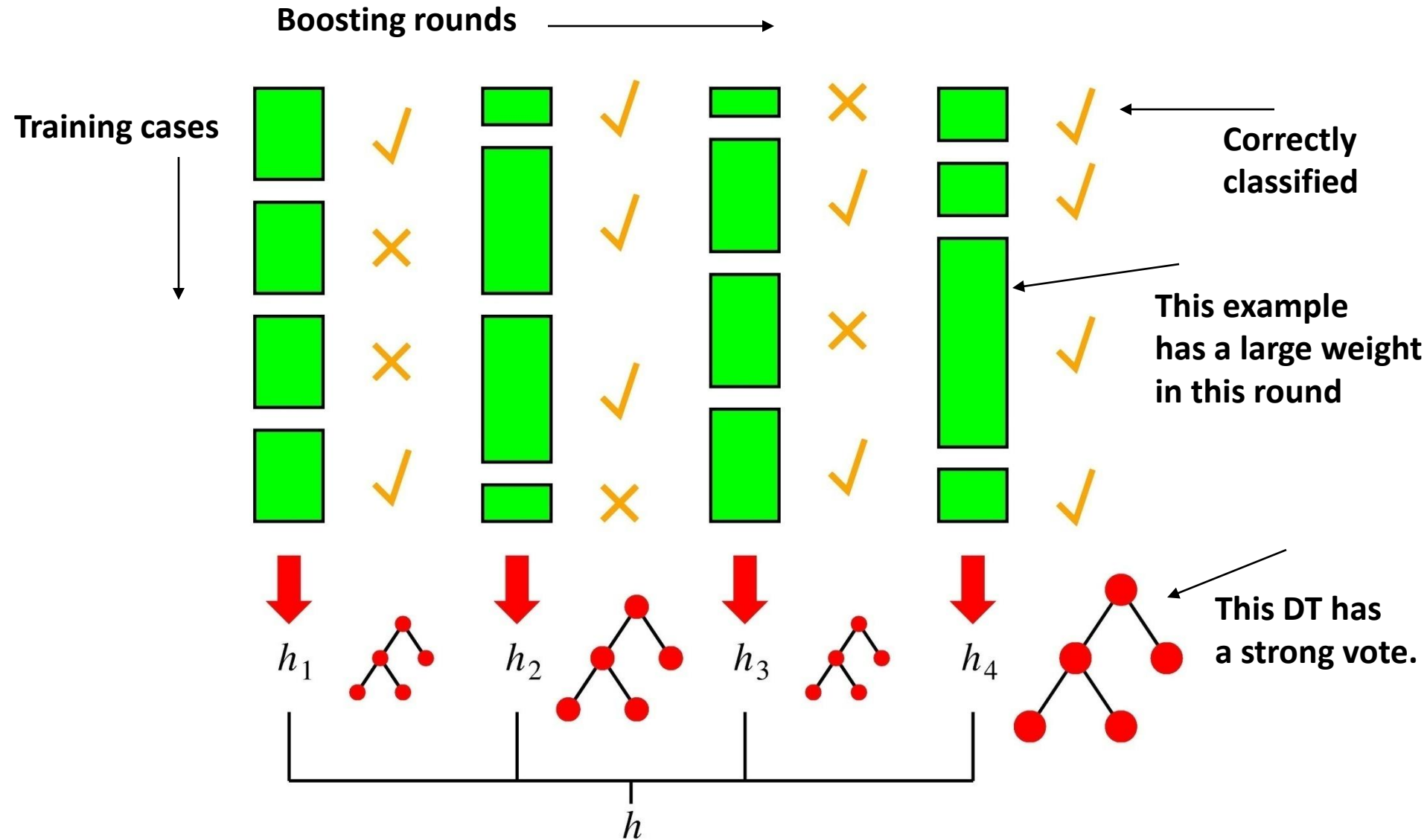
Boosting

- **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

- 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.

Boosting

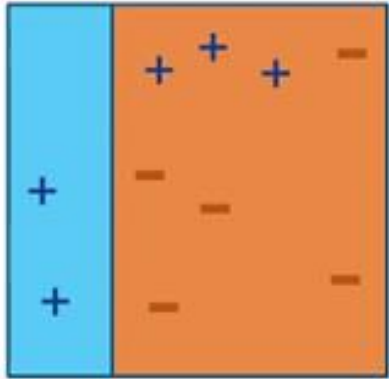


Boosting

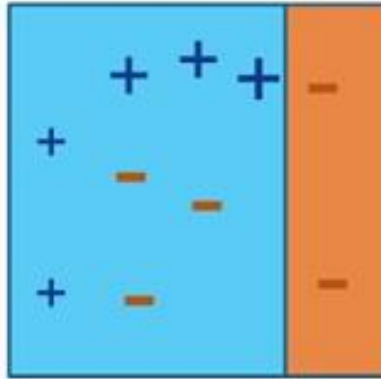
As the base learners:

- We will use decision trees with depth = 1.
- These are also known as “decision stumps”.

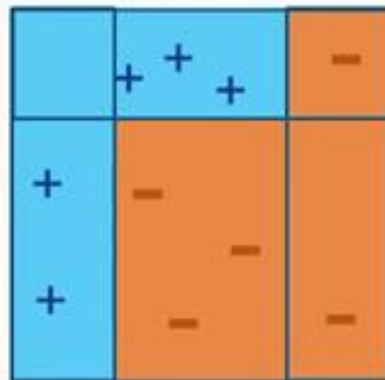
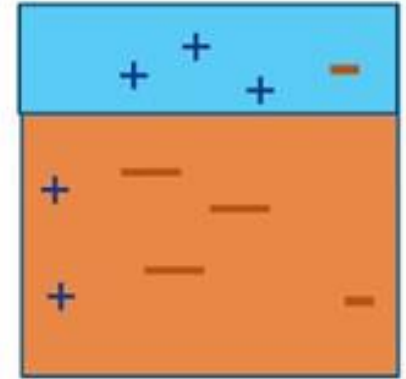
Iteration 1



Iteration 2



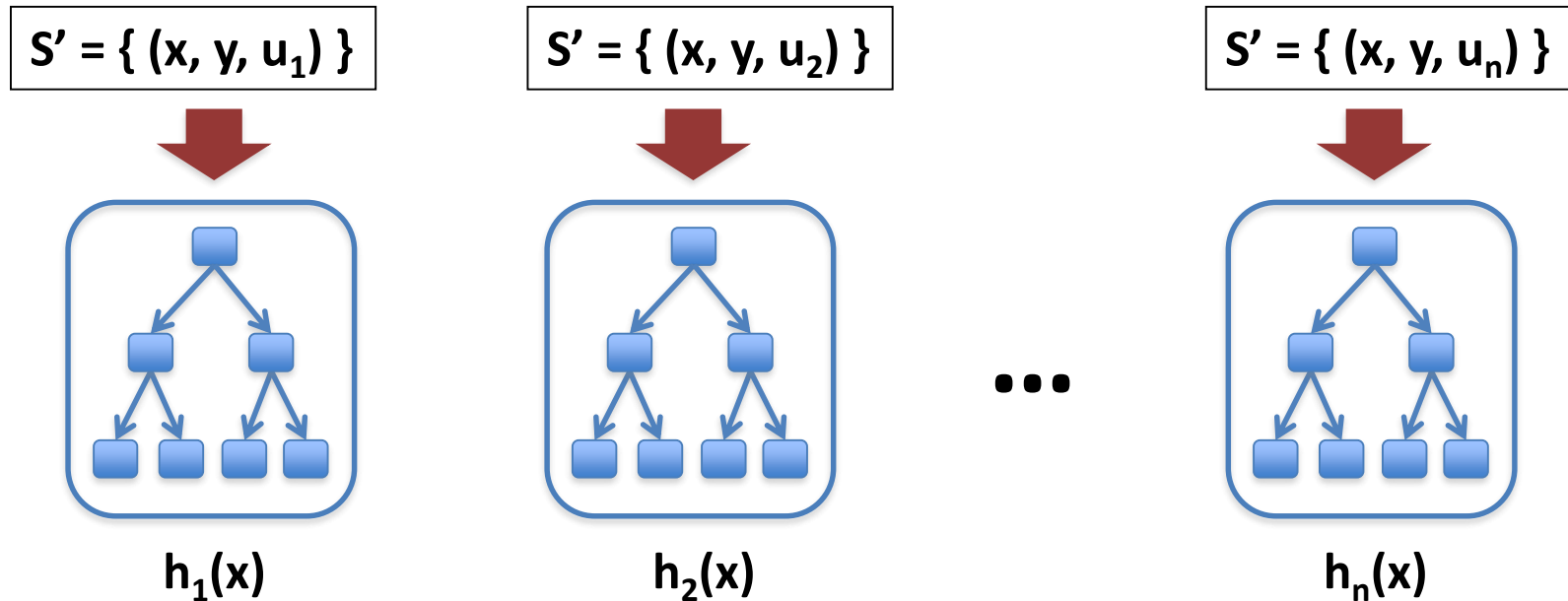
Iteration 3



Final Classifier/Strong classifier

AdaBoost .. Adaptive Boosting

$$h(x) = a_1 h_1(x) + a_2 h_2(x) + \dots + a_n h_n(x)$$



u – weighting on data points

a – weight of linear combination

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 $j = 1, \dots, L$

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 $d_j(x)$, $j = 1, \dots, L$

Calculate class outputs, $i = 1, \dots, K$:

$$y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j} \right) d_{ji}(x)$$

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.

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $D_1(i) = 1/m$ for $i = 1, \dots, m$.

← Initial Distribution of Data

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak hypothesis $h_t : \mathcal{X} \rightarrow \{-1, +1\}$.
- Aim: select h_t with low weighted error:

← Train model

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

← Error of model

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.

← Coefficient of model

- Update, for $i = 1, \dots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

← Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

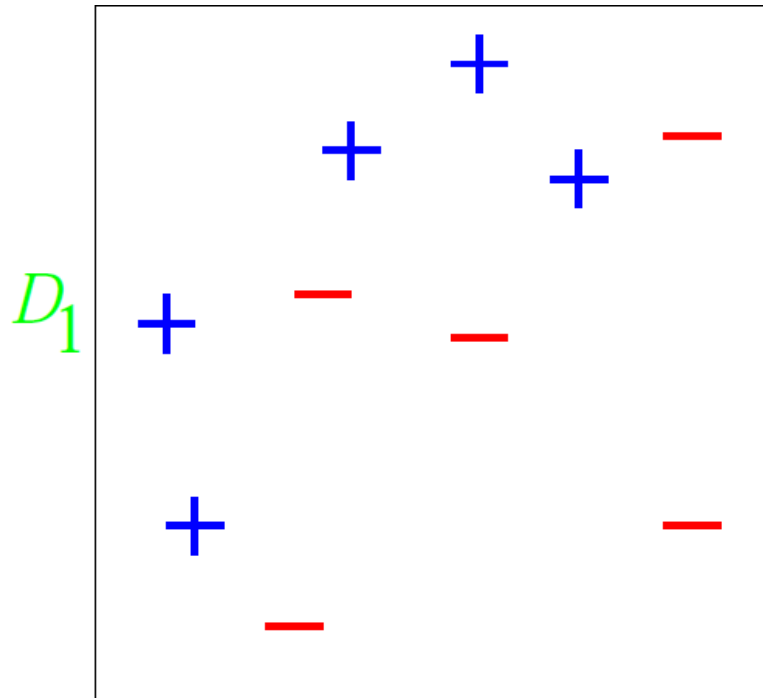
← Final average

Theorem: training error drops exponentially fast

AdaBoost .. An Example

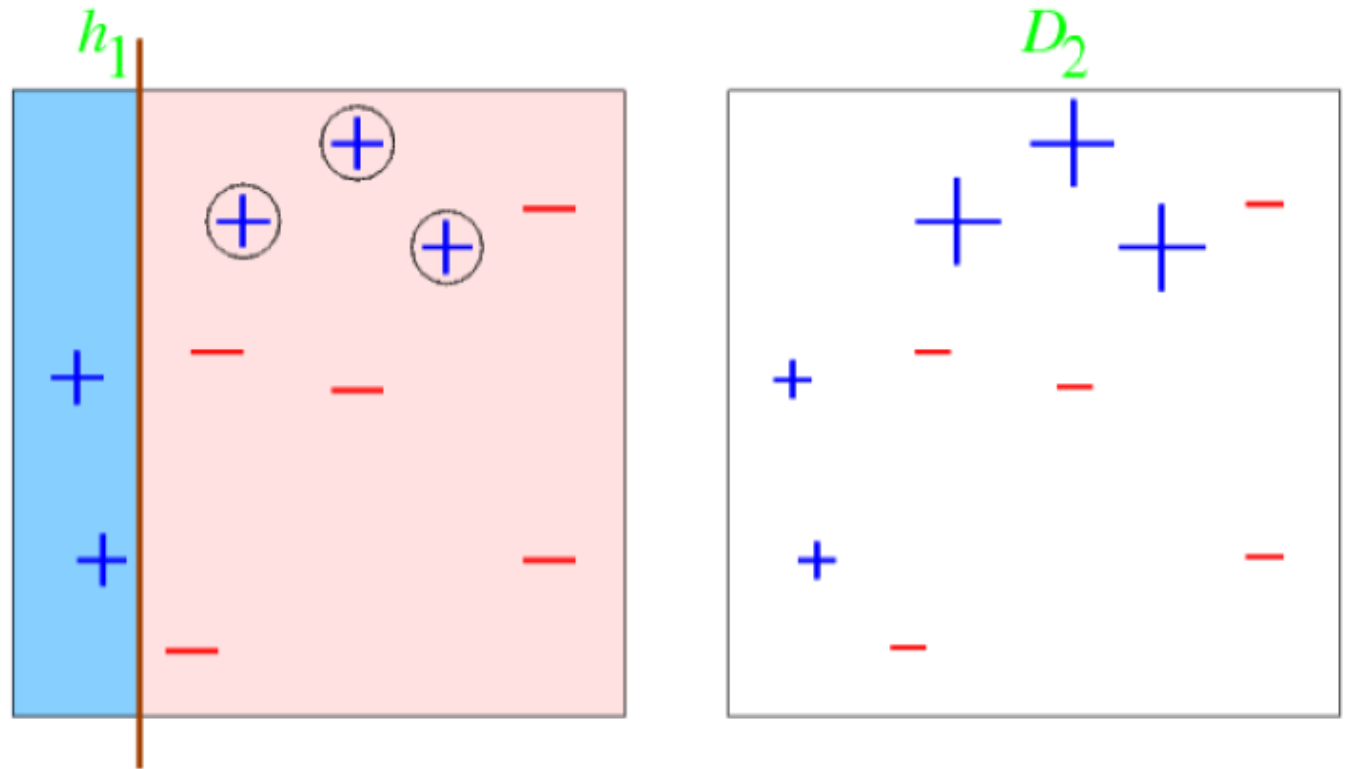
Consider binary classification with 10 training examples.

-Initial weight distribution D_1 is **uniform** (each point has equal weight = $1/10$).



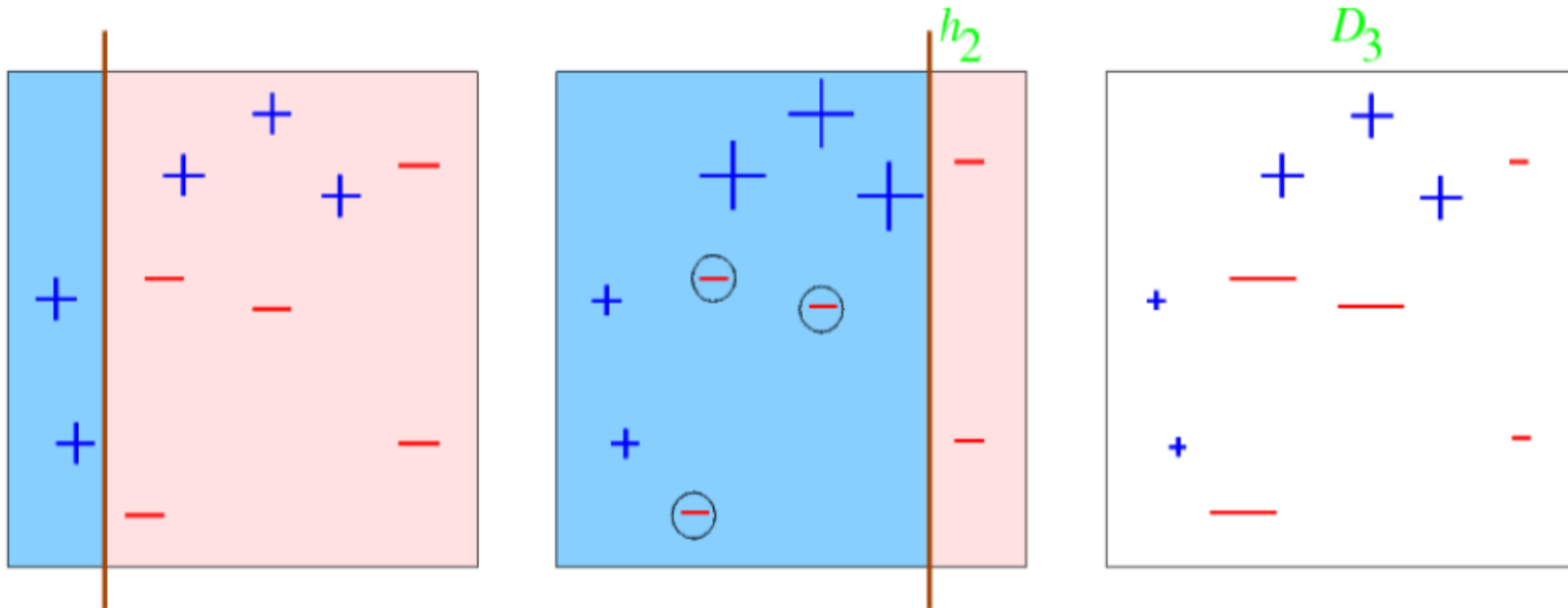
- Each of our weak classifiers will be an **axis-parallel linear classifier** ..

AdaBoost .. An Example (Round 1)



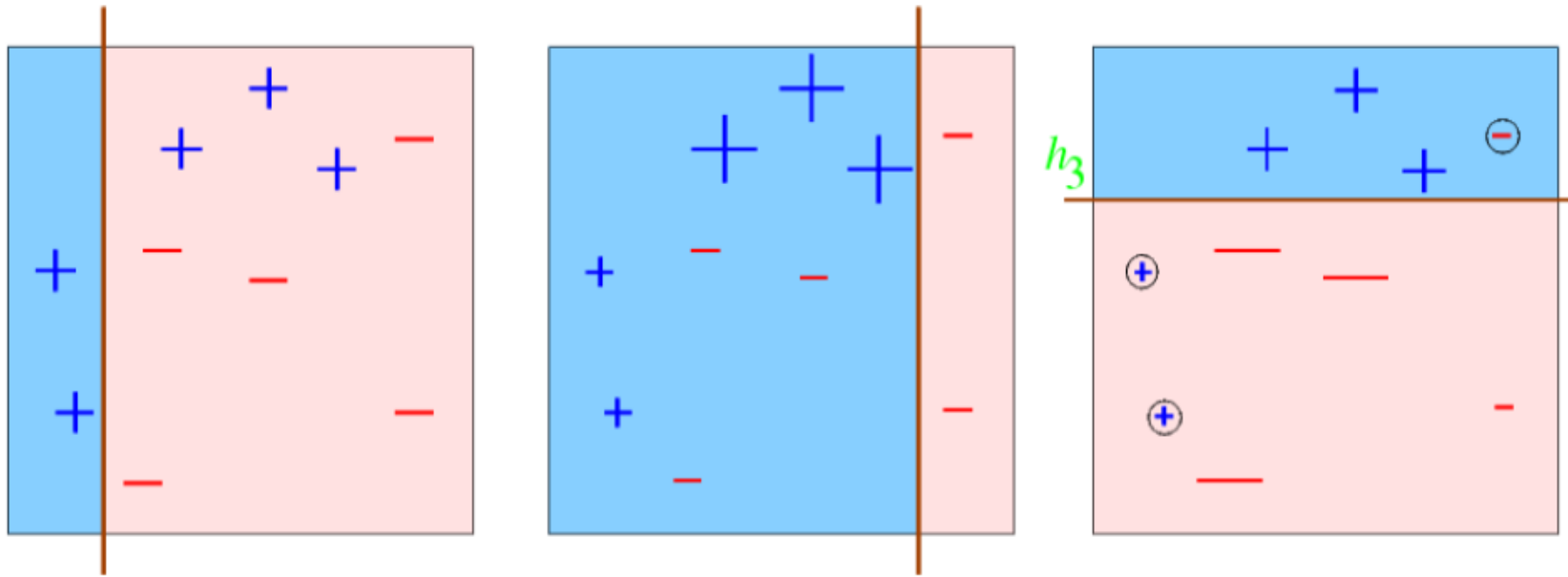
- Error rate of h_1 : $\epsilon_1 = 0.3$; weight of h_1 : $\alpha_1 = \frac{1}{2} \ln((1 - \epsilon_1)/\epsilon_1) = 0.42$
- Each **misclassified** point **upweighted** (weight multiplied by $\exp(\alpha_2)$)
- Each **correctly classified** point **downweighted** (weight multiplied by $\exp(-\alpha_2)$)

AdaBoost .. An Example (Round 2)



- Error rate of h_2 : $\epsilon_2 = 0.21$; weight of h_2 : $\alpha_2 = \frac{1}{2} \ln((1 - \epsilon_2)/\epsilon_2) = 0.65$
- Each **misclassified** point **upweighted** (weight multiplied by $\exp(\alpha_2)$)
- Each **correctly classified** point **downweighted** (weight multiplied by $\exp(-\alpha_2)$)

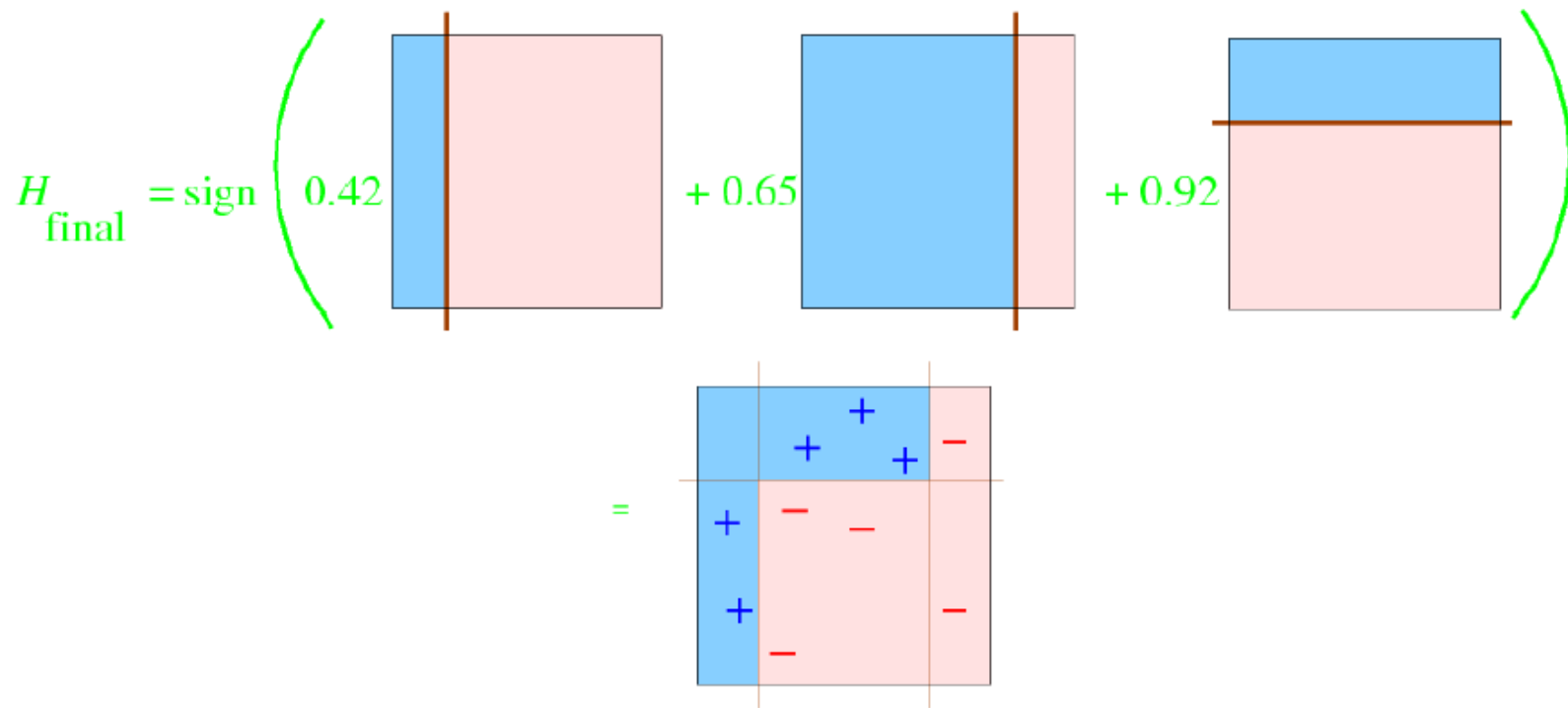
AdaBoost .. An Example (Round 3)



- Error rate of h_3 : $\epsilon_3 = 0.14$; weight of h_3 : $\alpha_3 = \frac{1}{2} \ln((1 - \epsilon_3)/\epsilon_3) = 0.92$
- Suppose we decide to stop after round 3
- Our **ensemble** now consists of 3 classifiers: h_1, h_2, h_3

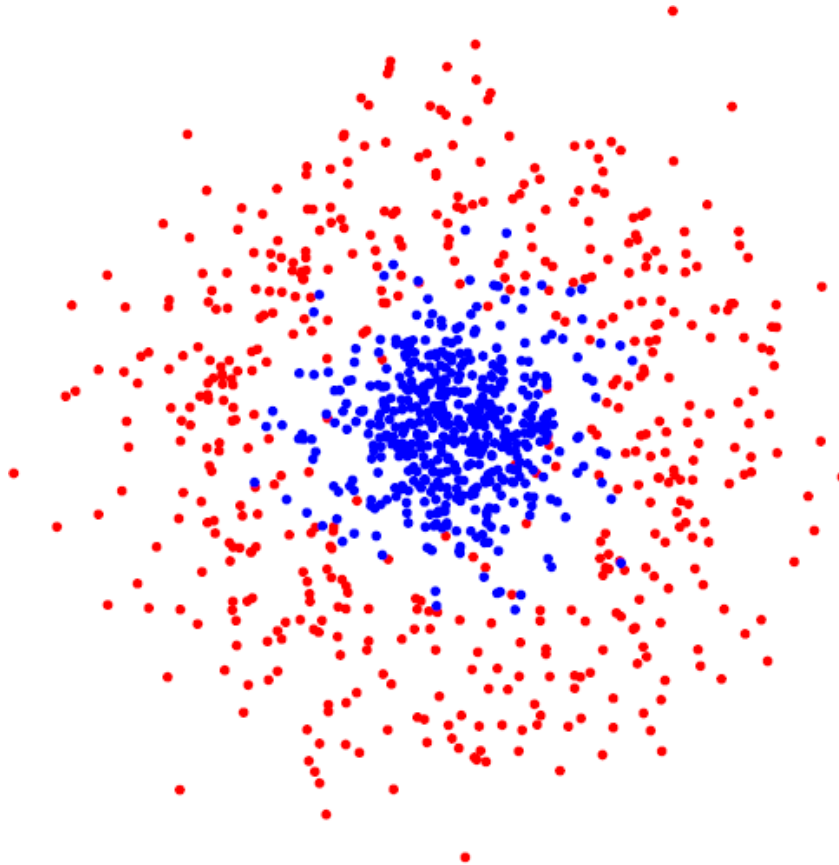
AdaBoost .. An Example (Final Classifier)

- Final classifier is a **weighted linear combination** of all the classifiers
- Classifier h_i gets a weight α_i



- Multiple **weak, linear classifiers** **combined** to give a **strong, nonlinear classifier**

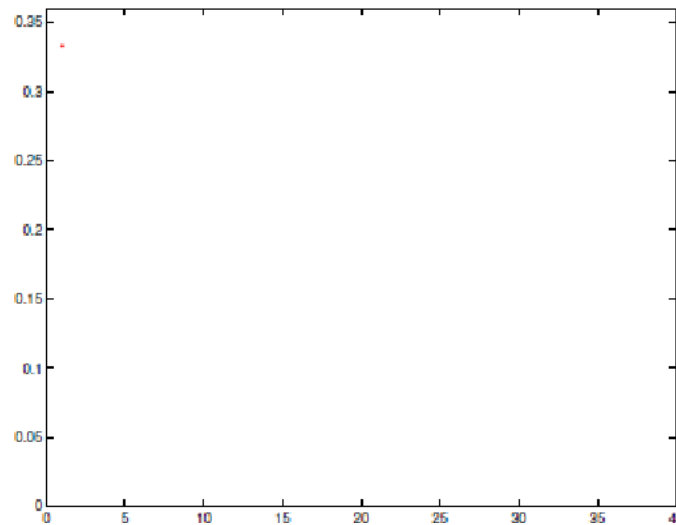
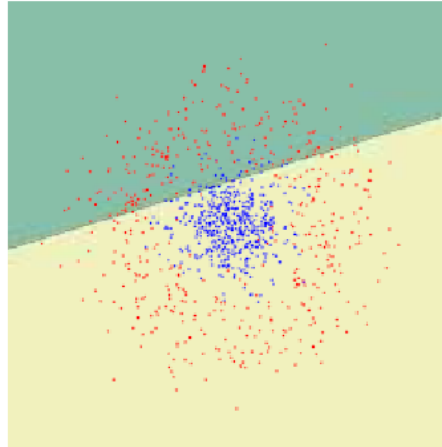
AdaBoost .. Example 2



- Given: A nonlinearly separable dataset.
- We want to use Perceptron (linear classifier) on this data.

AdaBoost .. Example 2

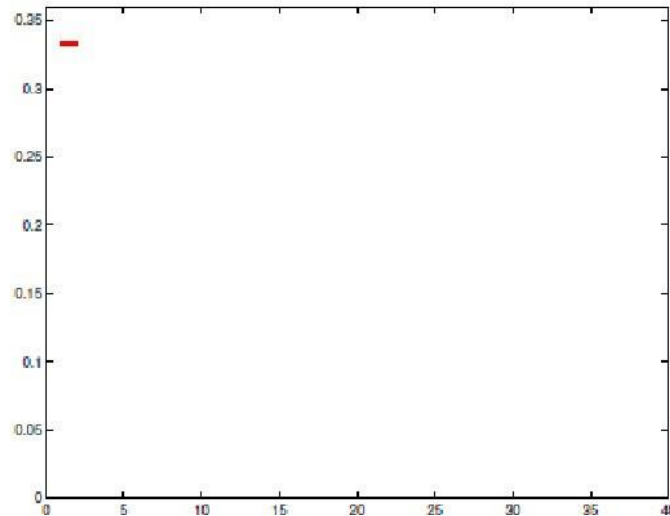
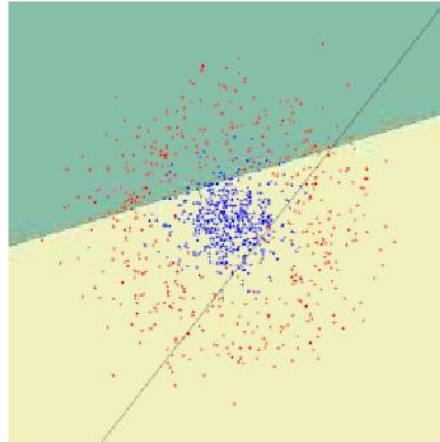
$t = 1$



After round 1, our ensemble has 1 linear classifier (Perceptron).
Bottom figure: X axis is number of rounds, Y axis is training error.

AdaBoost .. Example 2

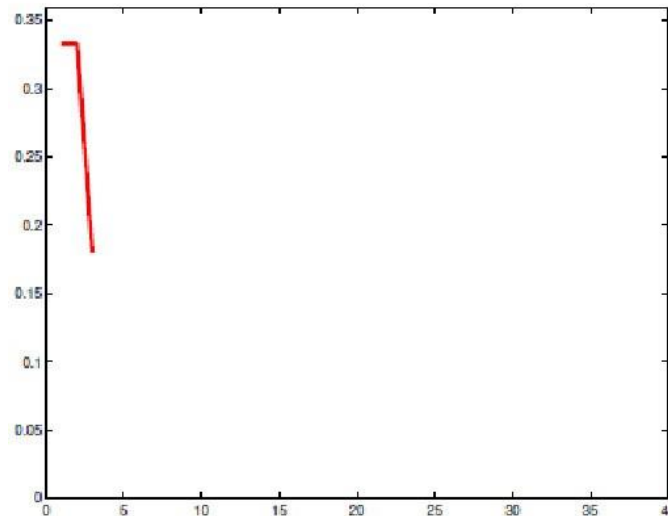
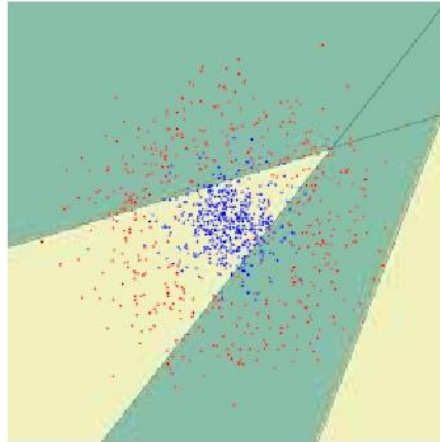
$t = 2$



After round 2, our ensemble has 2 linear classifiers (Perceptrons).
Bottom figure: X axis is number of rounds, Y axis is training error.

AdaBoost .. Example 2

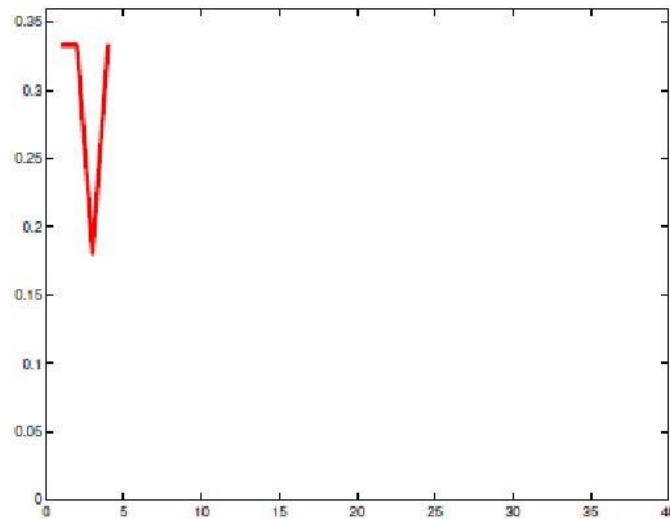
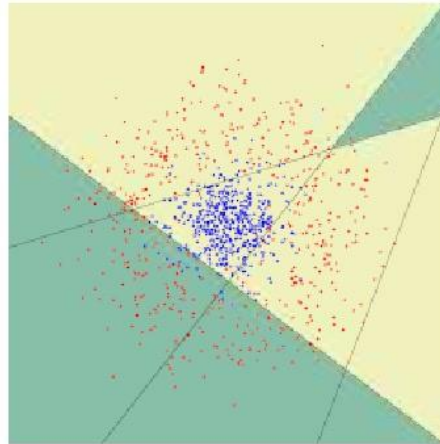
$t = 3$



After round 3, our ensemble has 3 linear classifiers (Perceptrons).
Bottom figure: X axis is number of rounds, Y axis is training error.

AdaBoost .. Example 2

$t = 4$

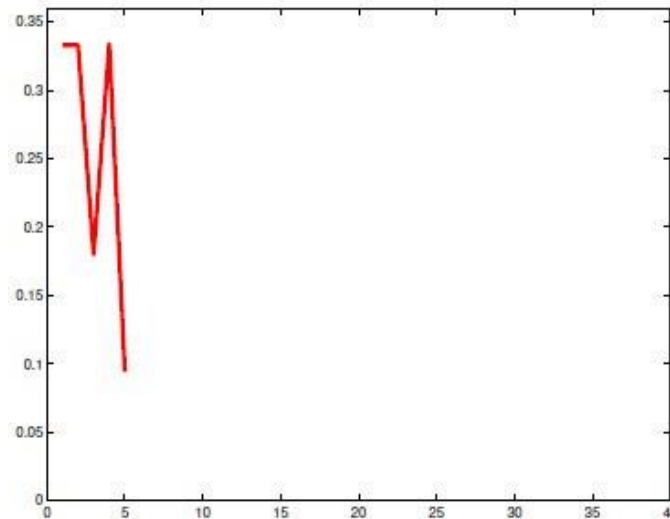
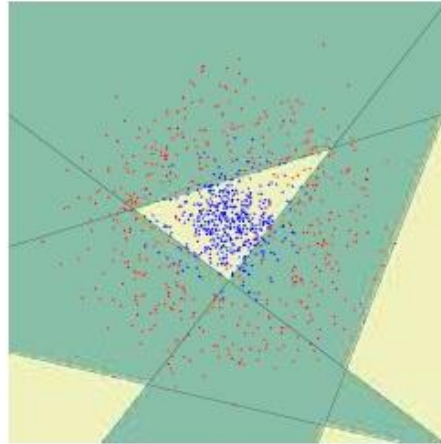


After round 4, our ensemble has 4 linear classifiers (Perceptrons).

Bottom figure: X axis is number of rounds, Y axis is training error.

AdaBoost .. Example 2

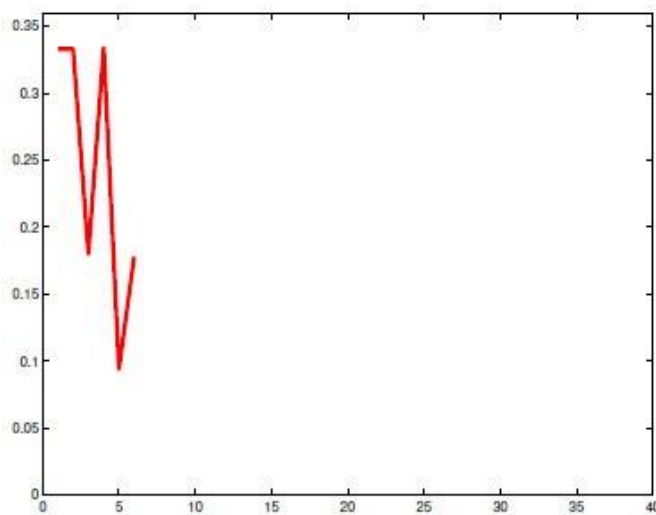
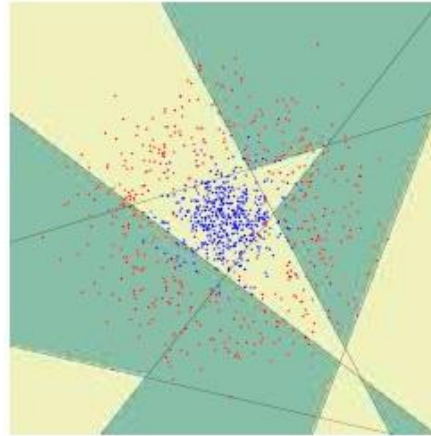
$t = 5$



After round 5, our ensemble has 5 linear classifiers (Perceptrons).
Bottom figure: X axis is number of rounds, Y axis is training error.

AdaBoost .. Example 2

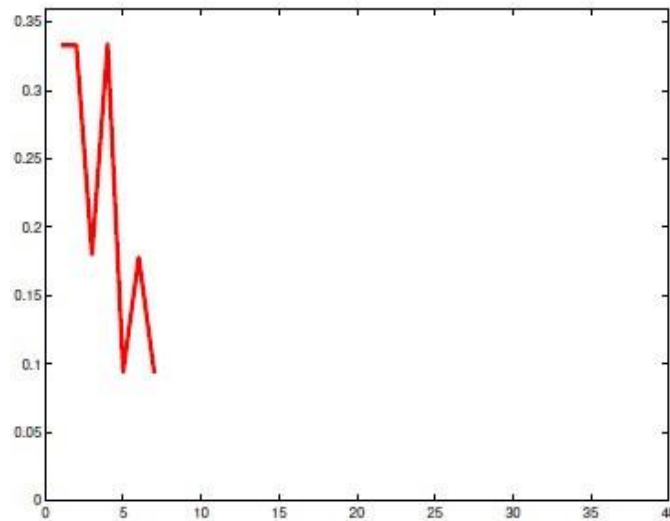
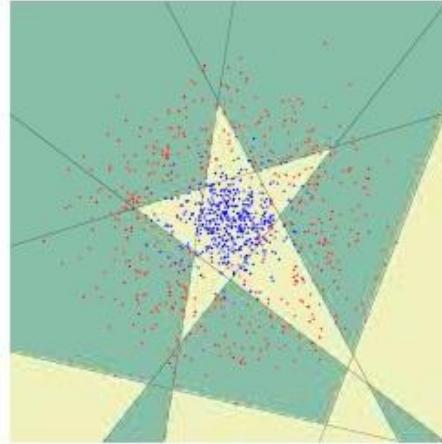
$t = 6$



After round 6, our ensemble has 6 linear classifiers (Perceptrons).
Bottom figure: X axis is number of rounds, Y axis is training error.

AdaBoost .. Example 2

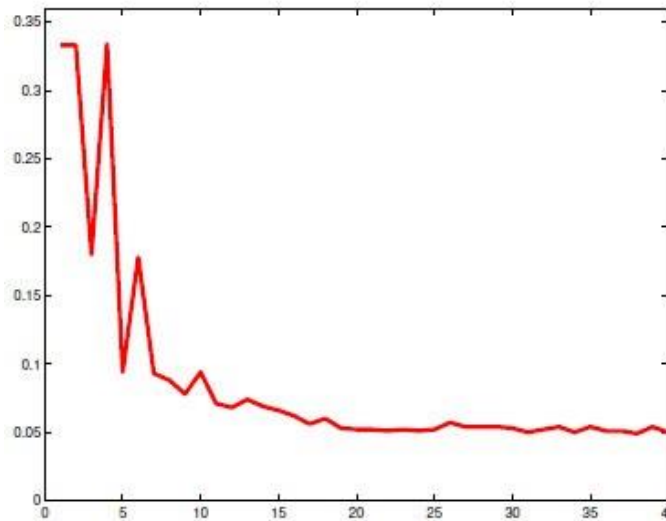
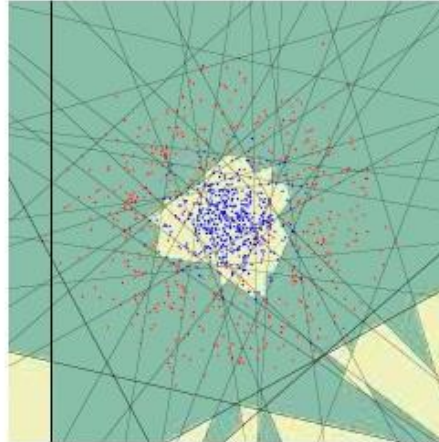
$t = 7$



After round 7, our ensemble has 7 linear classifiers (Perceptrons).
Bottom figure: X axis is number of rounds, Y axis is training error.

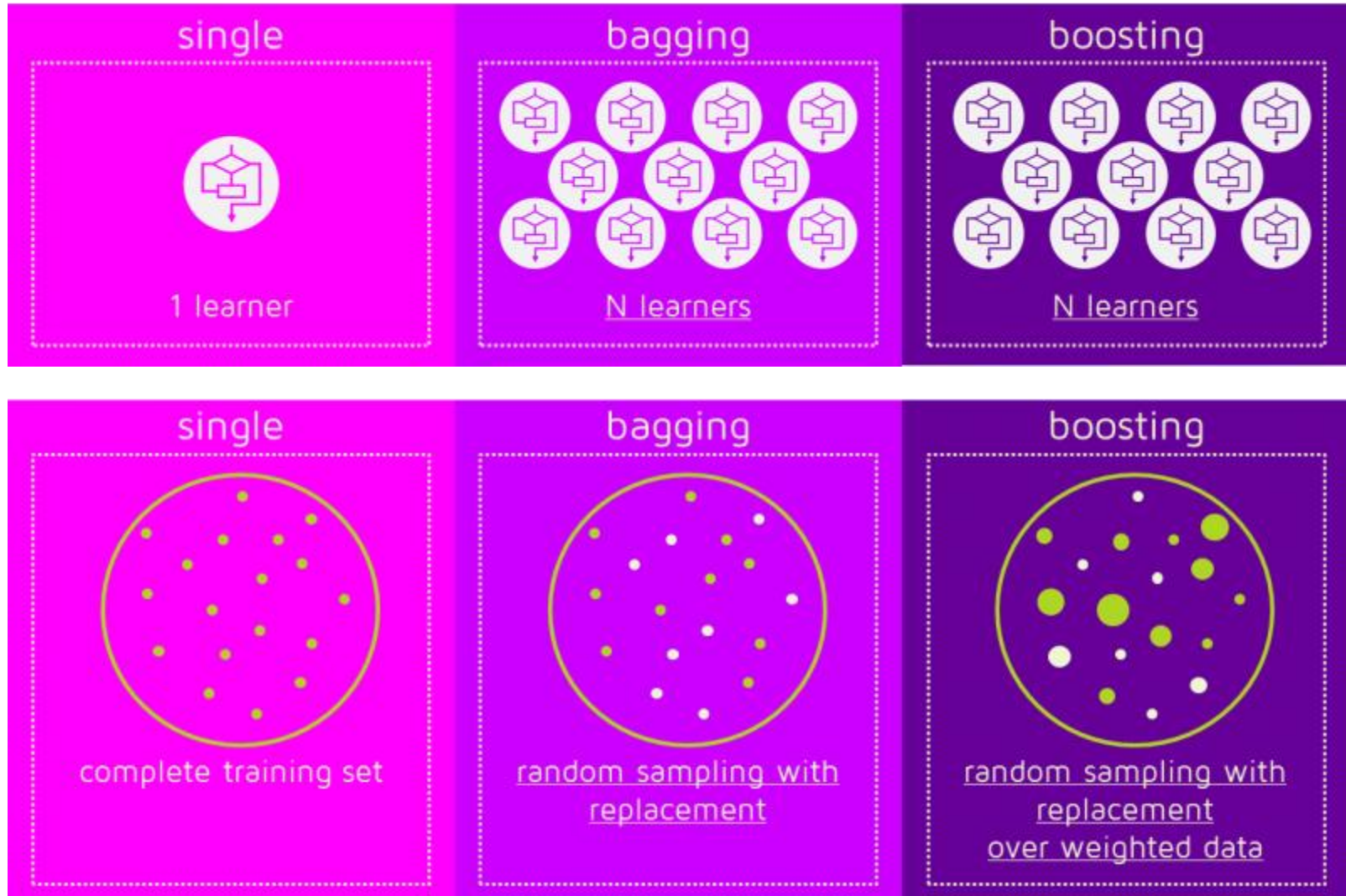
AdaBoost .. Example 2

$t = 40$

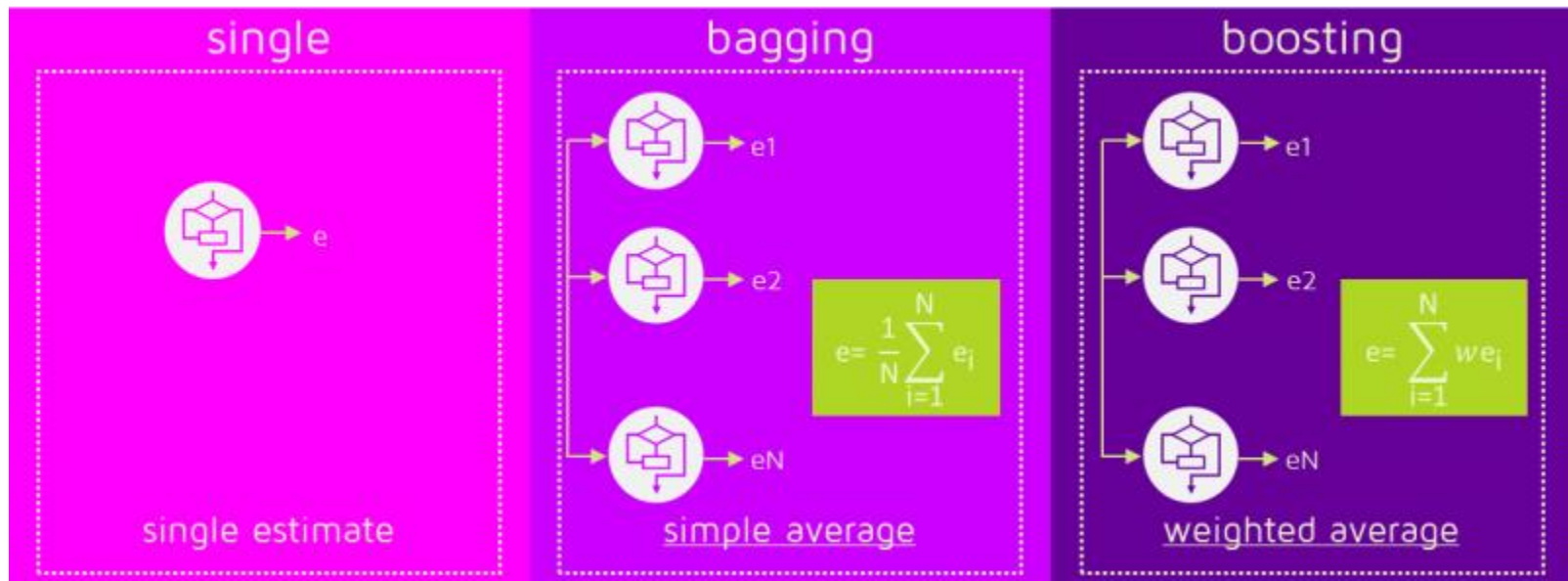
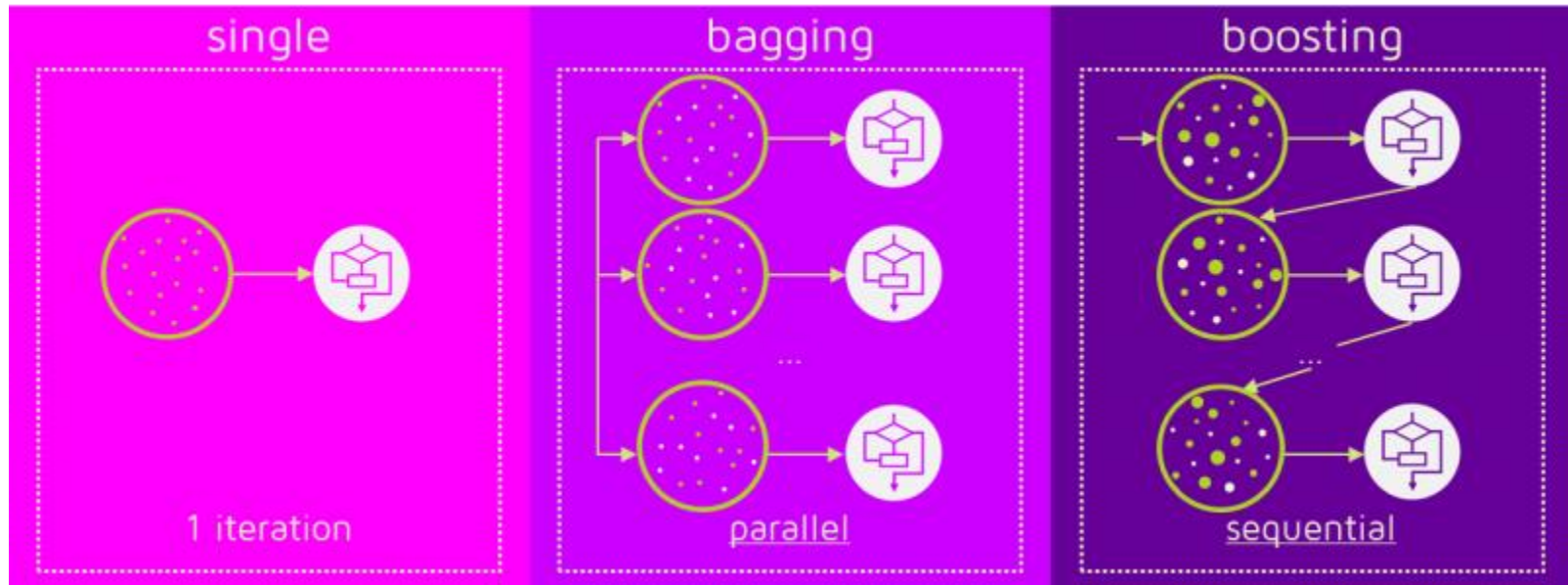


After round 40, our ensemble has 40 linear classifiers (Perceptrons).
Bottom figure: X axis is number of rounds, Y axis is training error.

Bagging *versus* Boosting



Bagging *versus* Boosting



Bagging *versus* Boosting



Thanks! ... *Questions?*