CS 3750 Machine Learning
Lecture 26

Ensemble methods.
Bagging and Boosting

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

Administrative announcements

- **Term projects:**
  - **Reports due on Friday, December 5 at 2pm.**
  - **15 (12+3) minutes presentations at the same time**

Ensemble methods

- **Mixture of experts**
  - Different 'base' models (classifiers, regressors) cover different parts of the input space
- **Committee machines:**
  - Train several 'base' models on the complete input space, but on slightly different train sets
  - Combine their decision to produce the final result
  - **Goal:** Improve the accuracy of the 'base' model
  - **Methods:**
    - **Bagging**
    - **Boosting**
    - Stacking (not covered)

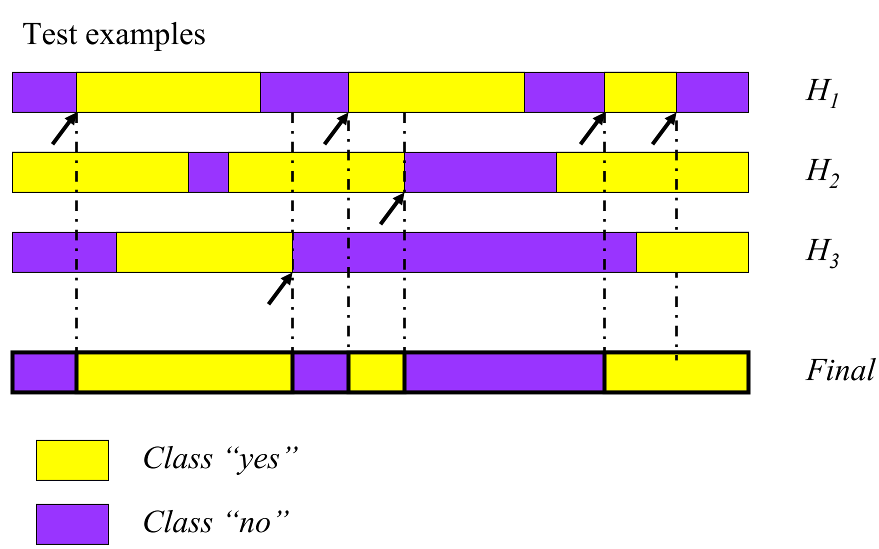
Bagging (Bootstrap Aggregating)

- **Given:**
  - Training set of N examples
  - A class of learning models (e.g. decision trees, neural networks, …)
- **Goal:**
  - Improve the accuracy of one model by using its multiple copies
- **Motivation:**
  - **Recall:** Average of misclassification errors on different data splits gives a better estimate of the predictive ability of a learning method
  - Train multiple models on different samples and average their predictions

Bagging algorithm

- **Training**
  - In each iteration t, t=1,…T
    - Randomly sample with replacement N samples from the training set
    - Train a chosen "base model" (e.g. neural network, decision tree) on the samples
- **Test**
  - For each test example
    - Start all trained base models
    - Predict by combining results of all T trained models:
      - **Regression:** averaging
      - **Classification:** a majority vote

Simple Majority Voting



When Bagging Works

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

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

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

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

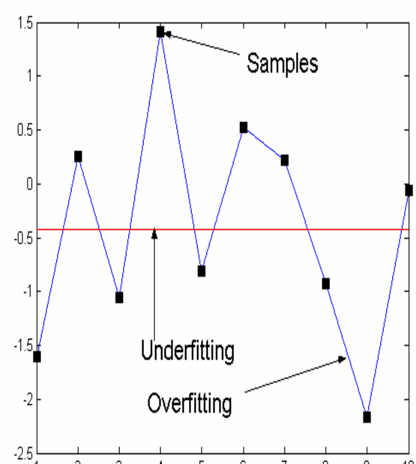
  - *Variance* is expected divergence of the estimated function vs. its average value

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

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 we measure a random variable x with a $N(\mu,\sigma^2)$ distribution
  - If only one measurement x_1 is done,
    - The expected mean of the measurement is μ
    - Variance is $Var(x_1)=\sigma^2$
  - If random variable x is measured K times $(x_1,x_2,\ldots x_k)$ and the value is estimated as: $(x_1+x_2+\ldots+x_k)/K$,
    - Mean of the estimate is still μ
    - But, variance is smaller:
      - $[Var(x_1)+\ldots Var(x_k)]/K^2=K\sigma^2 / K^2 = \sigma^2/K$
- Observe: **Bagging is a kind of averaging!**

When Bagging works

- **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
- **It does not help much**
  - High bias. When the base model is robust to the changes in the training data (due to sampling)

Boosting. Theoretical foundations.

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

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

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

PAC Learnability

Strong (PAC) learnability:

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

Strong (PAC) learner:

- A learning algorithm P that given an arbitrary
  - classification error ε (<1/2), and
  - confidence δ (<1/2)
- Outputs a classifier
  - With a classification accuracy > (1-ε)
  - A confidence probability > (1- δ)
  - And runs in time polynomial in 1/ δ, 1/ε
    - Implies: number of samples N is polynomial in 1/ δ, 1/ε

Weak Learner

Weak learner:

- A learning algorithm (learner) *W*
  - Providing classification accuracy $>1\text{-}\varepsilon_o$
  - With probability $>1\text{-}\ \delta_o$
- For some **fixed** and **uncontrollable**
  - classification error ε_o (<1/2)
  - confidence δ_o (<1/2)

  on an arbitrary problem

Weak learnability=Strong (PAC) learnability

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

Weak=Strong learnability!!!

- **Proof due to R. Schapire**
  An arbitrary (ε-δ) improvement is possible

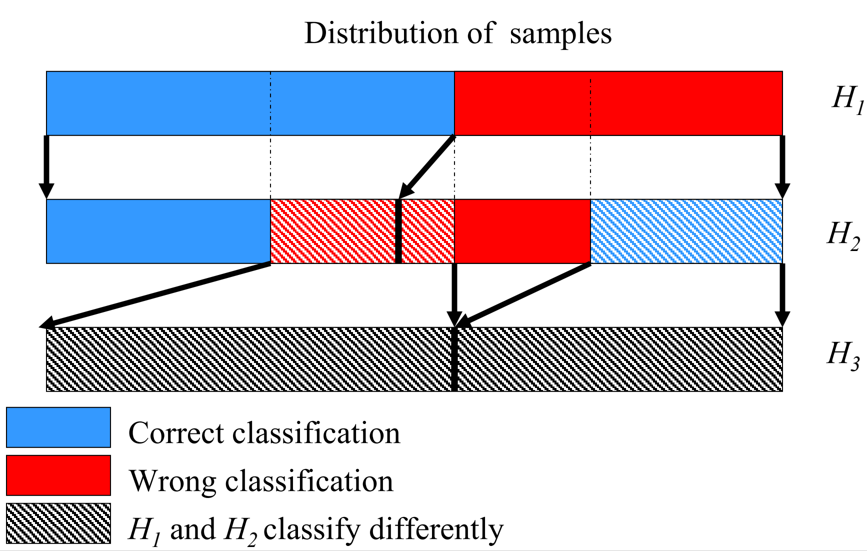
Idea: combine multiple weak learners together

- 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

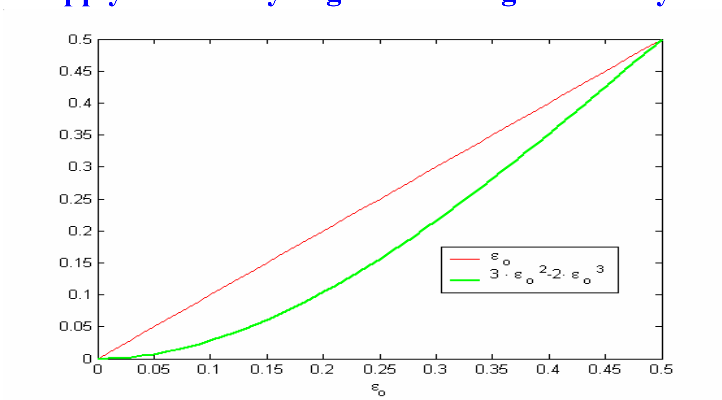


Boosting accuracy

- **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 hypothesis has an error ε_o, the final 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 problems
  - We cannot properly sample from data-distribution
  - The method requires large training set

- **Solution to the sampling problem:**
  - Boosting by sampling
    - **AdaBoost** algorithm and variants

AdaBoost

- **AdaBoost: boosting by sampling**

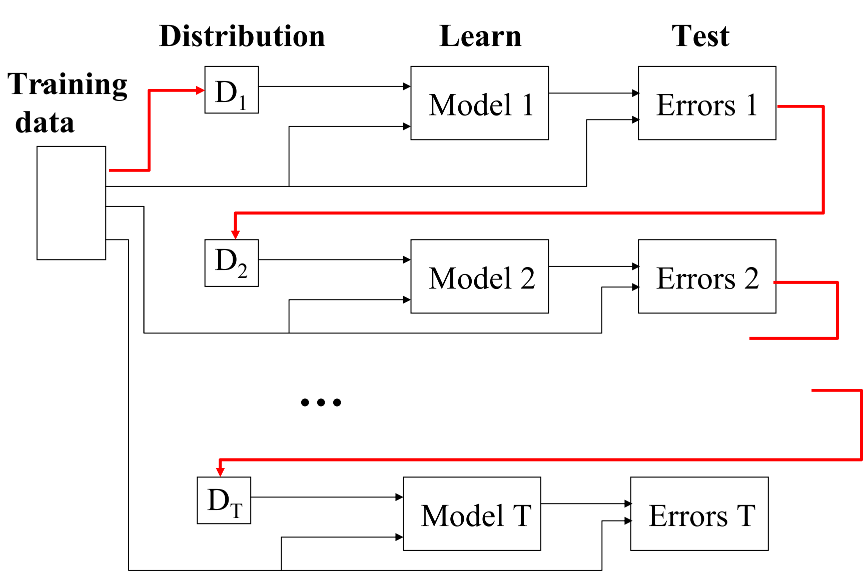
- **Classification** (Freund, Schapire; 1996)
  - AdaBoost.M1 (two-class problem)
  - AdaBoost.M2 (multiple-class problem)

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

AdaBoost

- **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 training



AdaBoost algorithm

Training (step t)

- **Sampling Distribution** D_t

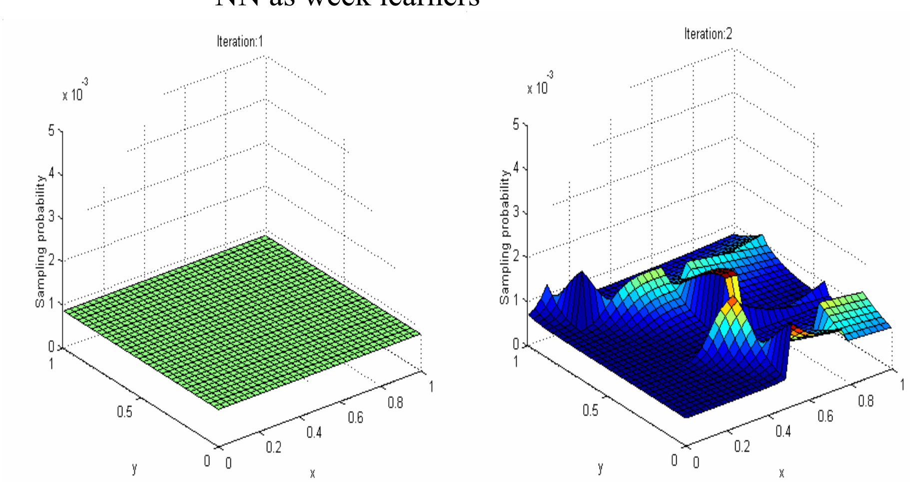
  $D_t(i)$ - a probability that example i from the original training dataset is selected

  $D_1(i) = 1/N$ for the first step (t=1)
- Take K samples from the training set D 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

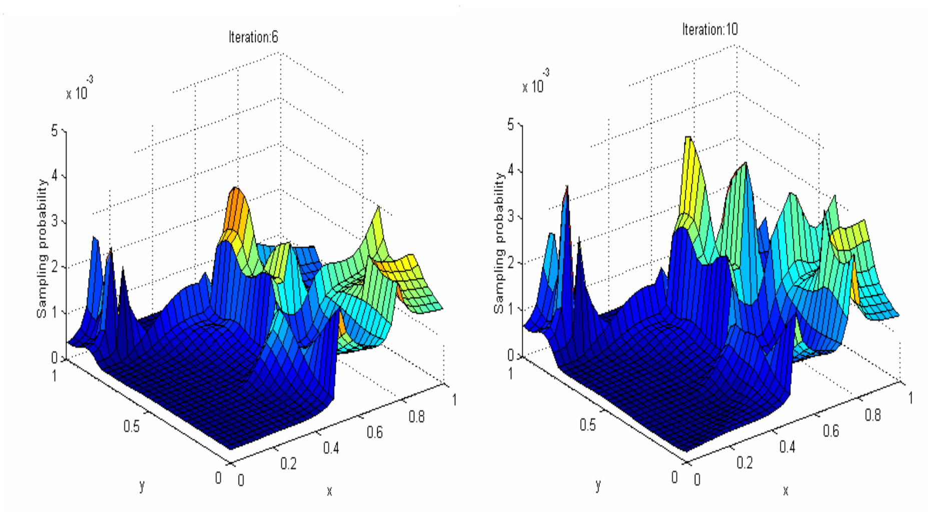
$$D_{t+1}(i) = \frac{D_t(i)}{\underbrace{Z_t}_{\textbf{Norm. constant}}} \times \begin{cases} \beta_t & h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

AdaBoost. Sampling Probabilities

Example: - Nonlinearly separable binary classification
- NN as week learners



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:**

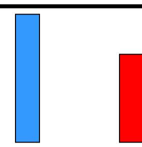
  For every class j=0,1
  - Compute the sum of weights w corresponding to ALL classifiers that predict class j;
  - Output class that correspond to the maximal sum of weights (weighted majority)

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

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 choose 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)
  - But also to 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 **readings**

Boosting. Error performances