CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Cybernetics

Committees, ensembles.

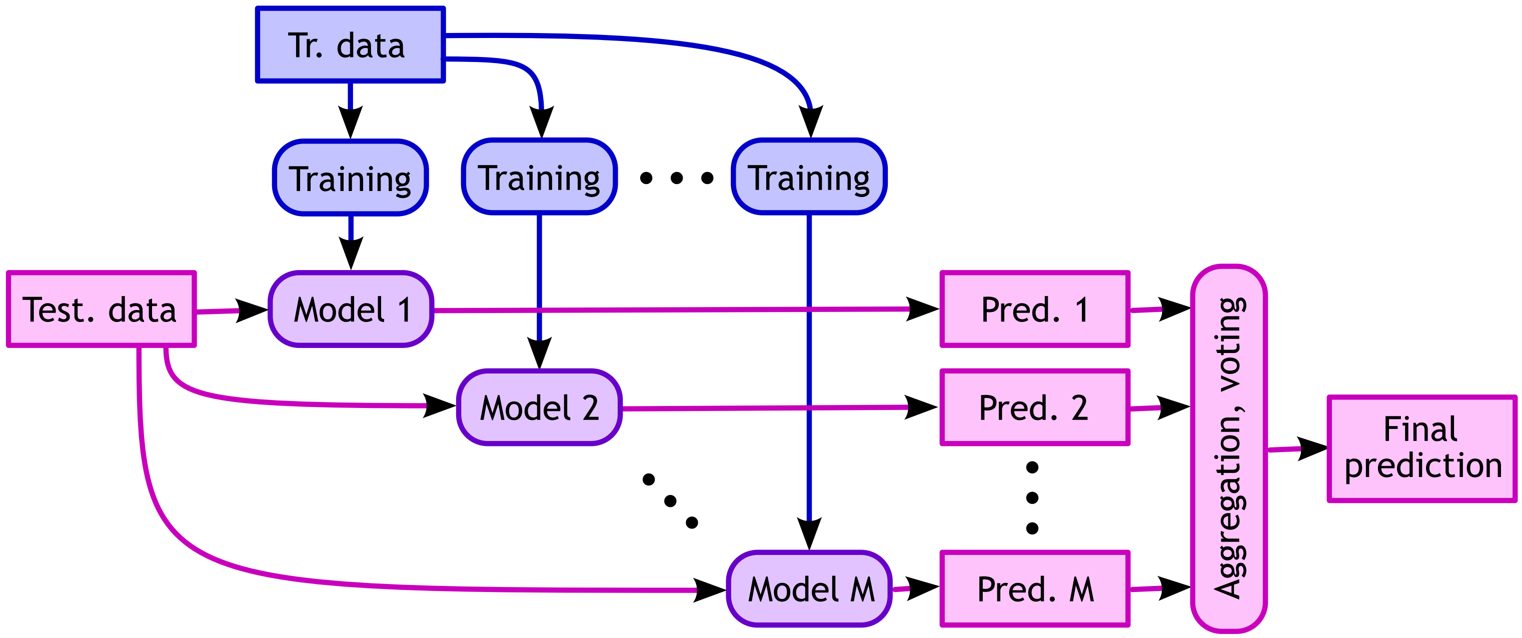
Petr Pošík

Czech Technical University in Prague
Faculty of Electrical Engineering
Dept. of Cybernetics

Introduction

Ensemble a.k.a committee

- ML model composing multiple different models to obtain better predictive performance than could be obtained from any of the constituent models.
- A way to compensate for poor learning algorithms by performing a lot of extra computations.
- Ensembles tend to yield better results when there is a significant diversity among the models (Intuition: averaging reduces variance).
- Individual ensamble/committee methods differ in the way they create individual *models different from each other*.
- Use different kinds of models, or models unstable w.r.t. a change in the training data.



Ensemble examples

Some examples of committee/ensemble methods:

- Stacking
- Bagging
- Random forests
- Boosting
- ...

Decision trees (classification and regression) are used most often as the base models because

- they are relatively fast to learn,
- they are unstable w.r.t. the changes in the training dataset, and thus
- it is quite easy to make a lot of trees which are very diverse.

Aggregation

The final aggregation of results of individual models is usually done by

- (weighted) *voting* of individual models for *classification* problems,
- (weighted) *averaging* of individual models for *regression* problems,
- or by other techniques.

Aggregation

The final aggregation of results of individual models is usually done by

- (weighted) *voting* of individual models for *classification* problems,
- (weighted) *averaging* of individual models for *regression* problems,
- or by other techniques.

Stacking

- Assume we have M different models h_m created for the same modeling task, each being a function $h_m(\boldsymbol{x})$ of the input features $\boldsymbol{x}$.
- The predictions of these models, $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \ldots, h_M(\boldsymbol{x}))$, may be considered new features extracted from the data set (basis expansion).
- We can thus train a higher-level classification/regression model h_{stack} as a function of these new features, i.e. $h_{\text{stack}}(\boldsymbol{h})$ (sometimes together with the original features, i.e. $h_{\text{stack}}(\boldsymbol{x}, \boldsymbol{h})$).
- For classification, logistic regression is often used as h_{stack}.
- For regression, multiple linear regression is often used as h_{stack} with the constraint on the weights w_i such that $\sum w_i = 1$ and $w_i > 0\ \forall i$.
- An obvious way to estimate the weights $\boldsymbol{w}$ as $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{|T|} L\left(y_i, \sum_{m=1}^{M} w_m h_m(\boldsymbol{x}_i)\right)$, however, can result in overfitting; this is solved by LOO cross-validation, i.e. using the estimate $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{|T|} L\left(y_i, \sum_{m=1}^{M} w_m \hat{h}_m^{-i}(\boldsymbol{x}_i)\right)$, where $\hat{h}_m^{-i}$ is a predictor obtained by training on data excluding $(\boldsymbol{x}_i, y_i)$, i.e. at the price of high-computational demands.

Bagging

Bootstrapping

- A general statistical technique for assessing the accuracy of parameter estimates and for hypotheses testing.
- It relies on *many repetitions* and *random sampling with replacement*.

Bootstrapping

- A general statistical technique for assessing the accuracy of parameter estimates and for hypotheses testing.
- It relies on *many repetitions* and *random sampling with replacement*.

Example: Assume we want to estimate the average height of all the people in the world. How to do that?

- Cannot measure the whole population, measure just a sample of N people.
- Using this sample, we can obtain a (single) point estimate of the average population height: $\hat{h} = \frac{1}{N} \sum_{i=1}^{N} h_i$.
- We also need some measure of uncertainty/variability of this estimate. How to do that?
- Use "classic" statistics: compute the sample variance $\hat{s}_h^2$ and compute the variance of the estimate as $\hat{s}_{\hat{h}}^2 = \frac{\hat{s}_h^2}{N}$, or:
- Use *bootstrapping*:
  1. Repeat M times ($M = 10^2, \ldots, 10^6$):
     - Create a bootstrap sample from the original dataset.
     - Compute bth estimate of the statistic (here average) from the bootstrap sample.
  2. Now you have a histogram of the estimates (here averages), from which you can estimate the mean, variance, ... of the sampling distribution.

Similar process works for many other estimators.

Bootstrap sample

Assume we have a dataset T with N items. What is the **bootstrap sample** T^b?

- A pertubed version of the original dataset T.
- Each item of T^b was chosen uniformly with replacement from the original dataset T. Usually, $|T| = N = |T^b|$.
- Some items of T are copied to T^b more than once. Some items are not copied at all.

Bootstrap sample

Assume we have a dataset T with N items. What is the **bootstrap sample** T^b?

- A pertubed version of the original dataset T.
- Each item of T^b was chosen uniformly with replacement from the original dataset T. Usually, $|T| = N = |T^b|$.
- Some items of T are copied to T^b more than once. Some items are not copied at all.

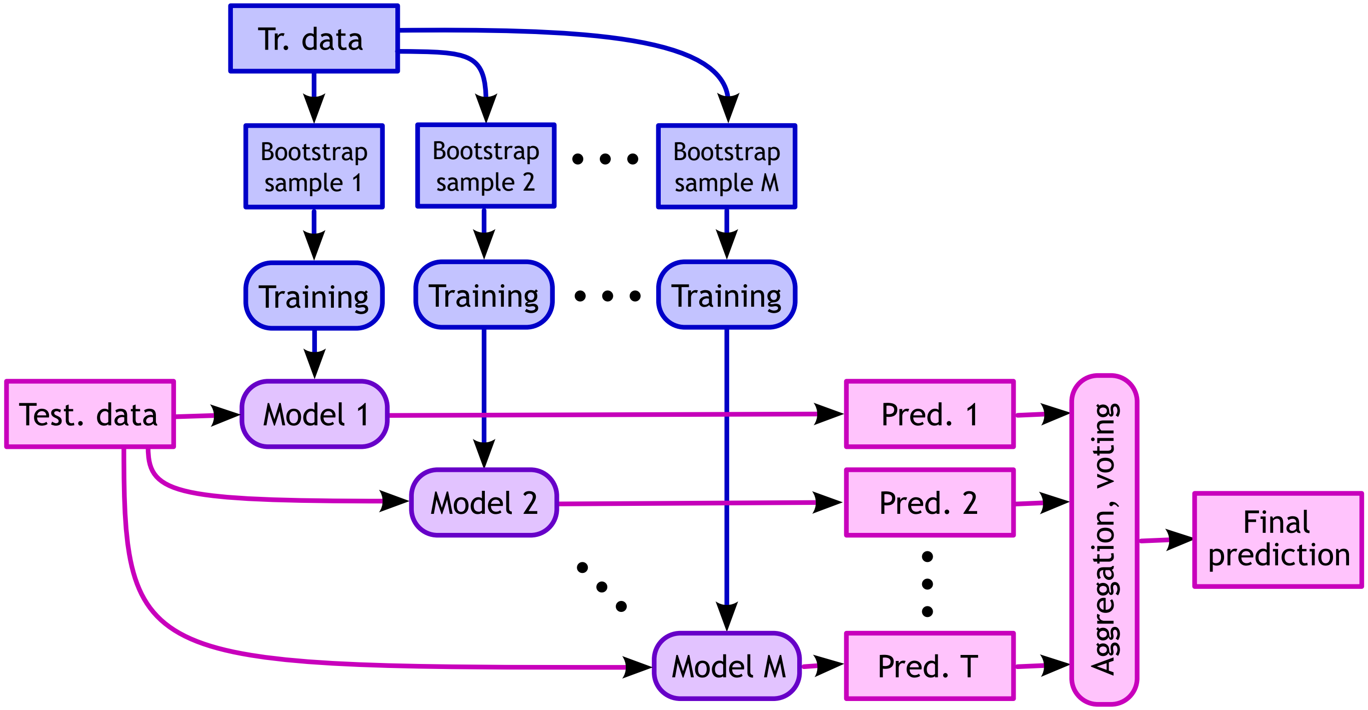
How many unique elements of T are present in T^b (on average)?

- Probability that a particular item will not be chosen in one particular pick: $1 - \frac{1}{N}$
- Probability that a particular item will not be chosen in any of N picks: $\left(1 - \frac{1}{N}\right)^N$
- The expected number of items that will not be copied to a bootstrap sample: $N\left(1 - \frac{1}{N}\right)^N \approx Ne^{-1} = N \cdot 0.368$
- The expected number of unique elements copied from T: $N\left(1 - \left(1 - \frac{1}{N}\right)^N\right) \approx N\left(1 - e^{-1}\right) = N \cdot 0.632$

Bagging a.k.a. Bootstrap aggregation

- Uses bootstrap to improve the estimate or the prediction itself.
- Aggregating results of several models reduces variance and prevents overfitting.
- Algorithm:
  1. Create M bootstrap samples T^i from training data T ($i = 1, \ldots, M$).
  2. Build a model h_i on each bootstrap sample T^i.
  3. Construct final model by averaging/voting the predictions of individual models:

$$\hat{y} = h_{\text{bag}}(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} h_i(\boldsymbol{x}), \text{ resp. } \hat{y} = h_{\text{bag}}(\boldsymbol{x}) = \arg\max_{y \in C} \sum_{i=1}^{M} I(y = h_i(\boldsymbol{x}))$$



Introduction
Bagging
• Bootstrapping
• Bootstrap sample
• Bagging
• Features
Random forests
Boosting
Summary

Features

Bagging

- leads to improvements for unstable procedures (artificial neural networks, classification and regression trees, etc.), but
- it can mildly degrade the performance of stable methods such as K-nearest neighbors.

- Thanks to bootstrapping, it can provide not only predictions, but also estimates of uncertainty of those predictions.

Estimate of prediction error (out-of-bag error):

- Around 37 % of training examples are not part of a bootstrap sample; they are called OOB (out of bag).
- We can predict the model response for each training sample $\boldsymbol{x}_i$ using only the models that did not have $\boldsymbol{x}_i$ in their bootstrap sample.
- We can average these predicted responses (regression) or can take a majority vote (classification) to get a single "OOB prediction" for the each observation.
- OOB predictions then can be used to compute OOB estimate of the error.
- With M sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

Random forests

Random forest (RF)

An ensamble method using set of decision trees (i.e. forest):

- Trees that are grown very deep tend to learn highly irregular patterns: they *overfit* their training sets, i.e. have *low bias*, but *very high variance*.
- RF perform averaging of multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.

RF combine

- bagging, and
- random subspace method (see below).

Predictions are computed using voting/averaging.

To train a single tree, RF algorithm

- creates a bootstrap sample of the training data (bagging), and
- uses a modified tree-learning algorithm which considers only *a random subset of input features* at each candidate split in the learning process ("feature bagging"; this further decorrelates the resulting trees). Suggestions:
  - Classification: consider $\sqrt{D}$ features at each split.
  - Regression: consider $D/3$ features at each split, use minimum node size of 5.
- In *ExtraTrees* (extremely randomized trees), instead of searching for the locally optimal split for each variable, a random value is used for the split.

RF features

Estimate of prediction uncertainty and OOB error:

- See bagging.

Variable importance:

1. Grow the forest. Compute OOB error for each data point averaged over the whole forest.
2. To measure the importance of jth variable, permute its values, and compute OOB error on this perturbed dataset. Compute the difference of the estimates before and after permutation.
3. The larger the difference, the larger the importance of variable j.

Boosting

Boosting

Hypothesis Boosting Problem

- If there exists an efficient algorithm able to create *weak classifiers* (i.e. classifiers only slightly better than random guessing), does it also mean that there is an efficient algorithm able to build *strong classifiers* (i.e. classifiers with an arbitrary precision)?
- No constraint on the algorithm.

Boosting

Hypothesis Boosting Problem

- If there exists an efficient algorithm able to create *weak classifiers* (i.e. classifiers only slightly better than random guessing), does it also mean that there is an efficient algorithm able to build *strong classifiers* (i.e. classifiers with an arbitrary precision)?
- No constraint on the algorithm.

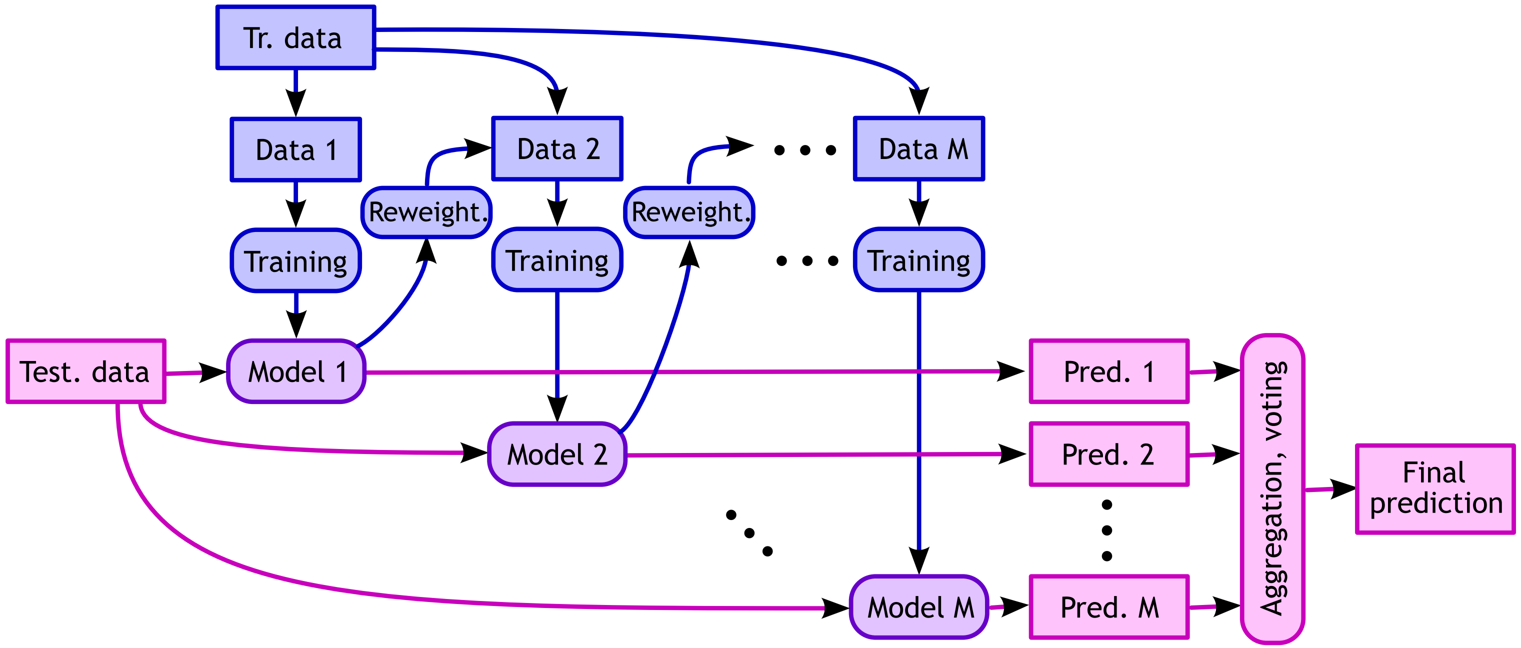
Most (not all) **Boosting algorithms**

- sequentially learn weak classifiers using weighted training set (using information from previous trees),
- construct the final strong classifier as a weighted sum of the weak classifiers,
- assign the weights to individual weak learners depending on their accuracy,
- re-weight the training data for another round of the weak learner,
- differ in the way how they weight the training data and/or the individual weak classifiers.

AdaBoost (informally)

AdaBoost

- Training data:
  - In each iteration $t = 1, \ldots, M$, it uses different weights $w_t(i)$ of the training examples $\boldsymbol{x}_i$.
  - *Misclassified examples get a larger weight* for the next iteration.
- The resulting classifier:
  - Weighted voting.
  - More accurate models get larger weight.



AdaBoost.M1

- AdaBoost for classification problem and weak learners with class label as output.
- A slightly different version exists for weak learners with output in the form of class probabilities.

Algorithm 1: AdaBoost.M1

Input: Training set of labeled examples: $T = \{\boldsymbol{x}_i, y_i\}, \boldsymbol{x}_i \in \mathcal{R}^D, y_i \in \{+1, -1\}, i = 1, \ldots, |T|$

Output: Final classifier $H_{\text{final}}(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(\boldsymbol{x})\right)$

begin

  Initialize the weights of training examples: $w_1(i) = \frac{1}{|T|}$.

  for $m = 1, \ldots, M$ **do**

    Train a weak classifier h_m using T with weights w_m.

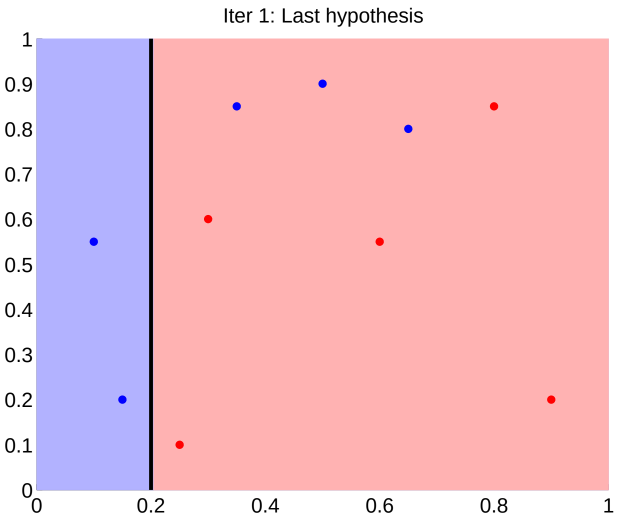
    Compute the weighted error: $\epsilon_m = \dfrac{\sum_{m=1}^{|T|} w_m(i) I\left(y_i \neq h_m(\boldsymbol{x}_i)\right)}{\sum_{m=1}^{|T|} w_m(i)}$

    Compute the weight of classifier h_m: $\alpha_m = \ln\left(\dfrac{1-\epsilon_m}{\epsilon_m}\right) > 0$

    Update the weights of the training examples: $w_{m+1}(i) = w_m(i) \cdot \exp\left[\alpha_m I\left(y_i \neq h_m(\boldsymbol{x}_i)\right)\right]$.

AdaBoost.M1 graphically

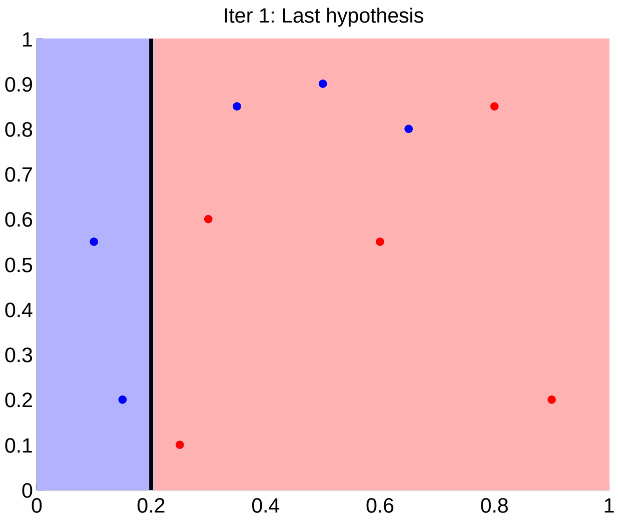
Iteration 1:



$$\epsilon_1 = 0.3$$
$$\alpha_1 = 0.42$$

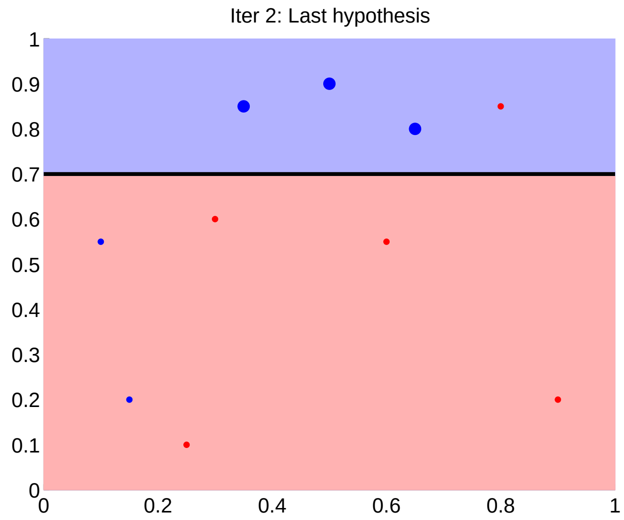
AdaBoost.M1 graphically

Iteration 1:



$$\epsilon_1 = 0.3$$
$$\alpha_1 = 0.42$$

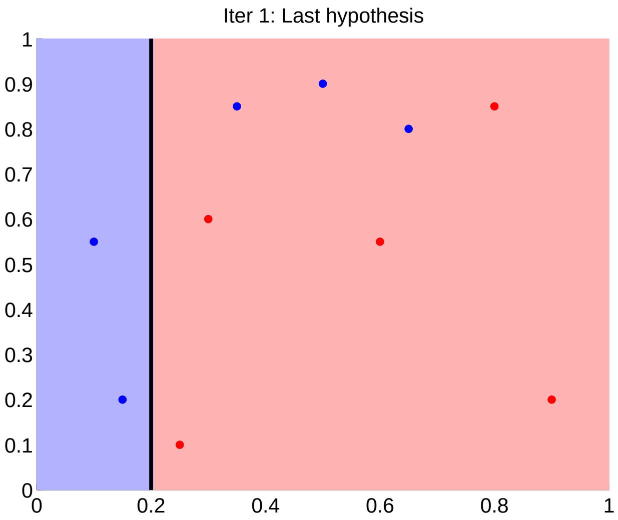
Iteration 2:



$$\epsilon_2 = 0.21$$
$$\alpha_2 = 0.65$$

AdaBoost.M1 graphically

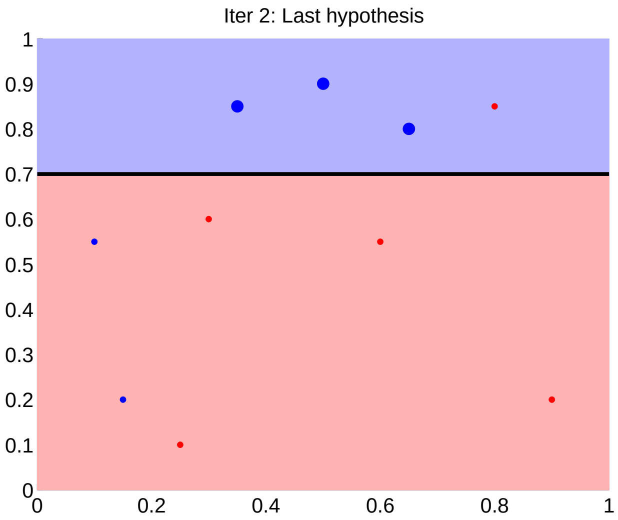
Iteration 1:



$\epsilon_1 = 0.3$

$\alpha_1 = 0.42$

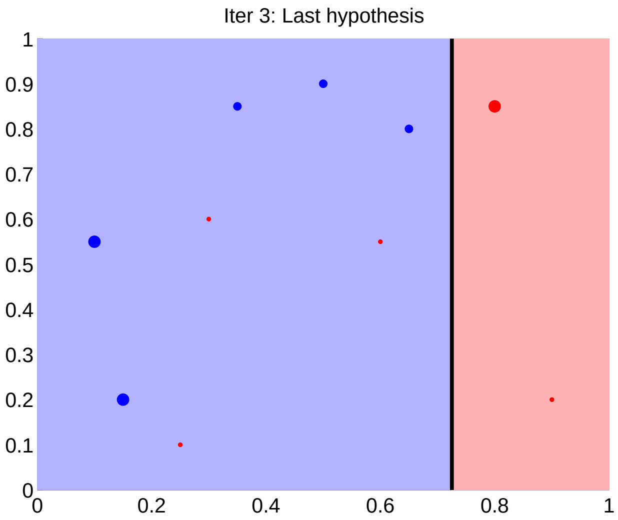
Iteration 2:



$\epsilon_2 = 0.21$

$\alpha_2 = 0.65$

Iteration 3:

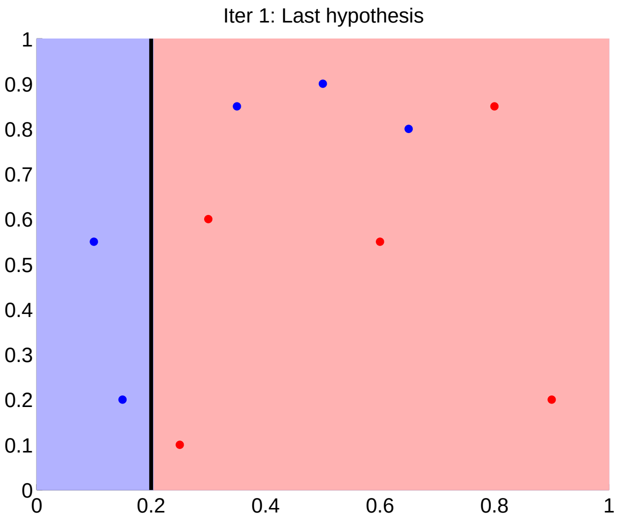


$\epsilon_3 = 0.13$

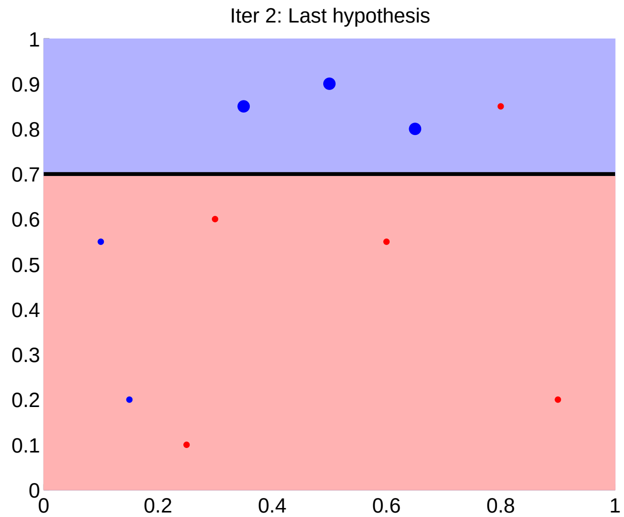
$\alpha_3 = 0.92$

AdaBoost.M1 graphically

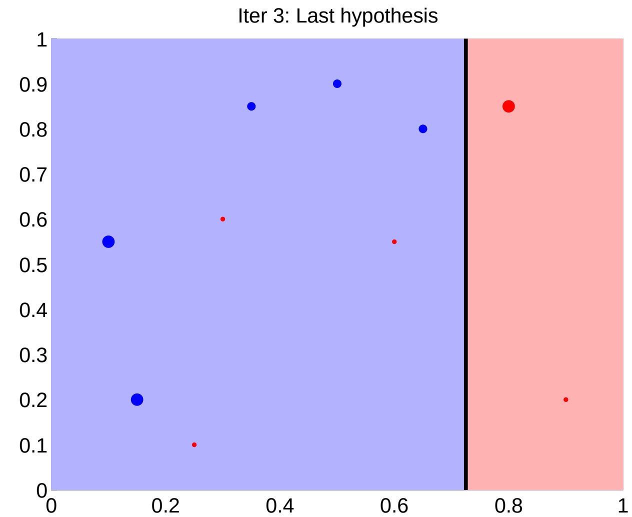
Iteration 1:



Iteration 2:



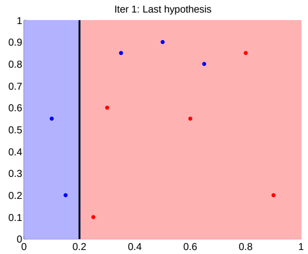
Iteration 3:



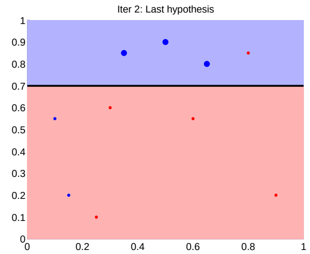
$$\epsilon_1 = 0.3 \qquad \epsilon_2 = 0.21 \qquad \epsilon_3 = 0.13$$

$$\alpha_1 = 0.42 \qquad \alpha_2 = 0.65 \qquad \alpha_3 = 0.92$$

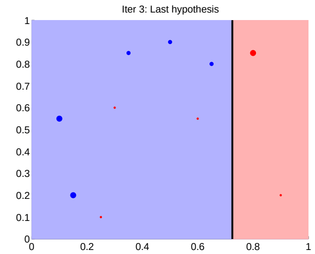
$0.42 \times$



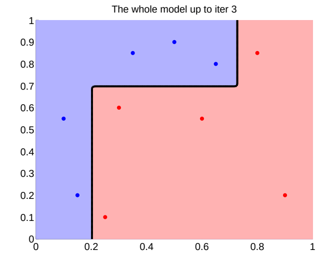
$+0.65 \times$



$+0.92 \times$



$=$



AdaBoost: remarks

The training error:

- Let $\gamma_t = 0.5 - \epsilon_t$ be the improvement of the t-th model over a random guess.
- Let $\gamma = \min_t \gamma_t$ be the minimal improvement, i.e. the difference of error of all models h_t compared to the error of random guessing is at least γ, i.e.

$$\forall t : \gamma_t \geq \gamma > 0.$$

- It can be shown that the training error

$$\mathrm{Err}_{\mathrm{Tr}}(H_{\mathrm{final}}) \leq e^{-2\gamma^2 M}$$

Forward stagewise additive modeling

Boosting tries to solve the following optimization problem: $f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{|T|} L(y_i, f(\boldsymbol{x}_i))$

Finding the optimal f^* is hard; we shall tackle it sequentially:

Algorithm 2: Forward stagewise additive modeling (FSAM)

```
begin
    Initialize f_0(x) = 0.
    for m = 1, ..., M do
        Compute (α_m, θ_m) = argmin_{α,θ} Σ_{i=1}^{|T|} L(y_i, f_{m-1}(x_i) + α h(x_i; θ)).
        Set f_m(x) = f_{m-1}(x) + α_m h(x; θ_m).
```

AdaBoost.M1 is equivalent to FSAM using the **exponential loss function** $L(y, f(\boldsymbol{x})) = \exp(-y \cdot f(\boldsymbol{x}))$.

$$
\begin{aligned}
(\alpha_m, \theta_m) &= \arg\min_{\alpha,\theta} \sum_{i=1}^{|T|} \exp\left[-y_i \left(f_{m-1}(\boldsymbol{x}_i) + \alpha h(\boldsymbol{x}_i; \theta)\right)\right] \\
&= \arg\min_{\alpha,\theta} \sum_{i=1}^{|T|} w_m(i) \exp\left[-y_i \alpha h(\boldsymbol{x}_i; \theta)\right],
\end{aligned}
$$

where $w_m(i) = \exp(-y_i f_{m-1}(\boldsymbol{x}_i))$ depend neither on α_m nor θ_m and can be regarded as weights of training examples, which change each iteration. AdaBoost.M1 then follows from minimization of the last expression.

L2Boosting

Suppose we need to solve regression problem with squared error loss (L2).

- Then at step m we have:

$$\begin{aligned}L(y_i, f_m(\boldsymbol{x}_i)) &= L(y_i, f_{m-1}(\boldsymbol{x}_i) + \alpha_m h(\boldsymbol{x}_i;\theta_m)) = \\ &= (y_i - f_{m-1}(\boldsymbol{x}_i) - \alpha_m h(\boldsymbol{x}_i;\theta_m))^2 = \\ &= (r_{im} - \alpha_m h(\boldsymbol{x}_i;\theta_m))^2,\end{aligned}$$

  where we define $r_{im} = y_i - f_{m-1}(\boldsymbol{x}_i)$ to be the current **residual** of the model for ith data point.
- By fitting each weak model h_m to the residuals r_{im}, the mth model f_m learns to correct its predecessor f_{m-1}.
- Observation: the residuals $r_{im} = y_i - f_{m-1}(\boldsymbol{x}_i)$ are negative gradients of the squared error loss function $\frac{1}{2}(y - f(\boldsymbol{x}))^2$.
- The algorithm can be viewed as a *gradient descent in the space of functions*.
- The generalization of
  - FSAM using exponential loss (AdaBoost.M1) and
  - FSAM using L2 loss (L2Boosting)

  for a general differentiable loss function L is called **Gradient Boosting Machine**.

Gradient Boosting Algorithms

Algorithm 3: Gradient boosting

Input: Training set of labeled examples: $T = \{\boldsymbol{x}_i, y_i\}, i = 1, \ldots, |T|$, a differentiable loss function $L(y, f(\boldsymbol{x}))$, number of iterations M.
Output: Final model $f_{\mathrm{M}}(\boldsymbol{x})$.

begin

Initialize model with constant value: $f_0(\boldsymbol{x}) = \arg\min_{\gamma} \sum_{i=1}^{|T|} L(y_i, \gamma)$

for $m = 1, \ldots, M$ **do**

Compute *pseudo-residuals* $r_{im} = -\left. \frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)} \right|_{f(x) = f_{m-1}(x)}$ for all $i = 1, \ldots, |T|$.

Fit model $h_m(\boldsymbol{x})$ to pseudo-residuals, i.e. use training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|T|}$.

Compute multiplier α_m by solving the following 1D opt. problem:

$$\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{|T|} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \alpha h_m(\boldsymbol{x}_i)).$$

Update the model: $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \alpha_m h_m(\boldsymbol{x})$

By plugging in different loss functions, we can construct different boosting variants like

- AdaBoost,
- L2Boost,
- LogitBoost,
- etc.

Further considerations

Choosing the number of models M:

- The optimal value usually found by tracking the error on validation set.
- Often, we do not bother; we just set it sufficiently high (several hundreds). Boosting can overfit, but is quite resistant to it.

Further considerations

Choosing the number of models M:

- The optimal value usually found by tracking the error on validation set.
- Often, we do not bother; we just set it sufficiently high (several hundreds). Boosting can overfit, but is quite resistant to it.

Shrinkage:

- Often, the so-called shrinkage is applied, i.e. only a small part of the mth model is used:

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu\alpha_m h(\boldsymbol{x}; \theta_m),$$

  where $\nu \in (0, 1)$, often $\nu \approx 0.1$, is the so-called *learning rate*.
- Learning is slowed down; it requires more models to be added to the model, providing a configuration trade-off between the number of trees and learning rate.

Further considerations

Choosing the number of models M:

- The optimal value usually found by tracking the error on validation set.
- Often, we do not bother; we just set it sufficiently high (several hundreds). Boosting can overfit, but is quite resistant to it.

Shrinkage:

- Often, the so-called shrinkage is applied, i.e. only a small part of the mth model is used:

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu\alpha_m h(\boldsymbol{x}; \theta_m),$$

  where $\nu \in (0, 1)$, often $\nu \approx 0.1$, is the so-called *learning rate*.
- Learning is slowed down; it requires more models to be added to the model, providing a configuration trade-off between the number of trees and learning rate.

Stochastic gradient boosting

- It is possible to subsample the training data set and use only a subset of it to train each model.
- Subsample examples as in boosting (but without replacement).
- Subsample features as in random forests.
- It further *prevents overfitting*, *speeds up learning* of individual models, and gives chance to *compute out-of-bag error* estimates.

Summary

Competencies

After this lecture, a student shall be able to . . .

- describe the basic principle behind all committee/ensemble methods;
- list and conceptually compare several methods to achieve diversity among models trained on the same data, and know which of these methods are used in which ensemble algorithms;
- explain the purpose and the basic principle of stacking;
- explain how a bootstrap sample is created from the available data, and describe its properties;
- describe features of bagging;
- explain how to compute out-of-bag error estimate when using bagging;
- explain the principle of random forests and describe their difference to bagging with trees;
- explain how to compute a score of variable importance using random forest;
- explain the hypothesis boosting problem, and define a weak and a strong classifier in this context;
- explain the basic principle of AdaBoost.M1 algorithm;
- relate the training error of the AdaBoost algorithm to the number of constituent models and to the errors of individual models;
- describe the relations of AdaBoost.M1, L2Boost, and Gradient Boosting.