CS 2750 Machine Learning
Lecture 23

Learning with multiple models
Mixture of experts
Bagging and Boosting

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Learning with multiple models

We know how to build different classification or regression models from data

- **Question:**
  - Is it possible to learn and combine multiple (classification/regression) models and improve their predictive performance ?
- **Answer: yes**
- There are different ways of how to do it…

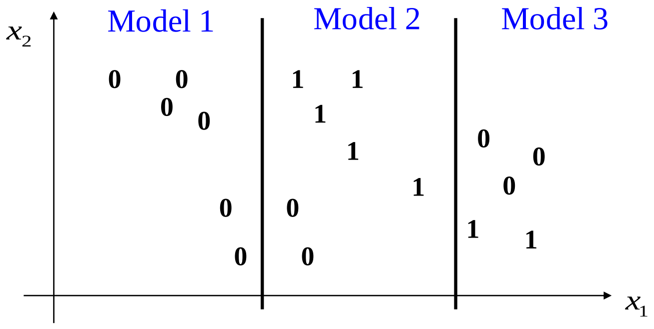
Learning with multiple models

- **Question:**
  - Is it possible to learn and combine multiple (classification/regression) models and improve their predictive performance ?
- There are different ways of how to do it…

- Assume you have models M1, M2, … Mk
- **Approach 1:** use different models (classifiers, regressors) to cover **the different parts of the input (x) space**
- **Approach 2:** use different models (classifiers, regressors) that cover **the complete input (x) space**, and combine their predictions

Approach 1

- Recall the decision tree:
  - **It partitions the input space to regions**
  - **picks the class independently**
- **What if we define a more general partitions of the input space and learn a model specific to these partitions**

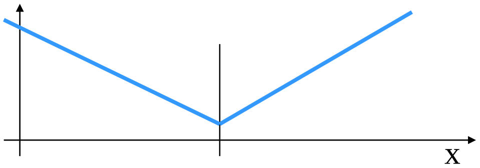


Learning with multiple models: Approach 1

Define a more general partitions of the input space and learn a model specific to these partitions

Example:

- 2 linear functions covering two regions of the input space



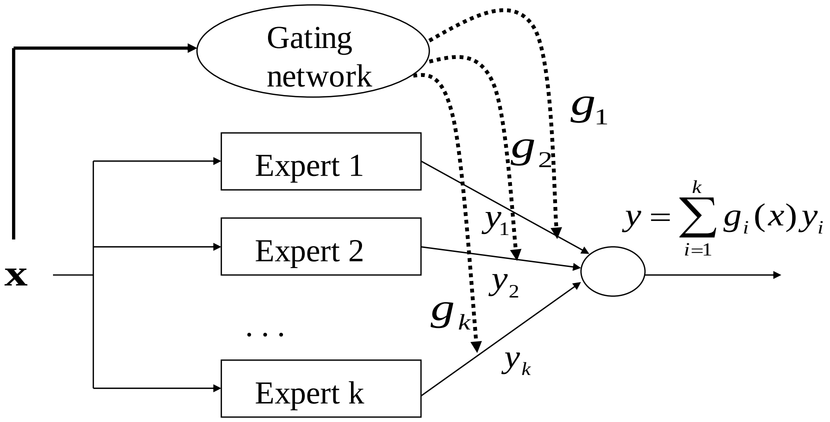
Mixture of expert model:

- Expert = learner (model)
- Different input regions covered with a different learner/model
- A "soft" switching between learners

Mixture of experts model

- **Gating network** : decides what expert to use

$g_1, g_2, \ldots g_k$ - gating functions



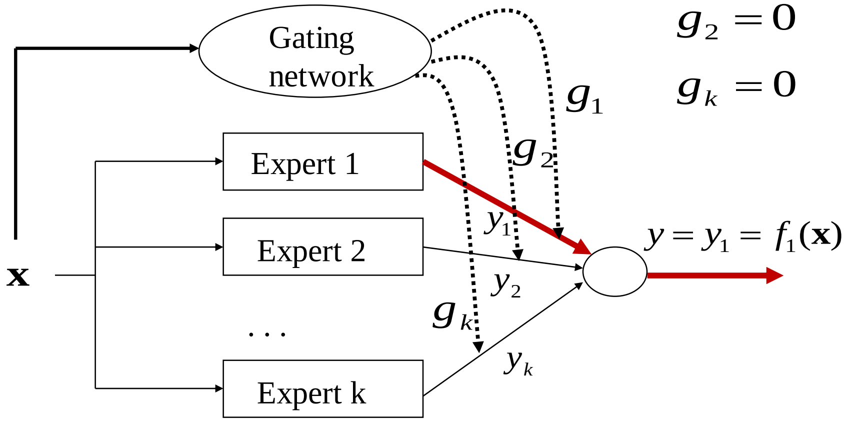
Mixture of experts model

- **Gating network** : decides what expert to use

$g_1, g_2, \ldots g_k$ - gating functions

Assume

$$g_1 = 1$$
$$g_2 = 0$$
$$g_k = 0$$



Learning mixture of experts

- **Learning consists of two tasks:**
  - Learn the parameters of individual expert networks
  - Learn the parameters of the gating (switching) network
    - Decides where to make a split
- **Assume:** gating functions give probabilities

$$0 \leq g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots g_k(\mathbf{x}) \leq 1 \qquad \sum_{u=1}^{k} g_u(\mathbf{x}) = 1$$

$$y = \sum_{u=1}^{k} g_u(\mathbf{x}) \, f_u(\mathbf{x})$$

- Based on the probability we partition the space
  - partitions belongs to different experts
- How to model the gating network?
  - **A multi-way classifier model:**
    - **softmax model**

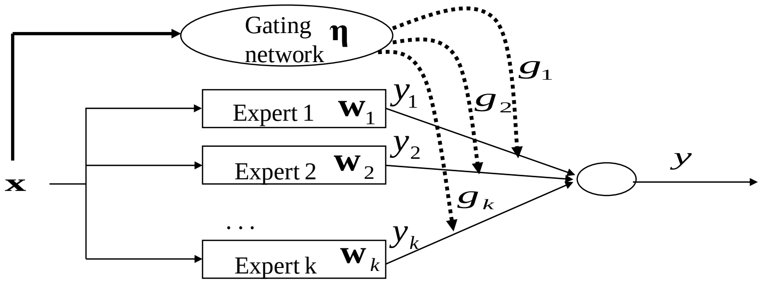
Learning mixture of experts

- Assume we have a **set of k linear experts**

  $y_i = \mathbf{w}_i{}^T \mathbf{x} + \varepsilon \qquad \varepsilon \sim N(0,\sigma)$ (Note: bias terms are hidden in x)

- Assume a **softmax gating network**

$$g_i(\mathbf{x}) = \frac{\exp(\boldsymbol{\eta}_i{}^T \mathbf{x})}{\sum_{u=1}^{k} \exp(\boldsymbol{\eta}_u{}^T \mathbf{x})} \approx p(\omega_i \mid \mathbf{x}, \boldsymbol{\eta})$$



Learning mixture of experts

- Assume we have a **set of linear experts**

  $y_i = \mathbf{w}_i{}^T \mathbf{x} + \varepsilon \qquad \varepsilon \sim N(0,\sigma)$ (Note: bias terms are hidden in x)

- Assume a **softmax gating network**

$$g_i(\mathbf{x}) = \frac{\exp(\boldsymbol{\eta}_i{}^T \mathbf{x})}{\sum_{u=1}^{k} \exp(\boldsymbol{\eta}_u{}^T \mathbf{x})} \approx p(\omega_i \mid \mathbf{x}, \boldsymbol{\eta})$$

- Likelihood of y (linear regression – assume errors for different experts are normally distributed with the same variance)

$$P(y \mid \mathbf{x}, \mathbf{W}, \boldsymbol{\eta}) = \sum_{i=1}^{k} P(\omega_i \mid \mathbf{x}, \boldsymbol{\eta})\, p(y \mid \mathbf{x}, \omega_i, \mathbf{W})$$

$$= \sum_{i=1}^{k} \left[\frac{\exp(\boldsymbol{\eta}_i{}^T \mathbf{x})}{\sum_{j=1}^{k} \exp(\boldsymbol{\eta}_j{}^T \mathbf{x})} \right] \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{\left\| y - \mathbf{w}_i{}^T \mathbf{x}_i \right\|^2}{2\sigma^2} \right) \right]$$

Learning mixture of experts

Learning of parameters of expert models:

On-line update rule for parameters $\mathbf{w}_i$ of expert *i*

– If we know the expert that is responsible for **x**

$$w_{ij} \leftarrow w_{ij} + \alpha_{ij}(y - \mathbf{w}_i{}^T\mathbf{x})x_j$$

– If we do not know the expert

$$\boxed{w_{ij} \leftarrow w_{ij} + \alpha_{ij} h_i (y - \mathbf{w}_i{}^T\mathbf{x})x_j}$$

h_i - responsibility of the *i*th expert = a kind of posterior

$$h_i(\mathbf{x}, y) = \frac{g_i(\mathbf{x})\, p(y \mid \mathbf{x}, \omega_i, \mathbf{W})}{\sum_{u=1}^{k} g_u(\mathbf{x})\, p(y \mid \mathbf{x}, \omega_u, \mathbf{W})} = \frac{g_i(\mathbf{x}) \exp\left(-1/2 \left\| y - \mathbf{w}_i{}^T\mathbf{x} \right\|^2\right)}{\sum_{u=1}^{k} g_u(\mathbf{x}) \exp\left(-1/2 \left\| y - \mathbf{w}_u{}^T\mathbf{x} \right\|^2\right)}$$

$g_i(\mathbf{x})$ - a prior exp(...) - a likelihood

Learning mixtures of experts

Learning of parameters of the gating/switching network:

- **On-line learning of gating network parameters $\boldsymbol{\eta}_i$**

$$\boxed{\eta_{ij} \leftarrow \eta_{ij} + \beta_{ij}(h_i(\mathbf{x}, y) - g_i(\mathbf{x}))x_j}$$

- The learning with conditional mixtures can be extended to learning of parameters of an **arbitrary expert network**
  – e.g. logistic regression, multilayer neural network

$$\theta_{ij} \leftarrow \theta_{ij} + \beta_{ij} \frac{\partial l}{\partial \theta_{ij}}$$

$$\frac{\partial l}{\partial \theta_{ij}} = \frac{\partial l}{\partial \mu_i} \frac{\partial \mu_i}{\partial \theta_{ij}} = h_i \frac{\partial \mu_i}{\partial \theta_{ij}}$$

Learning with multiple models: Approach 2

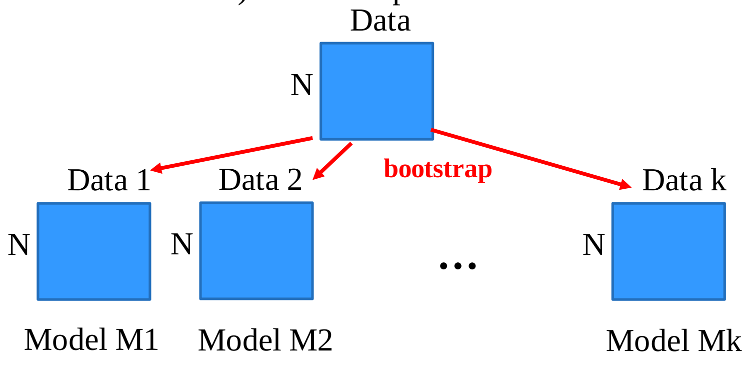
- **Approach 2:** use multiple models (classifiers, regressors) that cover the complete input (x) space and combine their outputs
- **Committee machines:**
  - Combine predictions of all models to produce the output
    - **Regression:** averaging
    - **Classification:** a majority vote
  - **Goal:** Improve the accuracy of the ‘base’ model
- **Methods:**
  - **Bagging (the same base models)**
  - **Boosting (the same base models)**
  - Stacking (different base model) not covered

Bagging (Bootstrap Aggregating)

- **Given:**
  - Training set of *N* examples
  - A base learning model (e.g. decision tree, neural network, …)
- **Method:**
  - Train multiple (k) base models on slightly different datasets
  - Predict (test) by averaging the results of k models
- **Goal:**
  - Improve the accuracy of one model by using its multiple copies
  - Average of misclassification errors on different data splits gives a better estimate of the predictive ability of a learning method

Bagging algorithm

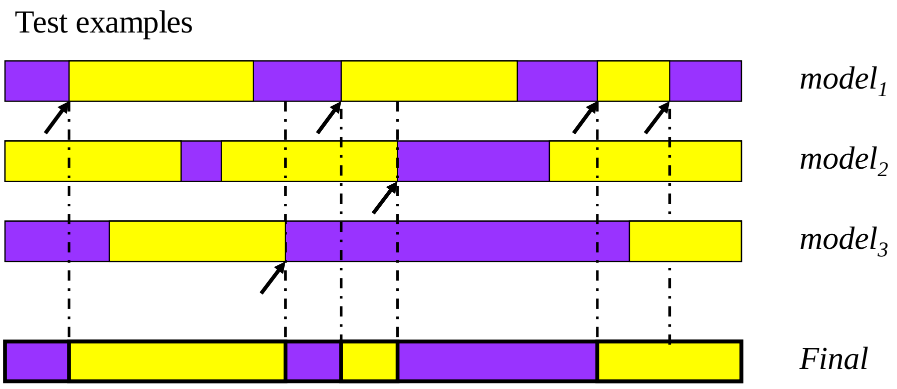
- **Training**
- For each model M1, M2, … Mk
  - Randomly sample with replacement *N* samples from the training set (bootstrap)
  - Train a chosen "base model" (e.g. neural network, decision tree) on the samples



Bagging algorithm

- **Training**
- For each model M1, M2, … Mk
  - Randomly sample with replacement *N* samples from the training set
  - Train a chosen "base model" (e.g. a neural network, or a decision tree) on the samples
- **Test**
  - For each test example
    - Run all base models M1, M2, … Mk
    - Predict by combining results of all T trained models:
      - **Regression:** averaging
      - **Classification:** a majority vote

Class decision via majority voting



🟨 *Class "yes"*

🟪 *Class "no"*

Analysis of Bagging

- **Expected error= Bias+Variance**
  - *Expected error* is the expected discrepancy between the estimated and true function

$$E\left[\left(\hat{f}(X)-E[f(X)]\right)^{2}\right]$$

It decomposes to two terms *Bias* + *Variance*

- *Bias* is a squared discrepancy between *averaged* estimated and true function

$$\left(E\left[\hat{f}(X)\right]-E[f(X)]\right)^{2}$$

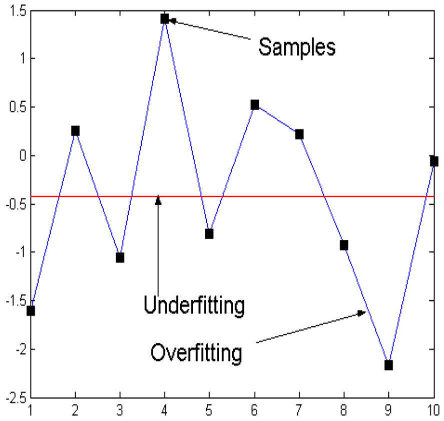
- *Variance* is an expected divergence of the estimated function vs. its average value

$$E\left[\left(\hat{f}(X)-E\left[\hat{f}(X)\right]\right)^{2}\right]$$

When Bagging works?

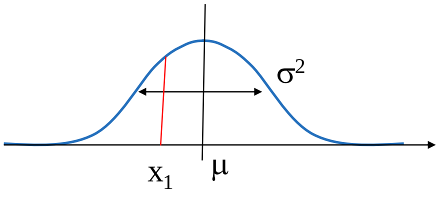
Under-fitting and over-fitting

- **Under-fitting:**
  - **High bias** (models are not accurate)
  - **Small variance** (smaller influence of examples in the training set)
- **Over-fitting:**
  - **Small bias** (models flexible enough to fit well to training data)
  - **Large variance** (models depend very much on the training set)



Averaging decreases variance

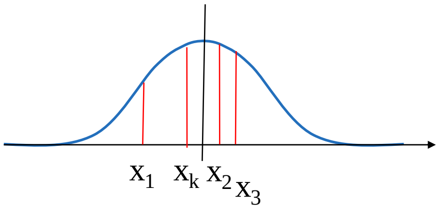
- **Example**
  - Assume a random variable x with a $N(\mu,\sigma^2)$ distribution



  - **Case 1:** we draw one example/measurement x_1 and use it to estimate the mean $\mu' = x_1$
    - The expected mean of the estimate $E[\mu']= E[x_1]= \mu$
    - The variance of the mean estimate $Var(\mu')= Var(x_1)=\sigma^2$

Averaging decreases variance

- **Example** Assume a random variable x with a $N(\mu,\sigma^2)$ distribution



  - **Case 2:** a variable *x* is measured independently *K* times $(x_1,x_2,\ldots x_k)$ and the mean is estimated as:

    $$\mu' = (x_1+x_2+\ldots+x_k)/K,$$

    - The expected mean of the estimate $E[\mu'] = \mu$
    - But, the variance of the mean estimate $Var(\mu')$is smaller:

    $$Var(\mu') = [Var(x_1)+\ldots Var(x_k)]/K^2 = K\sigma^2 / K^2 = \sigma^2/K$$

When Bagging works

Relation of the previous example to bagging:

- **Bagging is a kind of averaging!**

Main property of Bagging (proof omitted)

- Bagging **decreases variance** of the base model without changing the bias!!!
- Why? averaging!

Bagging typically helps

- When applied with an **over-fitted base model**
  - High dependency on actual training data
  - Example: fully grown decision trees

Bagging does not help much when

- Applied to models with a high bias. When the base model is robust to the changes in the training data (due to sampling)

Boosting

- **Bagging**
  - Multiple models covering the complete space, a learner is not biased to any region
  - Learners **are learned independently**
- **Boosting**
  - Every learner covers the complete space
  - Learners are biased to regions not predicted well by other learners
  - **Learners are dependent**

Boosting. Theoretical foundations.

- **PAC: Probably Approximately Correct framework**
  - **(ε,δ) solution**
- **PAC learning:**
  - Learning with **a pre-specified error** ε and **a confidence parameter δ**
  - the probability that the misclassification error (ME) is larger than ε is smaller than δ

$$P(ME(c) > \varepsilon) \le \delta$$

Alternative rewrite:

$$P(Acc(c) > 1-\varepsilon) > (1-\delta)$$

- **Accuracy (1-ε):** Percent of correctly classified samples in test
- **Confidence (1-δ):** The probability that in one experiment some target accuracy will be achieved

PAC Learnability

Strong (PAC) learnability:

- There exists a learning algorithm that **efficiently** learns the classification with a pre-specified **error and confidence values**

Strong (PAC) learner: A learning algorithm *P* that

- Given an arbitrary:
  - classification error ε (< 1/2), and
  - confidence δ (<1/2)

    or in other words:
    - classification accuracy > $(1-\varepsilon)$
    - confidence probability > $(1-\delta)$
- Outputs a classifier that satisfies this parameters
- **Efficiency: runs in time polynomial in $1/\delta$, $1/\varepsilon$**
  - Implies: number of samples *N* is polynomial in $1/\delta$, $1/\varepsilon$

Weak Learner

Weak learner:

- A learning algorithm (learner) *M* that gives **some fixed (not arbitrary !!!!)**:
  - error ε_o (<1/2) and
  - confidence δ_o (<1/2)
- Alternatively:
  - a classification accuracy > 0.5
  - with probability > 0.5

  and this on an arbitrary distribution of data entries

Weak learnability=Strong (PAC) learnability

- Assume there exists a **weak learner**
  - it is better that a random guess (> 50 %) with confidence higher than 50 % on any data distribution
- **Question:**
  - Is the problem also strongly PAC-learnable?
  - Can we generate an algorithm *P* that achieves an arbitrary (ε,δ) accuracy?
- **Why is this important?**
  - Usual classification methods (decision trees, neural nets), have good, but uncontrollable performances.
  - Can we improve their performance to achieve any pre-specified accuracy (confidence)?

Weak=Strong learnability!!!

- **Proof due to R. Schapire**

  An arbitrary (ε,δ) improvement is possible

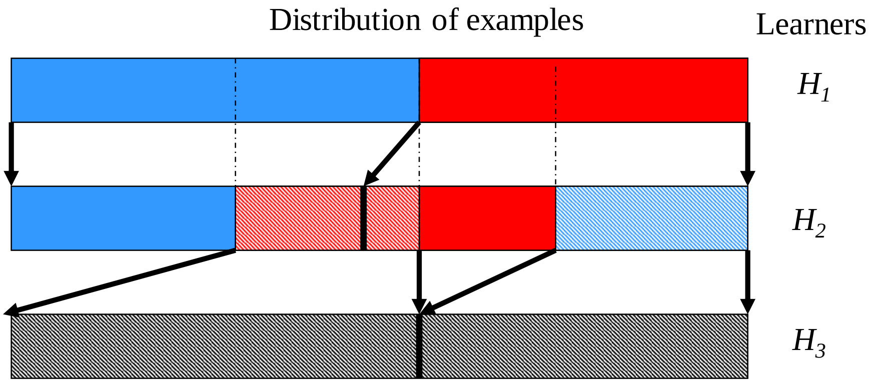
Idea: combine multiple weak learners together

- Weak learner *W* with confidence δ_o and maximal error ε_o
- It is possible:
  - To improve (boost) the confidence
  - To improve (boost) the accuracy

by training different weak learners on slightly different datasets

Boosting accuracy

Training



🟦 Correct classification

🟥 Wrong classification

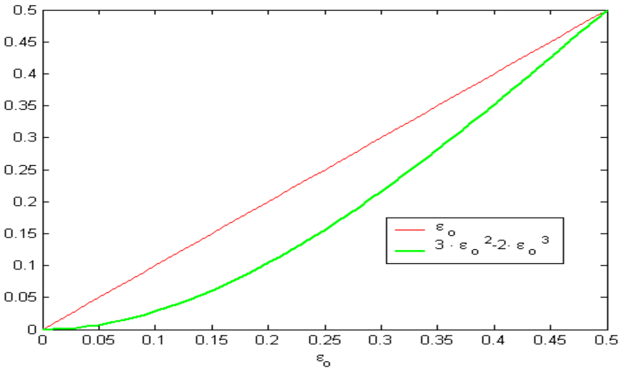
▨ H_1 and H_2 classify differently

Boosting accuracy

- **Training**
  - Sample randomly from the distribution of examples
  - Train hypothesis $H_{1.}$ on the sample
  - Evaluate accuracy of H_1 on the distribution
  - Sample randomly such that for the half of samples $H_{1.}$ provides correct, and for another half, incorrect results; Train hypothesis H_2.
  - Train H_3 on samples from the distribution where H_1 and H_2 classify differently
- **Test**
  - For each example, decide according to the majority vote of H_1, H_2 and H_3

Theorem

- If each classifier has an error < ε_o, the final 'voting' classifier has error < $g(\varepsilon_o) = 3\varepsilon_o^2 - 2\varepsilon_o^3$
- **Accuracy improved !!!!**
- **Apply recursively to get to the target accuracy !!!**



Theoretical Boosting algorithm

- Similarly to boosting the accuracy we can boost the confidence at some restricted accuracy cost
- **The key result:** we can improve both the accuracy and confidence

- **Problems with the theoretical algorithm**
  - A good (better than 50 %) classifier on all distributions and problems
  - We cannot get a good sample from data-distribution
  - The method requires a large training set
- **Solution to the sampling problem:**
  - Boosting by sampling
    - **AdaBoost** algorithm and variants

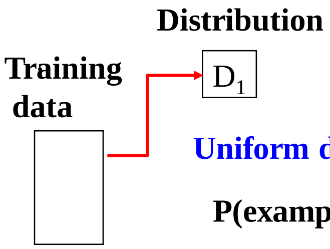
AdaBoost

- **AdaBoost: boosting by sampling**

- **Classification** (Freund, Schapire; 1996)
  - AdaBoost.M1 (two-class problem)
  - AdaBoost.M2 (multiple-class problem)

- **Regression** (Drucker; 1997)
  - AdaBoostR

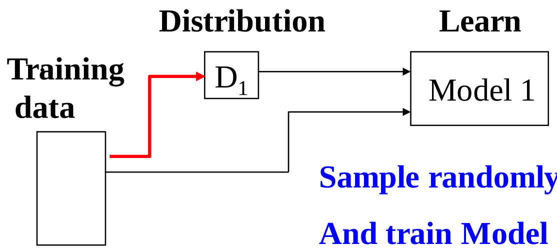
AdaBoost training



Uniform distribution D_1 of training examples

P(example i) = 1/N

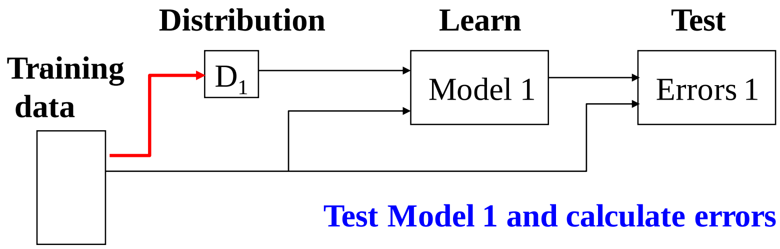
AdaBoost training



Sample randomly according to D_1

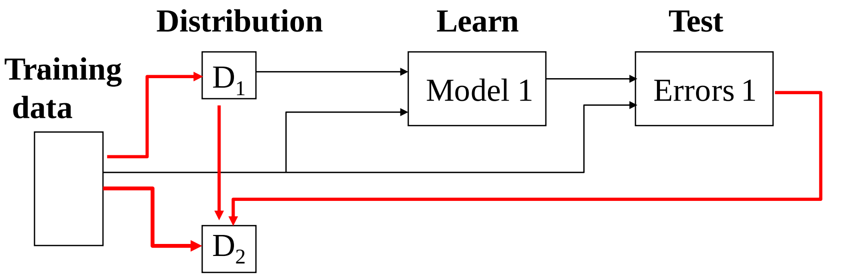
And train Model 1

AdaBoost training



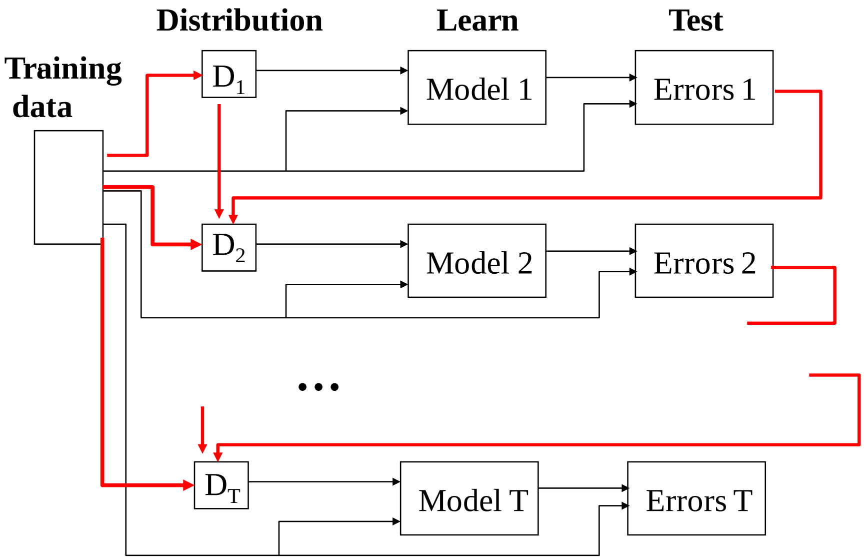
Test Model 1 and calculate errors

AdaBoost training



Use errors to recalculate the new distribution on data
Give more probability to pick examples with errors

AdaBoost training



AdaBoost

- **Given:**
  - A training set of *N* examples (attributes + class label pairs)
  - A “base” learning model (e.g. a decision tree, a neural network)
- **Training stage**:
  - Train a sequence of *T* “base” models on *T* different sampling distributions defined upon the training set (*D*)
  - A sample distribution D_t for building the model *t* is constructed by modifying the sampling distribution D_{t-1} from the *(t-1)*th step.
    - Examples classified incorrectly in the previous step receive higher weights in the new data (attempts to cover misclassified samples)
- **Application (classification) stage:**
  - **Classify according to** the **weighted majority** of classifiers

AdaBoost algorithm

Training (step t)

- **Sampling Distribution** D_t

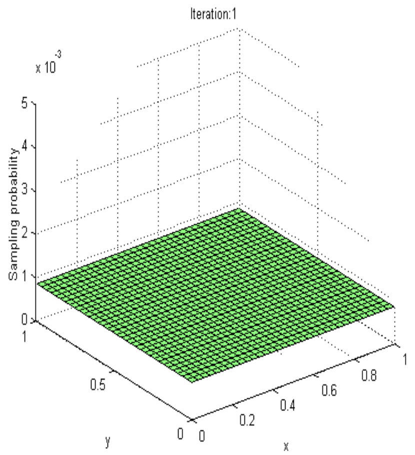
  $D_t(i)$ - a probability that example i from the original training dataset is selected

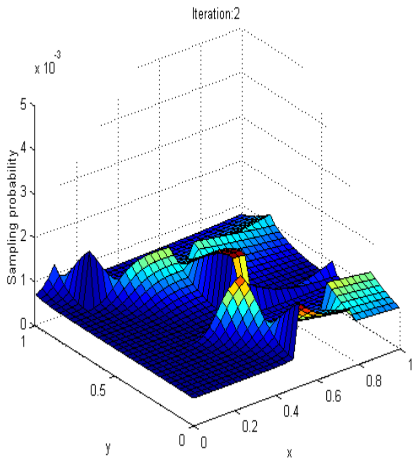
  $D_1(i) = 1/N$ for the first step (t=1)
- Take *K* samples from the training set according to D_t
- Train a classifier h_t on the samples
- Calculate the error ε_t of h_t: $\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
- **Classifier weight:** $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
- **New sampling distribution**

$$D_{t+1}(i) = \frac{D_t(i)}{\underbrace{Z_t}_{\text{Norm. constant}}} \times \begin{cases} \beta_t & h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

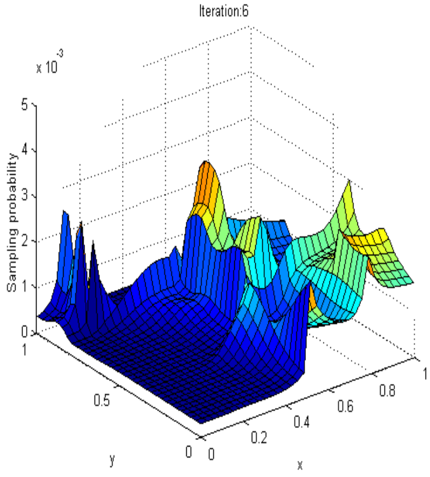
AdaBoost. Sampling Probabilities

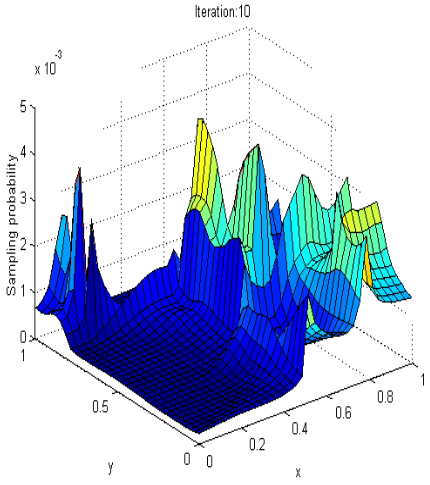
Example: - Nonlinearly separable binary classification
- NN used as a week learner





AdaBoost: Sampling Probabilities





AdaBoost classification

- We have T different classifiers h_t
  - weight w_t of the classifier is proportional to its accuracy on the training set

$$w_t = \log(1/\beta_t) = \log\left((1-\varepsilon_t)/\varepsilon_t\right)$$

$$\beta_t = \varepsilon_t / (1-\varepsilon_t)$$

- **Classification:**

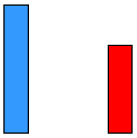
  For every class j=0,1
  - Compute the sum of weights w corresponding to ALL classifiers that predict class j;
  - Output class that correspond to the maximal sum of weights (weighted majority)

$$h_{final}(\mathbf{x}) = \arg\max_j \sum_{t:h_t(x)=j} w_t$$

Two-Class example. Classification.

• Classifier 1	"yes"	0.7	
• Classifier 2	"no"		0.3
• Classifier 3	"no"		0.2

- Weighted majority "yes"



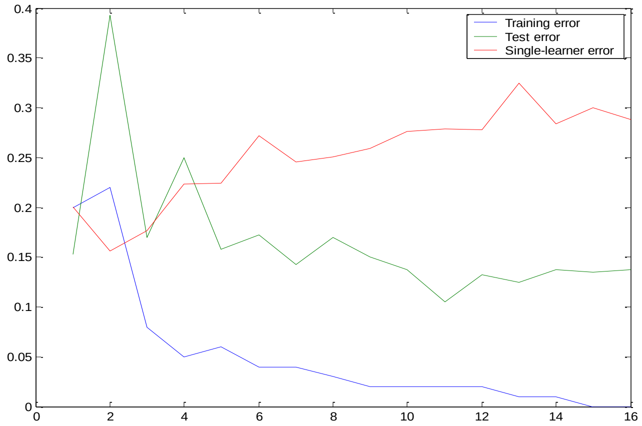
0.7 - 0.5 = + 0.2

- The final choice is "yes" + 1

What is boosting doing?

- Each classifier specializes on a particular subset of examples
- Algorithm is concentrating on “more and more difficult” examples
- **Boosting can:**
  - Reduce variance (the same as Bagging)
  - Eliminate the effect of high bias of the weak learner (unlike Bagging)
- **Train versus test errors performance**:
  - Train errors can be driven close to 0
  - But test errors do not show overfitting
- Proofs and theoretical explanations in **a number of papers**

Boosting. Error performances


Training error
Test error
Single-learner error