CS9840
Learning and Computer Vision
Prof. Olga Veksler

Lecture 4

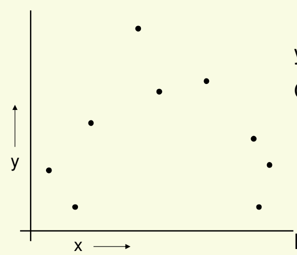
Cross Validation, Bagging and Boosting

Cross Validation slides are from Andrew Moore (CMU)

Some slides are due to Robin Dhamankar

Vandi Verma & Sebastian Thrun

A Regression Problem



y = f(x) + noise

Can we learn f from this data?

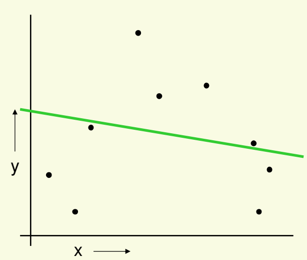
Let's consider three methods…

from Andrew Moore (CMU)

Today

- New Machine Learning Topics:
  1) Performance evaluation methods
     - cross-validation
  2) 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

Linear Regression



from Andrew Moore (CMU)

Linear Regression

Univariate Linear regression with a constant term:

X	Y
3	7
1	3
:	:

$$\mathbf{X}=\begin{bmatrix}3\\1\\:\end{bmatrix} \qquad \mathbf{y}=\begin{bmatrix}7\\3\\:\end{bmatrix}$$

$\mathbf{x}_1=(3)..$ $\qquad y_1=7..$

from Andrew Moore (CMU)

Linear Regression

Univariate Linear regression with a constant term:

X	Y
3	7
1	3
:	:

$$\mathbf{X}=\begin{bmatrix}3\\1\\:\end{bmatrix} \qquad \mathbf{y}=\begin{bmatrix}7\\3\\:\end{bmatrix}$$

$y_1=7..$

$$\mathbf{Z}=\begin{bmatrix}1 & 3\\1 & 1\\: & \end{bmatrix} \qquad \mathbf{y}=\begin{bmatrix}7\\3\\:\end{bmatrix}$$

$\mathbf{z}_1=(1,3)..$ $\qquad y_1=7..$

$\mathbf{z}_k=(1,x_k)$

$$\beta=(\mathbf{Z}^T\mathbf{Z})^{-1}(\mathbf{Z}^T\mathbf{y})$$

$$y^{est} = \beta_0+ \beta_1\, x$$

from Andrew Moore (CMU)

Linear Regression

Univariate Linear regression with a constant term:

X	Y
3	7
1	3
:	:

$$\mathbf{X}=\begin{bmatrix}3\\1\\:\end{bmatrix} \qquad \mathbf{y}=\begin{bmatrix}7\\3\\:\end{bmatrix}$$

$y_1=7..$

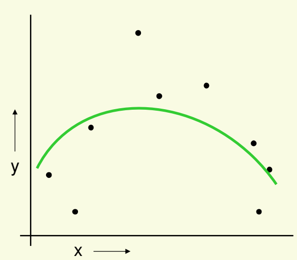
$$\mathbf{Z}=\begin{bmatrix}1 & 3\\1 & 1\\: & \end{bmatrix} \qquad \mathbf{y}=\begin{bmatrix}7\\3\\:\end{bmatrix}$$

$\mathbf{z}_1=(1,3)..$ $\qquad y_1=7..$

$\mathbf{z}_k=(1,x_k)$

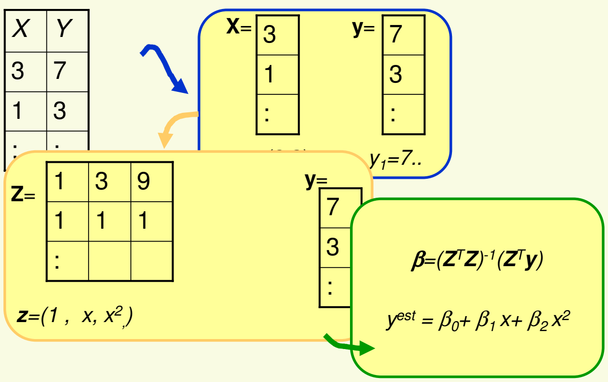
from Andrew Moore (CMU)

Quadratic Regression



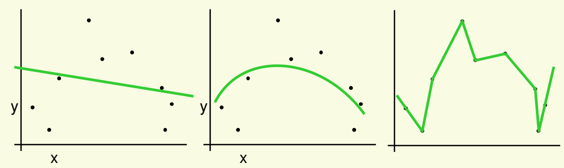
from Andrew Moore (CMU)

Quadratic Regression



from Andrew Moore (CMU)

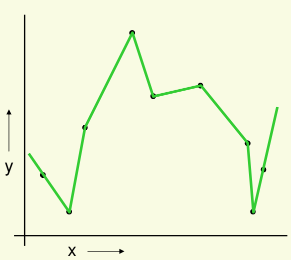
Which is best?



Why not choose the method with the best fit to the data?

from Andrew Moore (CMU)

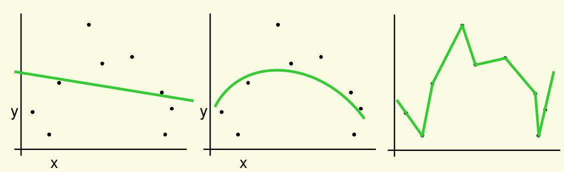
Join-the-dots



Also known as piecewise linear nonparametric regression if that makes you feel better

from Andrew Moore (CMU)

What do we really want?

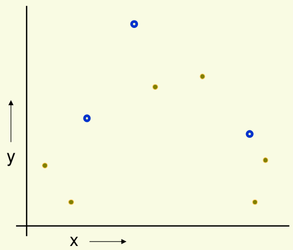


Why not choose the method with the best fit to the data?

"How well are you going to predict future data drawn from the same distribution?"

from Andrew Moore (CMU)

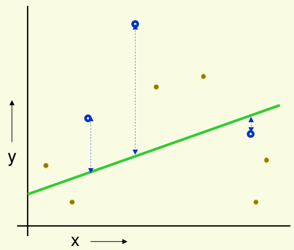
The test set method



1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set

from Andrew Moore (CMU)

The test set method



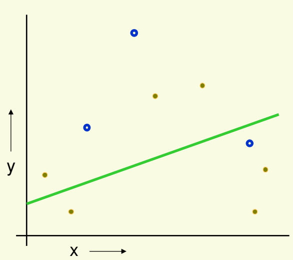
(Linear regression example)

Mean Squared Error = 2.4

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

from Andrew Moore (CMU)

The test set method

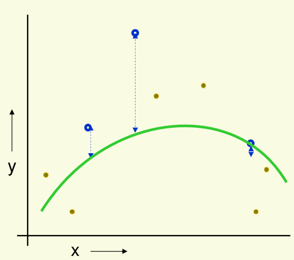


(Linear regression example)

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set

from Andrew Moore (CMU)

The test set method



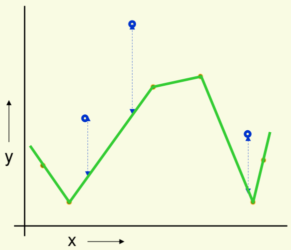
(Quadratic regression example)

Mean Squared Error = 0.9

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

from Andrew Moore (CMU)

The test set method



(Join the dots example)

Mean Squared Error = 2.2

1. Randomly choose 30% of the data to be in a test set

2. The remainder is a training set

3. Perform your regression on the training set

4. Estimate your future performance with the test set

from Andrew Moore (CMU)

The test set method

- Good news:
- Very very simple
- Can then simply choose the method with the best test-set score
- Bad news:
- Wastes data: we get an estimate of the best method to apply to 30% less data
  - if we don't have much data, our test-set might just be lucky or unlucky

We say the "test-set estimator of performance has high variance"

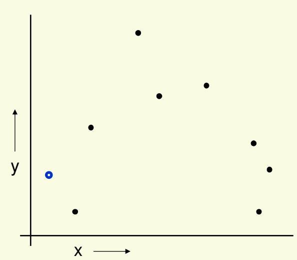
from Andrew Moore (CMU)

The test set method

- Good news:
- Very very simple
- Can then simply choose the method with the best test-set score
- Bad news:
- What's the downside?

from Andrew Moore (CMU)

LOOCV (Leave-one-out Cross Validation)

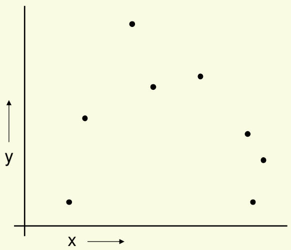


For k=1 to R

1. Let (x_k, y_k) be the k^{th} record

from Andrew Moore (CMU)

LOOCV *(Leave-one-out Cross Validation)*

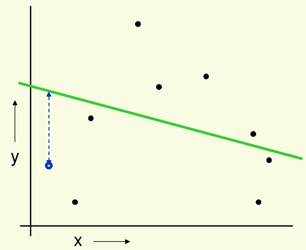


For k=1 to R

1. Let (x_k,y_k) be the k^{th} record
2. Temporarily remove (x_k,y_k) from the dataset

from Andrew Moore (CMU)

LOOCV *(Leave-one-out Cross Validation)*

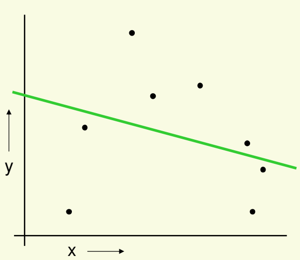


For k=1 to R

1. Let (x_k,y_k) be the k^{th} record
2. Temporarily remove (x_k,y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k,y_k)

from Andrew Moore (CMU)

LOOCV *(Leave-one-out Cross Validation)*

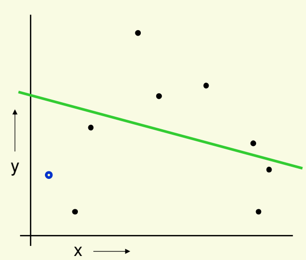


For k=1 to R

1. Let (x_k,y_k) be the k^{th} record
2. Temporarily remove (x_k,y_k) from the dataset
3. Train on the remaining R-1 datapoints

from Andrew Moore (CMU)

LOOCV *(Leave-one-out Cross Validation)*



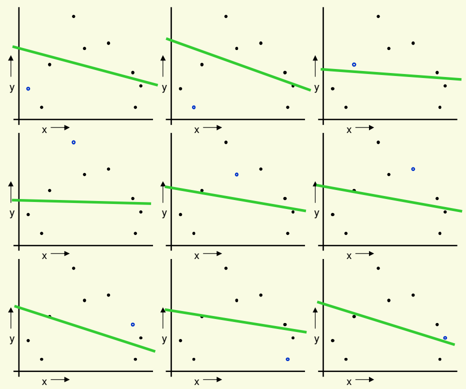
For k=1 to R

1. Let (x_k,y_k) be the k^{th} record
2. Temporarily remove (x_k,y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k,y_k)

When you've done all points, report the mean error.

from Andrew Moore (CMU)

LOOCV (Leave-one-out Cross Validation)



For k=1 to R

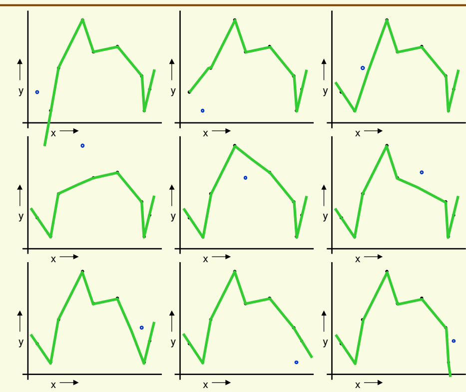
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$MSE_{LOOCV} = 2.12$

from Andrew Moore (CMU)

LOOCV for Join The Dots



For k=1 to R

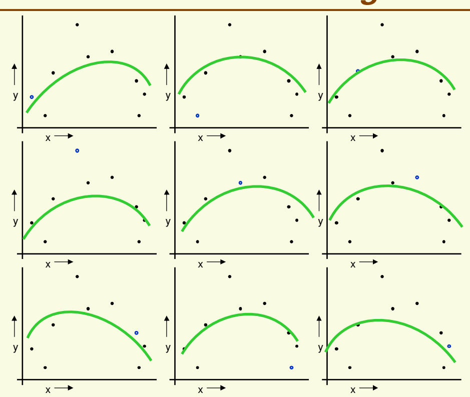
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$MSE_{LOOCV} = 3.33$

from Andrew Moore (CMU)

LOOCV for Quadratic Regression



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$MSE_{LOOCV} = 0.962$

from Andrew Moore (CMU)

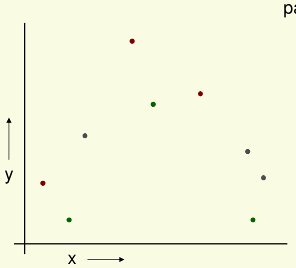
Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive	Doesn't waste data

..can we get the best of both worlds?

from Andrew Moore (CMU)

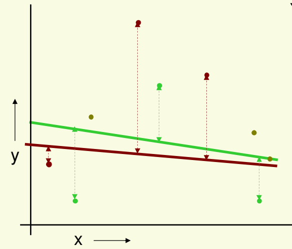
k-fold Cross Validation



Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

from Andrew Moore (CMU)

k-fold Cross Validation

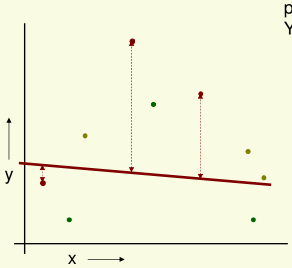


Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Yellow)

- For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.
- For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

from Andrew Moore (CMU)

k-fold Cross Validation

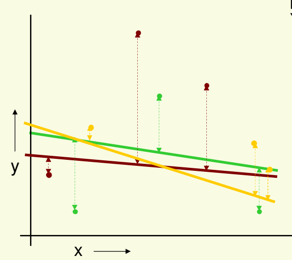


Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Yellow)

- For the red partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

from Andrew Moore (CMU)

k-fold Cross Validation

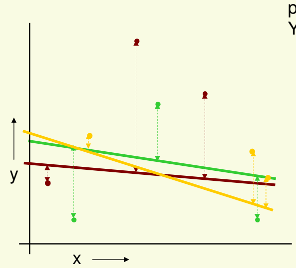


Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Yellow)

- For the red partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.
- For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.
- For the yellow partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

from Andrew Moore (CMU)

k-fold Cross Validation



Linear Regression
$MSE_{3FOLD}=2.05$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Yellow)

For the red partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

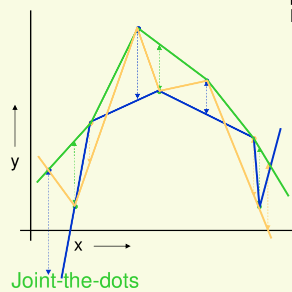
For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the yellow partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

Then report the mean error

from Andrew Moore (CMU)

k-fold Cross Validation



Joint-the-dots
$MSE_{3FOLD}=2.93$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Yellow Green and Blue)

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

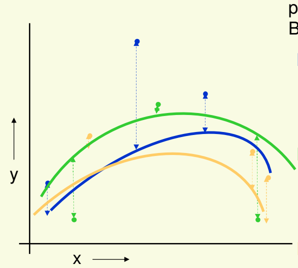
For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the gray partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

Then report the mean error

from Andrew Moore (CMU)

k-fold Cross Validation



Quadratic Regression
$MSE_{3FOLD}=1.11$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Yellow, Green and Blue)

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the yellow partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

Then report the mean error

from Andrew Moore (CMU)

Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than test set	Only wastes 10%. Only 10 times more expensive instead of R times.
3-fold	Wastier than 10-fold. Expensivier than test set	Slightly better than test-set
N-fold	Identical to Leave-one-out	

from Andrew Moore (CMU)

CV-based Model Selection

- We're trying to decide which algorithm to use.
- We train each machine and make a table…

i	f_i	TRAINERR	10-FOLD-CV-ERR	Choice
1	f_1			
2	f_2			
3	f_3			⌧
4	f_4			
5	f_5			
6	f_6			

from Andrew Moore (CMU)

CV-based Model Selection

- Example: Choosing "k" for a k-nearest-neighbor regression.
- Step 1: Compute LOOCV error for six different model classes:

Algorithm	TRAINERR	10-fold-CV-ERR	Choice
K=1			
K=2			
K=3			
K=4			⌧
K=5			
K=6			

- Step 2: Whichever model class gave best CV score: train it with all the data, and that's the predictive model you'll use.

from Andrew Moore (CMU)

CV-based Model Selection

- Example: Choosing number of hidden units in a one-hidden-layer neural net.
- Step 1: Compute 10-fold CV error for six different model

Algorithm	TRAINERR	10-FOLD-CV-ERR	Choice
0 hidden units			
1 hidden units			
2 hidden units			⌧
3 hidden units			
4 hidden units			
5 hidden units			

- Step 2: Whichever model class gave best CV score: train it with all the data, and that's the predictive model you'll use.

from Andrew Moore (CMU)

CV-based Model Selection

- Example: Choosing "k" for a k-nearest-neighbor regression.
- Step 1: Cor
  classes:

Algorithm	TR
K=1	
K=2	
K=3	
K=4	
K=5	
K=6	

Why did we use 10-fold-CV for neural nets and LOOCV for k-nearest neighbor?

The reason is Computational. For k-NN (and all other nonparametric methods) LOOCV happens to be as cheap as regular predictions.

And why stop at K=6

No good reason, except it looked like things were getting worse as K was increasing

Are we guaranteed that a local optimum of K vs LOOCV will be the global optimum?

Sadly, no. And in fact, the relationship can be very bumpy.

What should we do if we are depressed at the expense of doing LOOCV for K= 1 through 1000?

Idea One: K=1, K=2, K=4, K=8, K=16, K=32, K=64 … K=1024

Idea Two: Hillclimbing from an initial guess at K

- Step 2: Whichever model class gave best CV score: train it with all the data, and that's the predictive model you'll use.

from Andrew Moore (CMU)

CV-based Model Selection

- Can you think of other decisions we can ask Cross Validation to make for us, based on other machine learning algorithms in the class so far?

Cross-validation for classification

- Instead of computing the sum squared errors on a test set, you should compute…

from Andrew Moore (CMU)

CV-based Algorithm Choice

- Example: Choosing which regression algorithm to use
- Step 1: Compute 10-fold-CV error for six different model classes:

Algorithm	TRAINERR	10-fold-CV-ERR	Choice
1-NN			
10-NN			
Linear Reg'n			
Quad reg'n			☒
LWR, KW=0.1			
LWR, KW=0.5			

- Step 2: Whichever algorithm gave best CV score: train it with all the data, and that's the predictive model you'll use.

from Andrew Moore (CMU)

Cross-validation for classification

- Instead of computing the sum squared errors on a test set, you should compute…

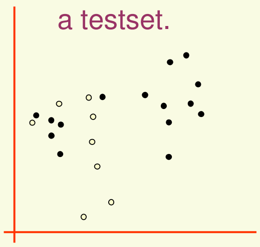
  The total number of misclassifications on a testset.

from Andrew Moore (CMU)

Cross-validation for classification

- Instead of computing the sum squared errors on a test set, you should compute…

  The total number of misclassifications on a testset.



- What's LOOCV of 1-NN?
- What's LOOCV of 3-NN?
- What's LOOCV of 22-NN?

from Andrew Moore (CMU)

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 trainig 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

Cross-Validation for classification

- Choosing k for k-nearest neighbors
- Choosing h for the Parzen windows
- Any other "free" parameter of a classifier
- Choosing which classifier to use
- Choosing Features to use

from Andrew Moore (CMU)

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

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

Ada Boost

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

$$g(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

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

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 $\sum_{i=1} d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}\left[\sum \alpha_t f_t(x)\right]$

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 $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}\left[\sum \alpha_t f_t(x)\right]$

- 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
    $\varepsilon_t = \sum d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$
  - 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 $\sum_{t+1} d(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}\left[\sum \alpha_t f_t(x)\right]$

- 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
    $\varepsilon_t = \sum d(x_i) \cdot I(y_i \neq f_t(x_i))$
  - 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 $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign}\left[\sum \alpha_t f_t(x)\right]$

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

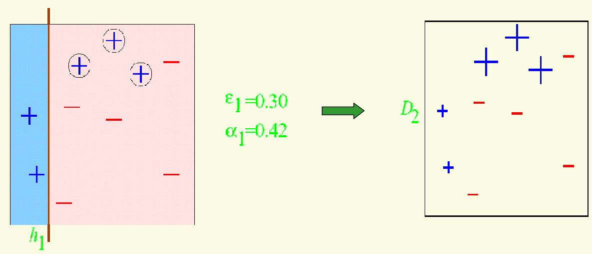
Ada Boost

- At each iteration t :
  - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
  - Compute ε_t the error rate as
    $\varepsilon_t = \sum d_t(x_i) \cdot \mathrm{I}(y_i \neq f_t(x_i))$
  - 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, $d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot \mathrm{I}(y_i \neq f_t(x_i))]$
  - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \mathrm{sign}[\sum \alpha_t f_t(x)]$

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

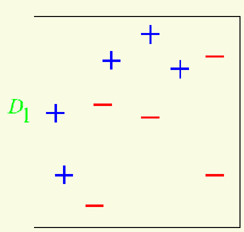
AdaBoost Example

ROUND 1



AdaBoost Example

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

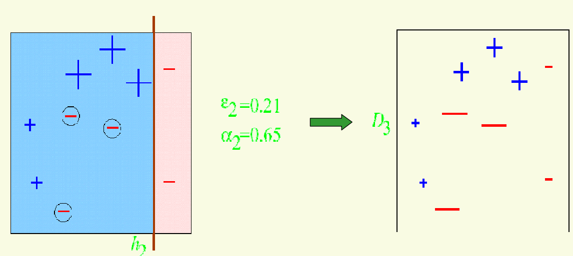


Original Training set : equal weights to all training samples

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

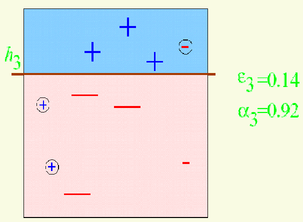
AdaBoost Example

ROUND 2



AdaBoost Example

ROUND 3



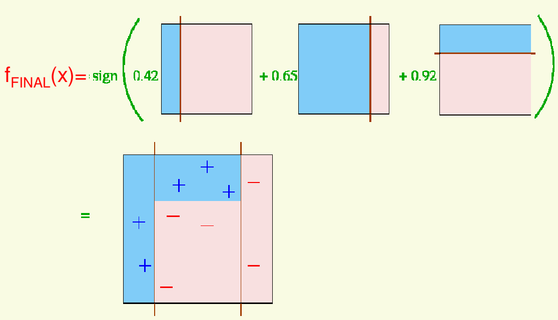
AdaBoost Comments

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

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

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

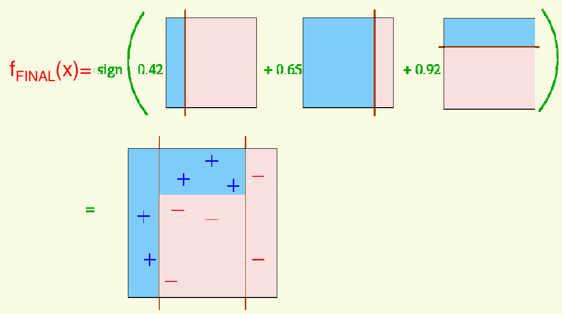
AdaBoost Example



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



Boosting As Additive Model

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

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

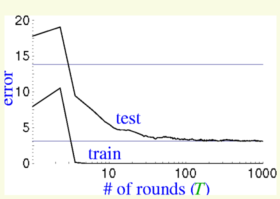
- Typically we would try to minimize a loss function on the N training examples

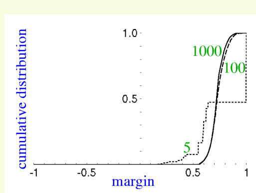
$$\min_{\alpha_1,\gamma_1,\ldots,\gamma_M,\alpha_M} \sum_{i=1}^{N} L\left(y_i, \sum_{k=1}^{M} \alpha_k f_k(x_i;\gamma_k)\right)$$

- For example, under squared-error loss:

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

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

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

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

- Under the squared difference loss function:

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

- 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 · g (x)) -- the exponential loss function
  - At stage (or iteration) ***m***, we fit:

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

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

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

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:

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

Exponential Loss vs. Squared Error Loss

- L(y, g (x)) = exp(-y · 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

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 ($\gamma_t \to 0$ too quickly),
    - underfitting
    - Low margins $\to$ overfitting
- empirically, AdaBoost seems especially susceptible to noise