CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 5

Boosting

Some slides are due to Robin Dhamankar
Vandi Verma & Sebastian Thrun

Today

- New Machine Learning Topics:
  - Ensemble Learning
    - Bagging
    - Boosting
- Next time **two** papers:
  - "Rapid Object Detection using a Boosted Cascade of Simple Features" by P. Viola and M. Jones from CVPR2001
  - "Detecting Pedestrians Using Patterns of Motion and Appearance" by P. Viola, M.J.Jones, D. Snow

Ensemble Learning: Bagging and Boosting

- So far we have talked about design of a single classifier that generalizes well (want to "learn" f(x))
- From statistics, we know that it is good to average your predictions (reduces variance)
- Bagging
  - reshuffle your training data to create k different training sets and learn $f_1(x), f_2(x), \ldots, f_k(x)$
  - Combine the k different classifiers by majority voting

$$f_{FINAL}(x) = \text{sign}[\Sigma\ 1/k\ f_i(x)\ ]$$

- Boosting
  - Assign different weights to training samples in a "smart" way so that different classifiers pay more attention to different samples
  - Weighted majority voting, the weight of individual classifier is proportional to its accuracy
  - Ada-boost (1996) was influenced by bagging, and it is superior to bagging

Bagging

- Generate a random sample from training set by selecting ***l*** elements (out of ***n*** elements available) with replacement
- each classifier is trained on the average of 63.2% of the training examples
  - For a dataset with N examples, each example has a probability of $1-(1-1/N)^N$ of being selected at least once in the N samples. For $N \rightarrow \infty$, this number converges to (1-1/e) or 0.632 [Bauer and Kohavi, 1999]
- Repeat the sampling procedure, getting a sequence of ***k*** independent training sets
- A corresponding sequence of classifiers $f_1(x), f_2(x), \ldots, f_k(x)$ is constructed for each of these training sets, using the same classification algorithm
- To classify an unknown sample x, let each classifier predict.
- The *bagged classifier* $f_{FINAL}(x)$ then combines the predictions of the individual classifiers to generate the final outcome, frequently this combination is simple voting

Boosting: motivation

- It is usually hard to design an accurate classifier which generalizes well
- However it is usually easy to find many "rule of thumb" *weak* classifiers
  - A classifier is weak if it is only slightly better than random guessing
- Can we combine several weak classifiers to produce an accurate classifier?
  - Question people have been working on since 1980's

Ada Boost

- Let's assume we have 2-class classification problem, with $y_i \in \{-1,1\}$
- Ada boost will produce a discriminant function:

$$g(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t \boldsymbol{f}_t(\boldsymbol{x})$$

- where $f_t(x)$ is the "weak" classifier
- As usual, the final classifier is the sign of the discriminant function, that is $f_{final}(x) = \text{sign}[g(x)]$

Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially distribution of weights is uniform
- At successive iterations, the weight of misclassified examples is increased, forcing the weak learner to focus on the hard examples in the training set

More Comments on Ada Boost

- Ada boost is very simple to implement, provided you have an implementation of a "weak learner"
- Will work as long as the "basic" classifier $f_t(x)$ is at least slightly better than random
  - will work if the error rate of $f_t(x)$ is less than 0.5 (0.5 is the error rate of a random guessing classifier for a 2-class problem)
- Can be applied to boost any classifier, not necessarily weak

Ada Boost (slightly modified from the original version)

- $d(x)$ is the distribution of weights over the N training points $\sum d(x_i)=1$
- Initially assign uniform weights $d_0(x_i) = 1/N$ for all x_i
- At each iteration t :
  - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
  - Compute the error rate ε_t as
    $\varepsilon_t = \sum_{i=1\ldots N} d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$
  - assign weight α_t the classifier f_t's in the final hypothesis
    $\alpha_t = \log((1 - \varepsilon_t)/\varepsilon_t)$
  - For each x_i, $d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot I(y_i \neq f_t(x_i))]$
  - Normalize $d_{t+1}(x_i)$ so that $\sum_{i=1} d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}[\sum \alpha_t f_t(x)]$

Ada Boost

- At each iteration t :
  - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
  - Compute ε_t the error rate as
    $\varepsilon_t = \sum d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$
  - assign weight α_t the classifier f_t's in the final hypothesis
    $\alpha_t = \log((1 - \varepsilon_t)/\varepsilon_t)$
  - For each x_i, $d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot I(y_i \neq f_t(x_i))]$
  - Normalize $d_{t+1}(x_i)$ so that $\sum_{t+1} d(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}[\sum \alpha_t f_t(x)]$

- If the classifier does not take weighted samples, this step can be achieved by sampling from the training samples according to the distribution $d_t(x)$

Ada Boost

- At each iteration t :
  - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
  - Compute ε_t the error rate as

    $\varepsilon_t = \sum d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$
  - assign weight α_t the classifier f_t's in the final hypothesis

    $\alpha_t = \log((1 - \varepsilon_t)/\varepsilon_t)$
  - For each x_i, $d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot I(y_i \neq f_t(x_i))]$
  - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}[\sum \alpha_t f_t(x)]$

- Since the weak classifier is better than random, we expect $\varepsilon_t < 1/2$

Ada Boost

- At each iteration t :
  - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
  - Compute ε_t the error rate as

    $\varepsilon_t = \sum d(x_i) \cdot I(y_i \neq f_t(x_i))$
  - assign weight α_t the classifier f_t's in the final hypothesis

    $\alpha_t = \log((1 - \varepsilon_t)/\varepsilon_t)$
  - For each x_i, $d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot I(y_i \neq f_t(x_i))]$
  - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}[\sum \alpha_t f_t(x)]$

- Recall that $\varepsilon_t < ½$
- Thus $(1 - \varepsilon_t)/\varepsilon_t > 1 \Rightarrow \alpha_t > 0$
- The smaller is ε_t, the larger is α_t, and thus the more importance (weight) classifier $f_t(x)$ gets in the final classifier

  $f_{FINAL}(x) = \text{sign}[\sum \alpha_t f_t(x)]$

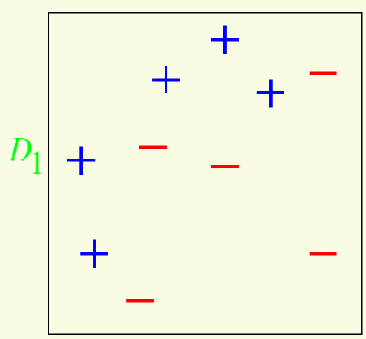
Ada Boost

- At each iteration t :
  - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
  - Compute ε_t the error rate as
    $\varepsilon_t = \sum d_t(x_i) \cdot I(y_i \neq f_t(x_i))$
  - assign weight α_t the classifier f_t's in the final hypothesis
    $\alpha_t = \log((1 - \varepsilon_t)/\varepsilon_t)$
  - For each x_i, $d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot I(y_i \neq f_t(x_i))]$
  - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}\,[\sum \alpha_t f_t(x)]$

- Weight of misclassified examples is increased and the new $d_{t+1}(x_i)$'s are normalized to be a distribution again

AdaBoost Example

from "**A Tutorial on Boosting**" **by** Yoav Freund and Rob Schapire

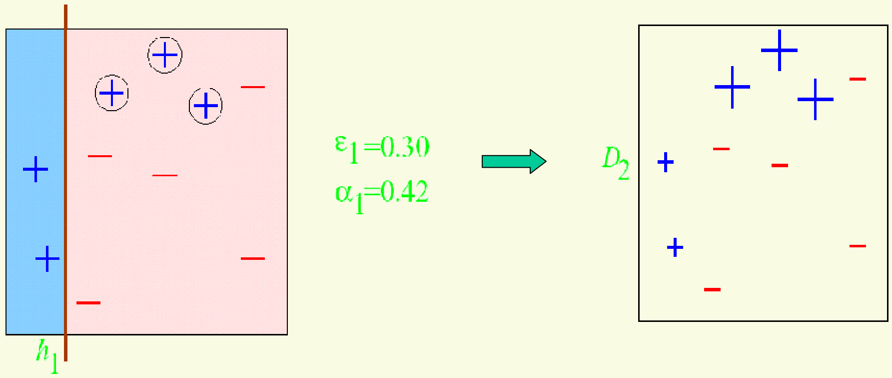


Original Training set : equal weights to all training samples

Note: in the following slides, $h_t(x)$ is used instead of $f_t(x)$, and D instead of d

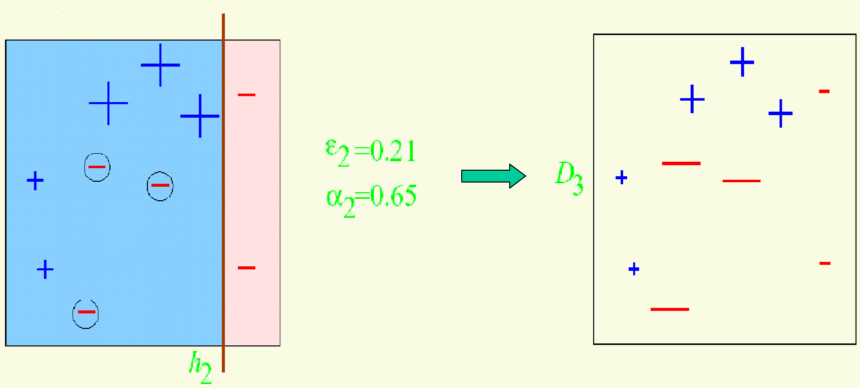
AdaBoost Example

ROUND 1



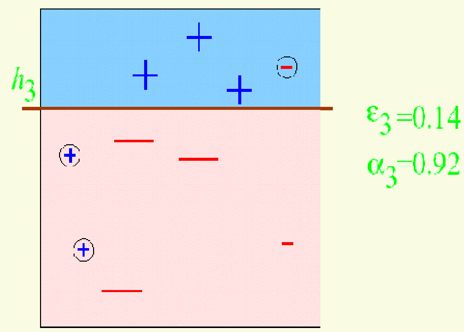
AdaBoost Example

ROUND 2

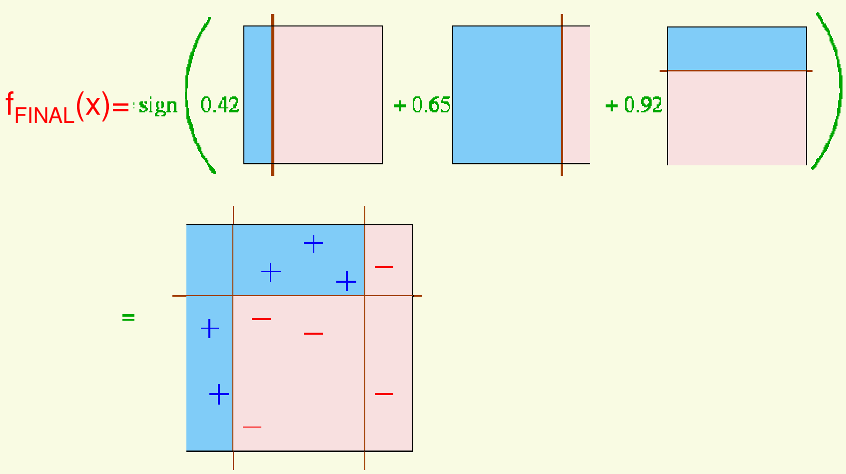


AdaBoost Example

ROUND 3



AdaBoost Example



AdaBoost Comments

- It can be shown that the training error drops exponentially fast, if each weak classifier is slightly better than random

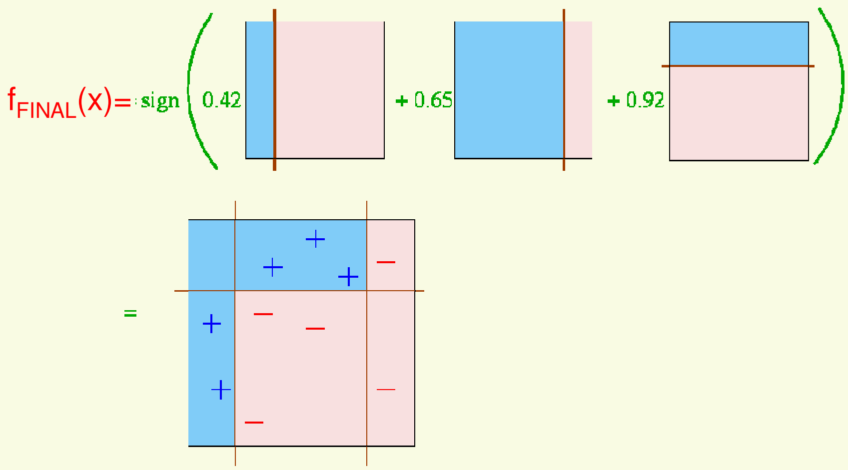
$$Err_{train} \le \exp\left(-2\sum_t \gamma_t^2\right)$$

- Here $\gamma_t = \varepsilon_t - 1/2$, where is classification error at round t (weak classifier f_t)

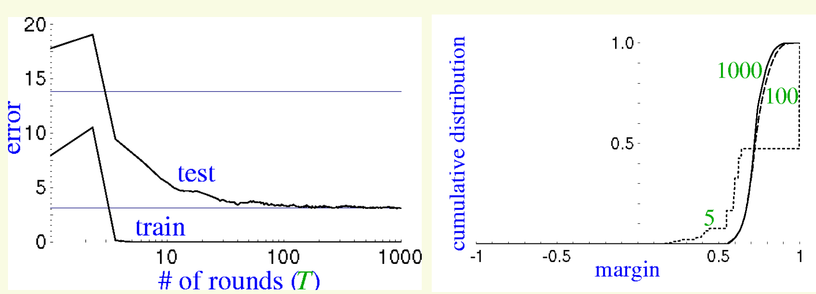
AdaBoost Comments

- But we are really interested in the generalization properties of $f_{FINAL}(x)$, not the training error
- AdaBoost was shown to have excellent generalization properties in practice
  - the more rounds, the more complex is the final classifier, so overfitting is expected as the training proceeds
  - but in the beginning researchers observed no overfitting of the data
  - It turns out it does overfit data eventually, if you run it really long
- It can be shown that boosting "aggressively" increases the margins of training examples, as iterations proceed
  - margins continue to increase even when training error reaches zero
  - Helps to explain empirically observed phenomena: test error continues to drop even after training error reaches zero

AdaBoost Example



The Margin Distribution



epoch	5	100	1000
training error	0.0	0.0	0.0
test error	8.4	3.3	3.1
%margins≤0.5	7.7	0.0	0.0
Minimum margin	0.14	0.52	0.55

Boosting As Additive Model

- The final prediction in boosting *g(x)* can be expressed as an additive expansion of individual classifiers

$$g(x) = \sum_{k=1}^{M} \alpha_k f_k(x;\gamma_k)$$

- Typically we would try to minimize a loss function on the N training examples

$$\min_{\alpha_1,\gamma_1,\ldots,\gamma_M,\alpha_M} \sum_{i=1}^{N} L\left(y_i, \sum_{k=1}^{M} \alpha_k f_k(x_i;\gamma_k)\right)$$

- For example, under squared-error loss:

$$\min_{\alpha_1,\gamma_1,\ldots,\gamma_M,\alpha_M} \sum_{i=1}^{N} \left(y_i - \sum_{k=1}^{M} \alpha_k f_k(x_i;\gamma_k)\right)^2$$

Boosting As Additive Model

- Forward stage-wise modeling is iterative and fits the $f_k(x,\gamma_k)$ sequentially, fixing the results of previous iterations

$$\underbrace{g_t(x)}_{\text{model at iteration } t} = \underbrace{g_{t-1}(x)}_{\text{fixed}} + \underbrace{\alpha_t f_t(x;\gamma_t)}_{\text{fit } \gamma_t,\ \alpha_t \text{ to produce improved } g_t(x)}$$

- Under the squared difference loss function:

$$L(y_i, g_{t-1}(x_i) + \alpha_t f_t(x_i;\gamma_t)) =$$
$$= (\underbrace{y_i - g_{t-1}(x_i)}_{\text{fixed}} - \alpha_t f_t(x_i;\gamma_t))^2$$

- Forward stage-wise optimization seems to produce classifier with better generalization, doing the process stagewise seems to overfit less quickly

Boosting As Additive Model

$$g(x) = \sum_{k=1}^{M} \alpha_k f_k(x; \gamma_k)$$

- It can be shown that AdaBoost uses forward stage-wise modeling under the following loss function:
  - $L(y, g(x)) = \exp(-y \cdot g(x))$ -- the exponential loss function
  - At stage (or iteration) ***m***, we fit:

$$\arg\min_{\alpha_m, f_m} \sum_{i=1}^{N} L(y_i, g(x_i)) =$$

$$= \arg\min_{\alpha_m, f_m} \sum_{i=1}^{N} \exp(-y_i \cdot [g_{m-1}(x_i) + \alpha_m \cdot f_m(x_i)])$$

$$= \arg\min_{\alpha_m, f_m} \sum_{i=1}^{N} \exp(-y_i \cdot g_{m-1}(x_i)) \cdot \exp(-y_i \cdot \alpha_m \cdot f_m(x_i))$$

Exponential Loss vs. Squared Error Loss

- $L(y, g(x)) = \exp(-y \cdot g(x))$
- $L(y, g(x)) = (y - g(x))^2$

- Squared Error Loss penalizes classifications that are "too correct", with $y \cdot g(x) > 1$, and thus it is inappropriate for classification
- Exponential loss encourages large margins, want $y \cdot g(x)$ large

Logistic Regression Model

- It can be shown that Adaboost builds a logistic regression model:

$$g(x) = log \frac{Pr(Y = 1 / x)}{Pr(Y = -1 / x)} = \sum_{k=1}^{M} \alpha_m f_m(x)$$

- It can also be shown that the the training error on the samples is at most:

$$\sum_{i=1}^{N} exp(-y_i \cdot g(x_i)) = \sum_{i=1}^{N} exp\left(-y_i \cdot \sum_{k=1}^{M} \alpha_m f_m(x_i)\right)$$

Practical Advantages of AdaBoost

- fast
- simple
- Has only one parameter to tune (T)
- flexible: can be combined with any classifier
- provably effective (assuming weak learner)
  - shift in mind set: goal now is merely to find hypotheses that are better than random guessing
- finds outliers
  - The hardest examples are frequently the "outliers"

Caveats

- performance depends on data & weak learner
- AdaBoost can fail if
  - weak hypothesis too complex (overfitting)
  - weak hypothesis too weak ($\gamma_t \rightarrow 0$ too quickly),
    - underfitting
    - Low margins $\rightarrow$ overfitting
- empirically, AdaBoost seems especially susceptible to noise