Performance and Generalization

Classifier Performance

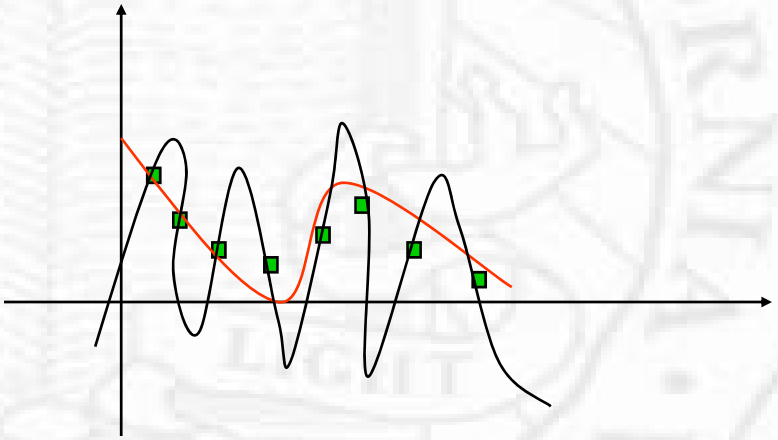
- Intuitively, performance of classifiers (learning algorithms) depends on
  - Complexity of the classifiers (e.g., how many layers and how many neurons per layers)
  - Training samples (generally more is better)
  - Training procedures (e.g., how many searches/epochs are allowed)
  - Etc.

Generalization Performance

- You can make a classifier performs very well on any *training* data set
  - Given enough structure complexity
  - Given enough training cycles
- But how does it do on a *validation* (unseen) data set?
  - Or how is the *generalization* performance?

Generalization Performance (cont.)

- First, try to do better on unseen data by doing better on training data might not work
- Because overfitting can be a problem
  - You can fit the training data arbitrarily well, but there is no prediction of what it will do on data not seen
- Example: curve fitting



  - Using a large network or complicated classifier does not necessarily lead to good generalization (they almost always lead to good training results)

Generalization Performance (cont.)

- In fact, some relations must exist in the data set even when the data set is made of random numbers
- Example: given *n* people, each
  - Has a credit card
  - Has a phone
  - The credit card and phone number *association* is captured by an *n-1*-degree polynomial
  - But can you extrapolate (predict other credit card, phone number association)?
  - A problem of overfitting

Intuitively

- Meaningful associations usually imply
  - *Simplicity (capacity)*
    - The association function should be simple
    - More generally to determine how much capability a classifier possesses
  - *Repeatability (stability)*
    - The association function should not change drastically when different training data sets are used to derive the function, or E(f)=0 (over different data set)
    - Average salary of Ph.D. is higher than that of high-school dropout – simple and repeatable relation (not sensitive to the particular training data set)

Generalization Performance (cont.)

- So does that mean we should always prefer simplicity?
- Occam's Razor: nature prefers simplicity
  - Explanations should not be multiplied beyond necessity
- Sometimes, it is a bias or preference over the forms and parameters of a classifier

No free lunch theorem

- Under very general assumption, one should not prefer one classifier (or learning algorithm) over another for the *generalization* performance
- Why?
  - Because given certain training data, there is no telling (*in general*) what unseen data will behave

Example

- Training data might not provide any information about F(**x**)
- There are multiple (2^5) target functions that are consistent with the n=3 patterns in training set
- Each inversion of F (-F) will make one good and the other bad

target

x	F	h_1	h_2
000	1	1	1
001	-1	-1	-1
010	1	1	1
011	-1	1	-1
100	1	1	-1
101	-1	1	-1
110	1	1	-1
111	1	1	-1

training

Unseen 2^5 combinations

Example (cont.)

- However, in reality, we also expect that learning (training) is effective
  - given a *large* number of representative samples and
  - a target function that is smooth
- Or the sampling theorem says that extrapolation and reconstruction is possible under certain conditions

How do we reconcile?

- So the choice of a classifier or a training algorithm depends on *our preconceived notion of the relevant target functions*
- In that sense, both Occam's Razor and no free lunch theorem can co-exist

Make it More Concrete

- Let's assume that there are two classes {+1, -1} and *n* samples $(x_1, y_1) \ldots (x_n, y_n)$
- The classifier is f(x,α) -> y (α: tunable parameters of *f, e.g.,* hyperplane)
- Loss (error) is $\frac{1}{2}|y_i - f(x_i, \alpha)|$
- Then there are two types of errors (risks)
  - Empirical error $e_{emp}(\alpha) = \frac{1}{2n}\sum_{i=1}^{n}|y_i - f(x_i, \alpha)|$
  - Expected error $e(\alpha) = \frac{1}{2}\int |y - f(x, \alpha)|\, dP(x, y)$

Issues with the Two Risks

- How do we estimate how good is a classifier based on a particular run of the experiment?
  - How best to compute e_{emp} from samples
- How do we estimate e from $e_{emp?}$
  - Practically (with some particular data sets), and
  - Theoretically (with an upper bound)

Answers

- Computing e_{emp}
  - There are statistical resampling techniques to give better estimate of a classifier's performance
- Estimating e from e_{emp}
  - Theoretically
    - There is an upper bound on e given e_{emp}
  - Practically
    - Given a particular set(s) of training data, a procedure exists to estimate e from e_{emp}

Practical Issues

- OK, so philosophically nothing is better than anything else
- In reality, we have to choose classifiers and learning algorithms
- Run some classification experiments in supervised mode (with labeled data)
- What should we look for in a "good" classifier or learning algorithm?

Assumption

- Use RBF interpretation
  - Interpolation a function with given data points
    - True function $F(\mathbf{x})$
    - Interpolation function $g(\mathbf{x}; D)$
    - D is the given data set

Bias and Variance

$$E_D\left[(g(\mathbf{x};D)-F(\mathbf{x}))^2\right]=(E_D[g(\mathbf{x};D)-F(\mathbf{x})])^2+E_D\left[(g(\mathbf{x};D)-E_D[g(\mathbf{x};D)])^2\right]$$

bias^2 variance

$$E_D\left[(g(\mathbf{x};D)-F(\mathbf{x}))^2\right]$$

$$=E_D[g^2]-2E_D[g]F+F^2$$

$$=E_D[g^2]-2E_D[g]F+F^2+(E_D[g])^2-2(E_D[g])^2+(E_D[g])^2$$

=0

$$=(E_D[g])^2-2E_D[g]F+F^2+E_D[g^2]-2E_D[g]E_D[g]+(E_D[g])^2$$

$$=(E_D[g-F])^2+E_D(g-E_D[g])^2$$

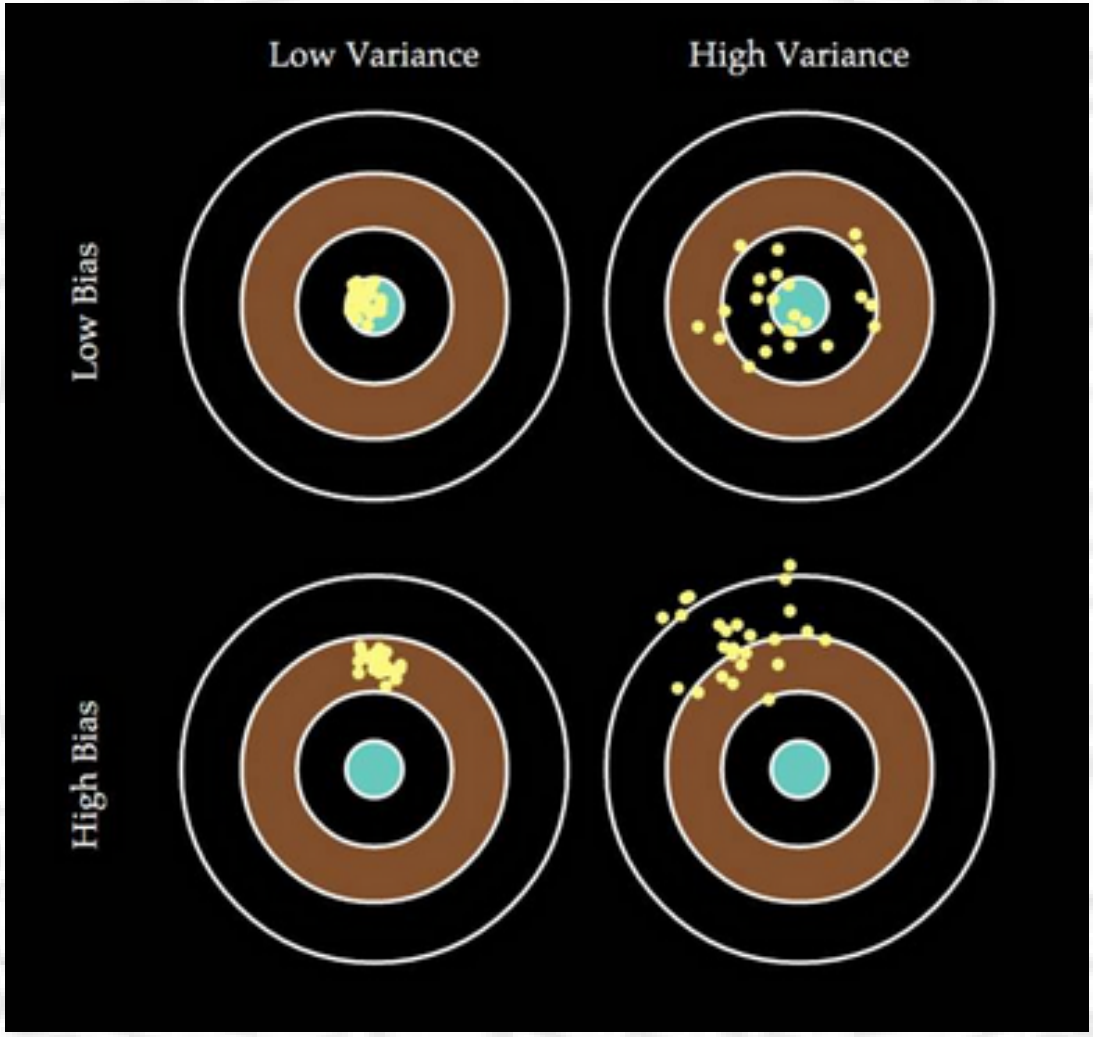
Bias and Variance

$$E_D\left[(g(\mathbf{x};D)-F(\mathbf{x}))^2\right]=(E_D[g(\mathbf{x};D)-F(\mathbf{x})])^2+E_D\left[(g(\mathbf{x};D)-E_D[g(\mathbf{x};D)])^2\right]$$

bias^2 variance

- Bias – measure the *accuracy*
  - How good are the classifiers confirm to *reality*
  - High bias implies a *poor* match
- Variance – measure the *precision* or specificity of a match
  - How good is the classifiers confirm to *one another*
  - High variance implies a *weak* match

Graphic Interpretation



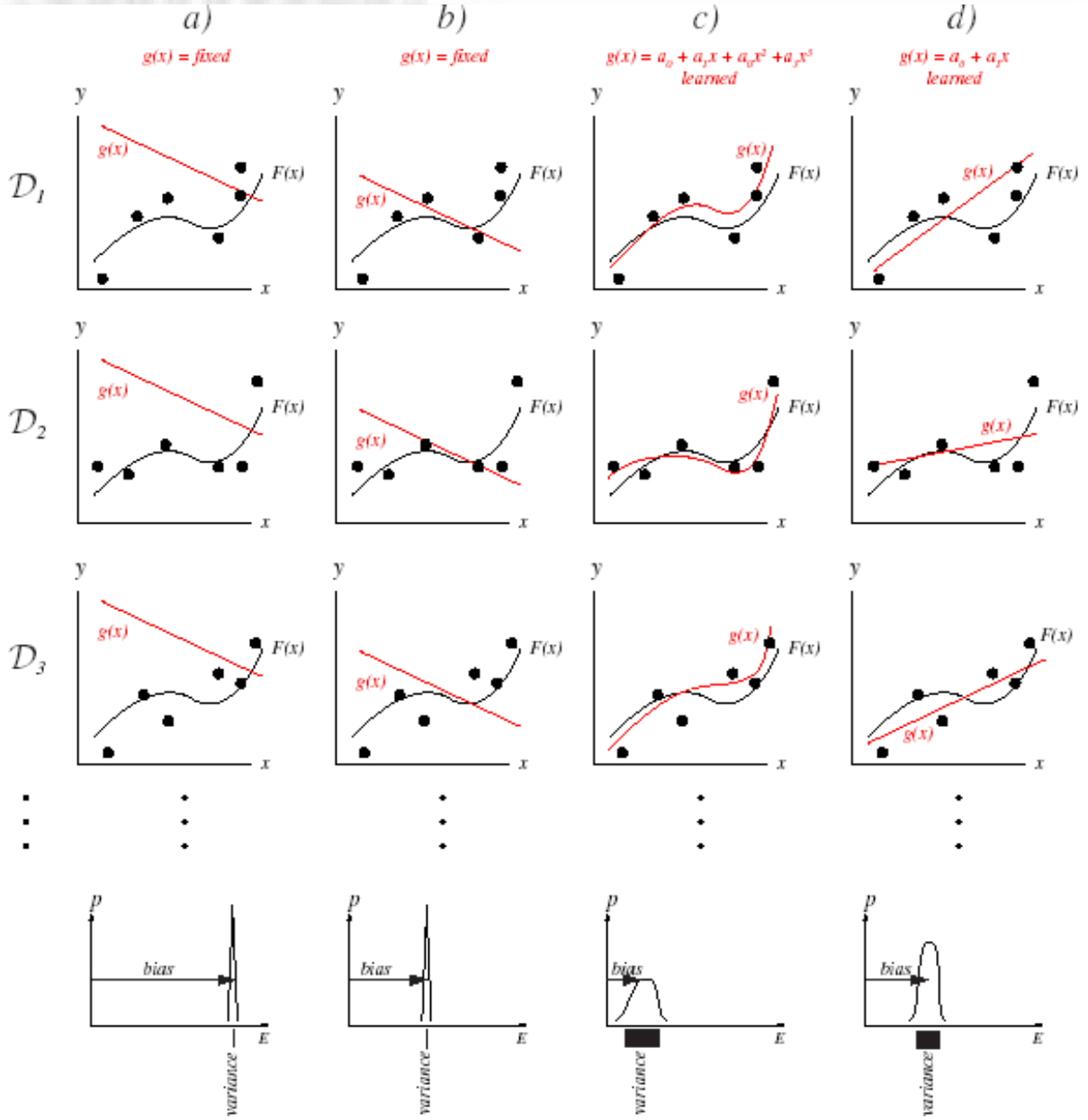
Tradeoff

- Intuitively, increasing the flexibility (more parameters)
  - Gives better fit (low bias)
  - Produces harder to control fit (high variance)
- Bias-variance tradeoff is a lot like precision-recall, you cannot have both

Bias and Variance in Fitting

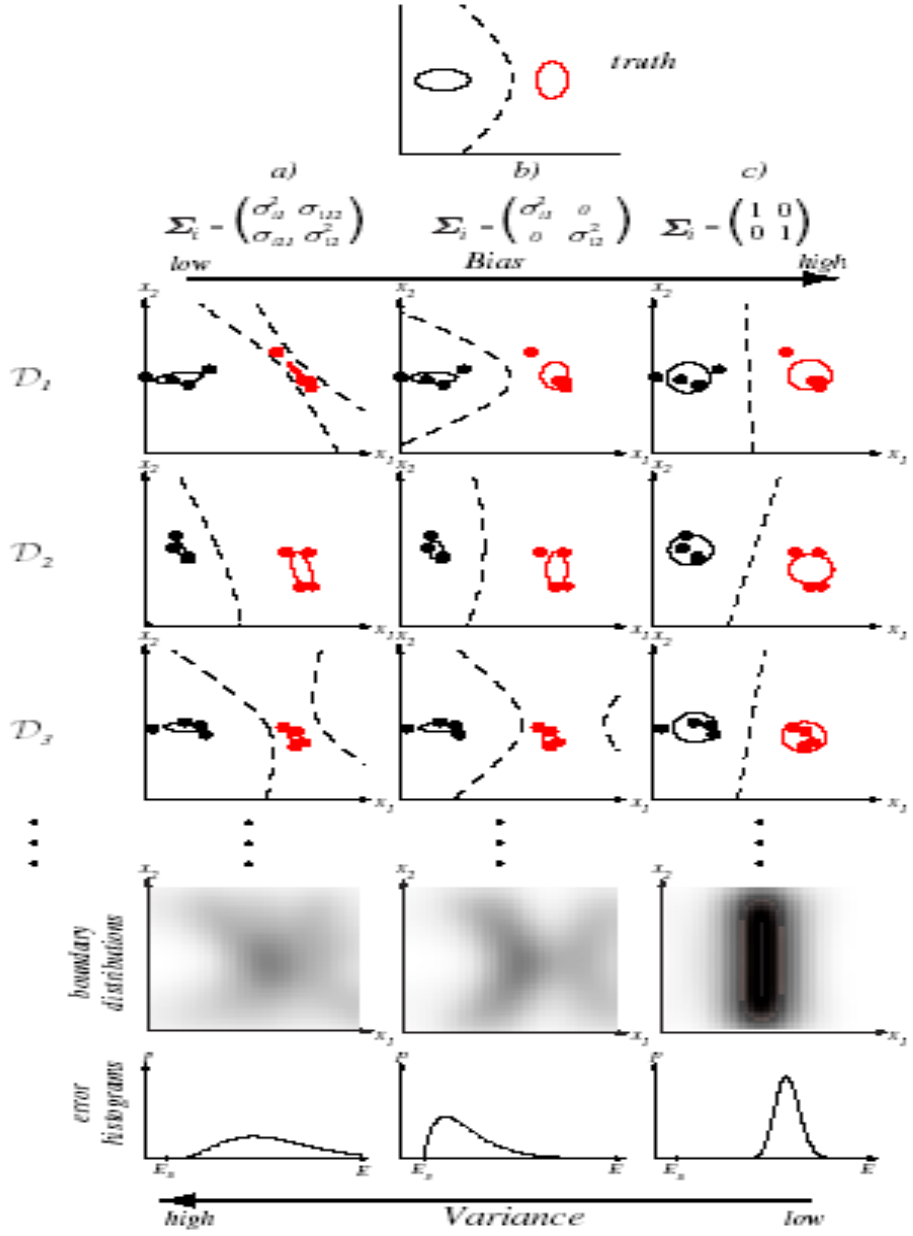
- With more parameters, a better fit (low bias) is possible, at the expense that parameter values may vary widely given different sampled data (high variance)

a, b: fixed linear model
c: learned cubic model
d: learned linear model



Bias and Variance in Classifiers

- Simple models
  - capture *aggregate* class properties which are usually more stable (hence low variance)
  - However, they miss *fine* details and give poor fit (hence high bias)
- Complicated models
  - capture *aggregate* class properties and *fine* variance (hence low bias)
  - However, fine details depend on samples used (hence high variance)



Curse of Dimensionality

- What happens to bias and variance when the dimensionality increase?
- It depends
- F(**X**) depends on all dimensions of **X**
  - Bias is likely to go up for nn classifier because neighbors will be further away from a data point for faithful interpolation
- F(**X**) depends on some dimensions of **X**
  - Variance is likely to go up for nn classifier because spread along the used dimensions might go up

Practical Issues (cont.)

- So you choose a labeled data set and test your classifier
- But that is just on one particular data set
- How do you know how it will do for other labeled data set? or
- How do you *estimate* bias and variance? (or how do you know the particular relation is *stable* and *repeatable*?)
  - We do not really know F for other data sets
  - We have only one data set, not an ensemble
- How do you extrapolate (from e_{emp} to e)?
- How do you *improve* bias and variance?

Example: Estimation - Jackknife

- Perform many "leave-one-out" estimations
- E.g., to estimate mean and variance

Traditional	Leave-one-out
$\hat{u} = \frac{1}{n}\sum_{i=1}^{n} x_i$	$u_{(.)} = \frac{1}{n}\sum_{i=1}^{n} u_{(i)}$ *where* $u_{(i)} = \frac{1}{n-1}\sum_{j=1, j\neq i}^{n} x_j$
$\hat{\sigma}^2 = \frac{n-1}{n}\sum_{i=1}^{n}(x_i - \hat{u})^2$	$Var[\hat{u}] = \frac{n-1}{n}\sum_{i=1}^{n}(u_{(i)} - u_{(.)})^2$
This does *not* work for other statistics, such as median and mode	This is applicable to other statistics, such as median and mode

Estimation – Jackknife (cont.)

- Jackknife estimation is defined as function of leave-one-out results
- Enable mean and variance computation from one data set

$$
\begin{aligned}
u_{(.)} &= \frac{1}{n}\sum_i u_{(i)} \\
&= \frac{1}{n}\sum_i \frac{n\hat{u} - x_i}{n-1} \\
&= \frac{1}{n}\frac{n^2\hat{u} - \sum_i x_i}{n-1} \\
&= \frac{1}{n}\frac{n^2\hat{u} - n\hat{u}}{n-1} \\
&= \hat{u}
\end{aligned}
$$

$$
\begin{aligned}
Var[\hat{u}] &= \frac{n-1}{n}\sum_i (u_{(i)} - u_{(.)})^2 \\
&= \frac{n-1}{n}\sum_i (u_{(i)} - \hat{u})^2 \\
&= \frac{n-1}{n}\sum_i (\frac{n\hat{u} - x_i}{n-1} - \hat{u})^2 \\
&= \frac{n-1}{n}\sum_i (\frac{\hat{u} - x_i}{n-1})^2 = \hat{\sigma}^2
\end{aligned}
$$

General Jackknife Estimation

- Similar to mean and variance estimation
- Perform many leave-one-out estimations of the parameter

$$\hat{\theta}_{(i)} = \hat{\theta}(x_1, x_2, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$$

$$\hat{\theta}_{(.)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}$$

e.g., θ can be the hyperplane equation

$$\text{var}[\hat{\theta}] = \frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(.)})^2$$

Bias and Variance of General Jackknife Estimation

$$Bias = \theta - E(\hat{\theta})$$

$$bias_{jackknife} = (n-1)(\hat{\theta}_{(.)} - \hat{\theta})$$

$$Var[\hat{\theta}] = E(\hat{\theta} - E(\hat{\theta}))^2$$

$$Var_{jackknife}[\hat{\theta}] = \frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(.)})^2$$

Example: Mode Estimate

- n=6
- D={0,10,10,10,20,20}

$$\hat{\theta}_{(.)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)} = \frac{1}{6}\{10+15+15+15+10+10] = 12.5$$

$$bias_{jackknife} = (n-1)(\hat{\theta}_{(.)} - \hat{\theta}) = 5(12.5-10) = 12.5$$

$$Var_{jackknife}[\hat{\theta}] = \frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(.)})^2 = \frac{5}{6}\{3(10-12.5)^2 + 3(15-12.5)^2\} = 31.25$$

Mode: (1)most common elements, (2) two equal peaks, midpoint btw
If two elements are equally likely, the mode is the midway point

Estimation - Bootstrap

- Perform many subset estimations (n out of n) *with* replacement
  - There are n^n possible samples
    - E.g., two samples (x1,x2) generate 2^2 subsets (x1,x1), (x1,x2), (x2,x1), (x2,x2)

$$\hat{\theta}^{*(.)} = \frac{1}{B}\sum_{b=1}^{B}\theta^{*(b)}$$

$$bias_{boot} = \frac{1}{B}\sum_{b=1}^{B}\theta^{*(b)} - \hat{\theta} = \hat{\theta}^{*(.)} - \hat{\theta}$$

$$\mathrm{var}_{boot}[\theta] = \frac{1}{B}\sum_{b=1}^{B}[\theta^{*(b)} - \hat{\theta}^{*(.)}]^2$$

The Question Remained

- From data sets we can estimate e_{emp}
- We desire e
- They are related by

$$e = e_{emp} \pm c\sqrt{\frac{e_{emp}(1-e_{emp})}{n}}$$

n : sample size

c : confidence interval

  - n=40
  - 28 correct and 12 incorrect
  - e_{emp} =12/40=.3
  - With a confidence interval of 95%, c=1.96

$$e = .3 \pm 1.96\sqrt{\frac{.3\times.7}{40}} = .3 \pm .14$$

What?

- The formula is valid if
  - Hypothesis is discrete-valued
  - Samples are independently drawn
  - With a fixed probabilistic distribution
- Then the experiment outcomes can be described by a binomial distribution

Comparison

- Repeat an experiment many times
- Each time, toss a coin n times to see if it lands on head (h) or tail (t) $h+t=n$
- A coin of an (*unknown*) probability p to land on head

- Use a classifier for many sets of data
- Each data set, the classifier gets h wrong and t correct out of n samples, $h+t=n$
- The classifier has an (unknown, ***but fixed***) probability p to classify data incorrectly

Comparison (cont.)

- Results of each coin toss is a random variable
- Results of how many heads in *n* toss is also a random variable
- Repetitive experiments give outcomes in binomial distribution

- Results of each sample classification is a random variable
- Results of how many incorrect labels in *n* samples is also a random variable
- Repetitive classifications give outcomes in binomial distribution

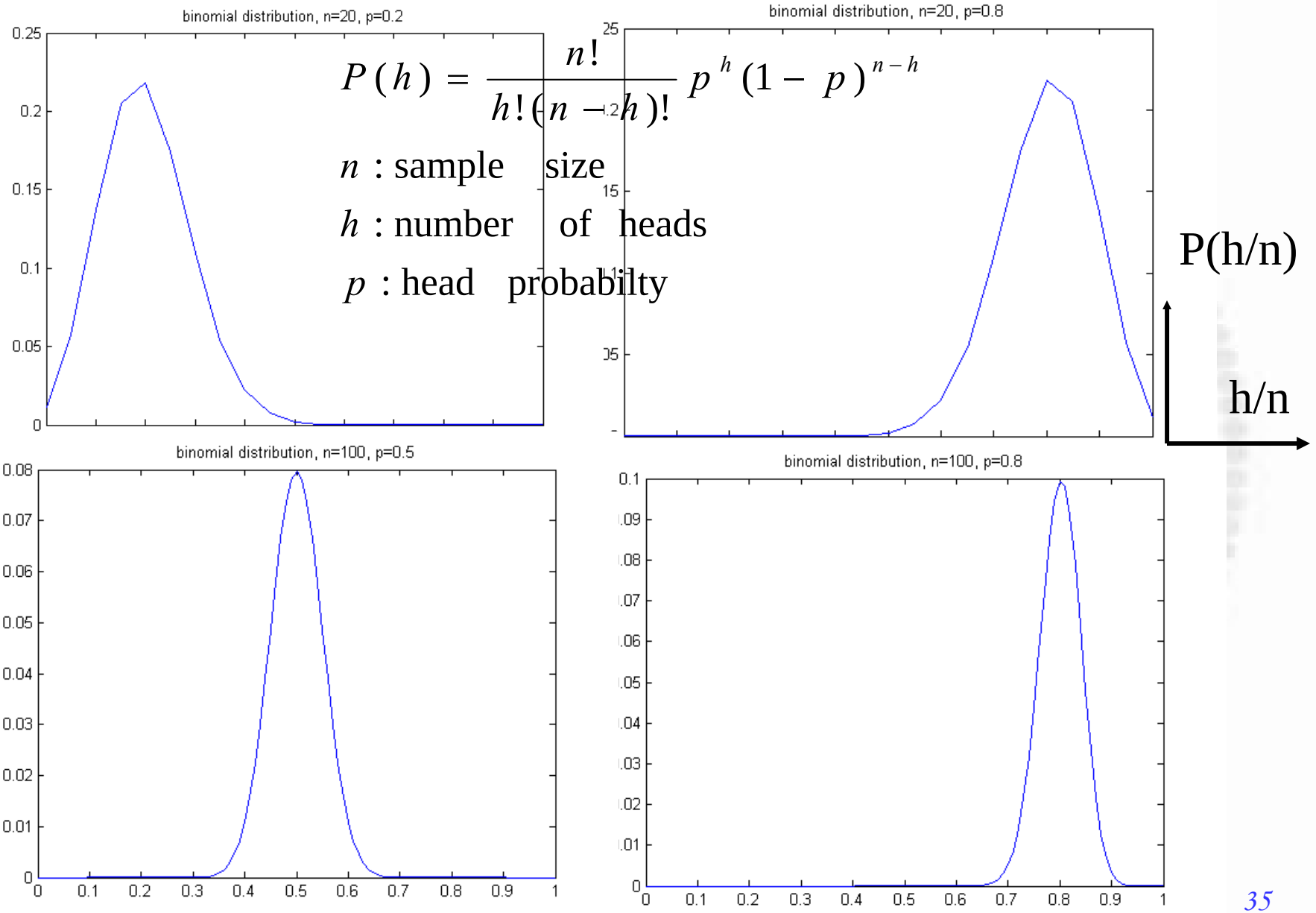
Binomial Distributions

$$P(h) = \frac{n!}{h!(n-h)!} p^h (1-p)^{n-h}$$

n : sample size

h : number of heads

p : head probabilty



Binomial Distribution

- Then it can be shown that

$$P(h) = \frac{n!}{h!(n-h)!} p^h (1-p)^{n-h}$$

$$E(h) = np$$

$$Var(h) = np(1-p)$$

$$\sigma_h = \sqrt{np(1-p)}$$

Estimators

- An estimator for p is (number of heads)/n
- This estimator is an unbiased estimator because

$$E(\frac{\#\text{head}}{\text{n}}) = \frac{np}{n} = p$$

- Standard deviation in the estimator is

$$\sigma(\frac{\#head}{n}) = \frac{\sqrt{np(1-p)}}{n} = \sqrt{\frac{p(1-p)}{n}}$$

- An estimator for $e\ (p)$ is e_{emp}
- This estimator is an unbiased estimator because

$$E\ (\frac{\#\ \text{error}}{\text{n}}) = \frac{np}{n} = p$$

- Standard deviation in the estimator is

$$\sigma(\frac{\#error}{n}) = \frac{\sqrt{np(1-p)}}{n} = \sqrt{\frac{p(1-p)}{n}}$$

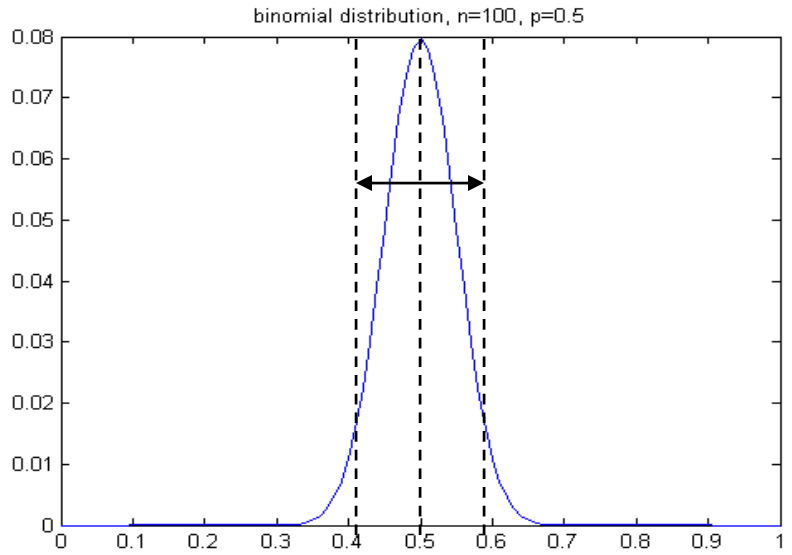
Confidence Interval

- An *N%* confidence interval for some parameter *p* is an interval that is expected with probability *N%* to contain *p*
- For binomial distribution, this can be approximated by normal

$$e = e_{emp} \pm c\sqrt{\frac{e_{emp}(1-e_{emp})}{n}}$$

n : sample size

c : confidence interval



Capacity (Simplicity)

- We have just discussed the *repeatability* issue
  - The assumption is that the classification error is the same for training and new data
  - The misclassification rate is drawn from the same population
  - True for simple classifiers, but not for complicated classifiers
- The other issue is *simplicity* (or more generally the *capacity*) of the classifier

General Theoretical Bound

- The sample size
  - The larger the sample size, the more confident we should be about "we have seen enough"
- The complexity of the classifier
  - The simpler the classifier, the more confident we should be about "we have observed enough"
  - Or complex classifier can do wired things when you are not looking ☺

VC Dimension

- Vapnik-Chervonenkis dimension
- Defined for a class of functions f(alpha)
- The maximum number of points that can be *shattered* by the function
  - Shatter means that given, say, *n* points, there are 2^n ways to label them {+1, -1}. These points are shattered if an f(alpha) can be found to correctly assign those labels
  - E.g., three points can be shattered by 1 line, but not four points
  - Linear function in *n* space is of VC dimension *n+1*
- A measure of "capacity" of a classifier

Generalization

- It can be shown that $e(\alpha) \le e_{emp}(\alpha) + \sqrt{\frac{h(\log(2n/h)+1) - \log(\eta/4)}{n}}$
- Basically, the expected error is bounded above, the bound depends on
  - Empirical error (e_{emp})
  - VC dimension (h)
  - n: training sample size
  - Expected performance (η), say loss of 0.05 with a probability of 0.95

Capacity Interpretation

$$e(\alpha) \le e_{emp}(\alpha) + \sqrt{\frac{h(\log(2n/h)+1) - \log(\eta/4)}{n}}$$

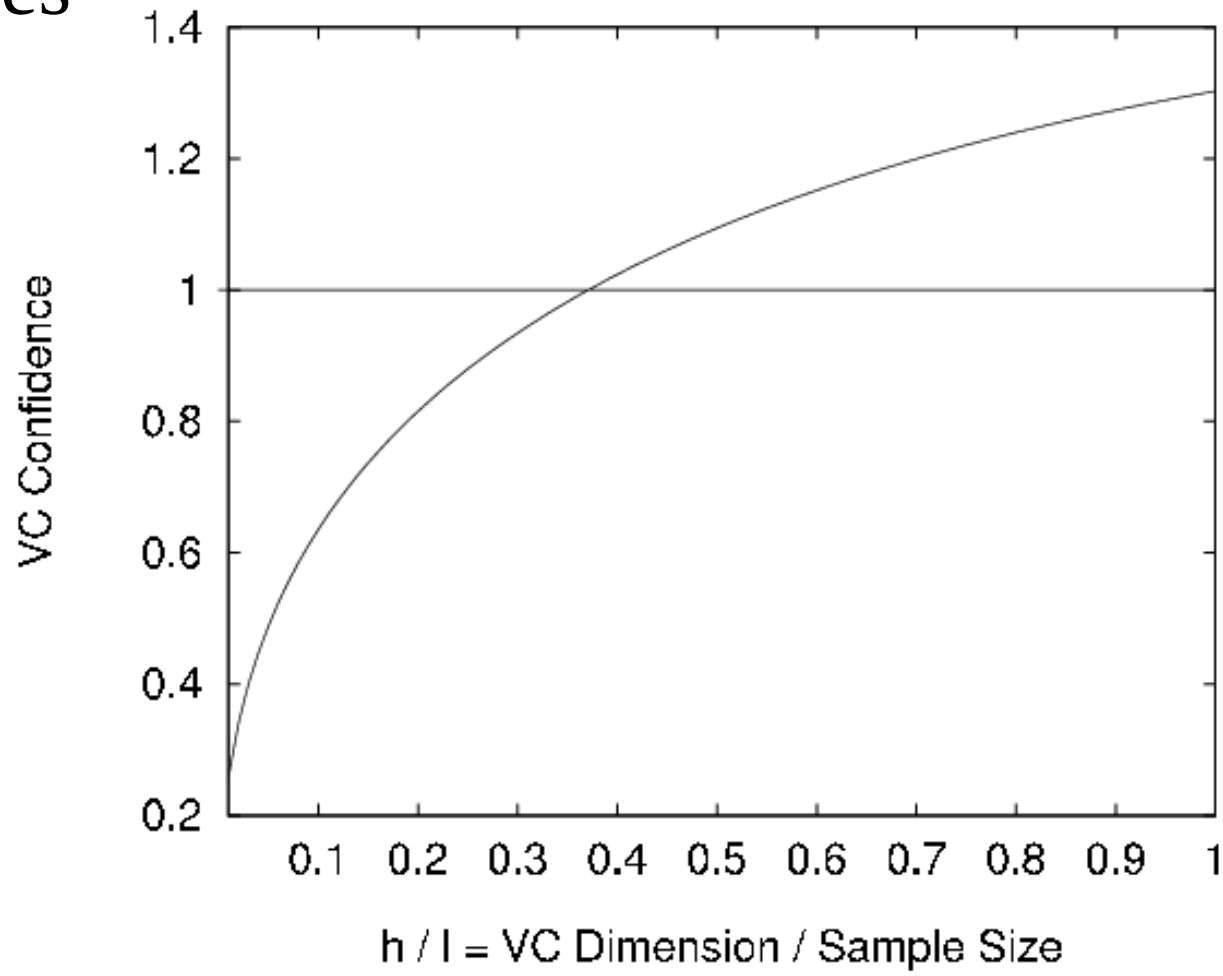
- A simple classifier
  - Has low VC dimension and small second term
  - Has high empirical error and large first term
- A complicated classifier
  - Has high VC dimension and large second term
  - Has low empirical error and small first term
- Some trade-off to achieve the lowest right-hand side

Generalization Performance

- Hence, a classifier should be chosen to give the lowest bound
- However, many times the bound is not tight, easily the bound can reach 1 and make it useless
- Only useful for small VC dimension

Upper Bound

- 95% confidence level
- 10,000 samples
- h/n >0.37

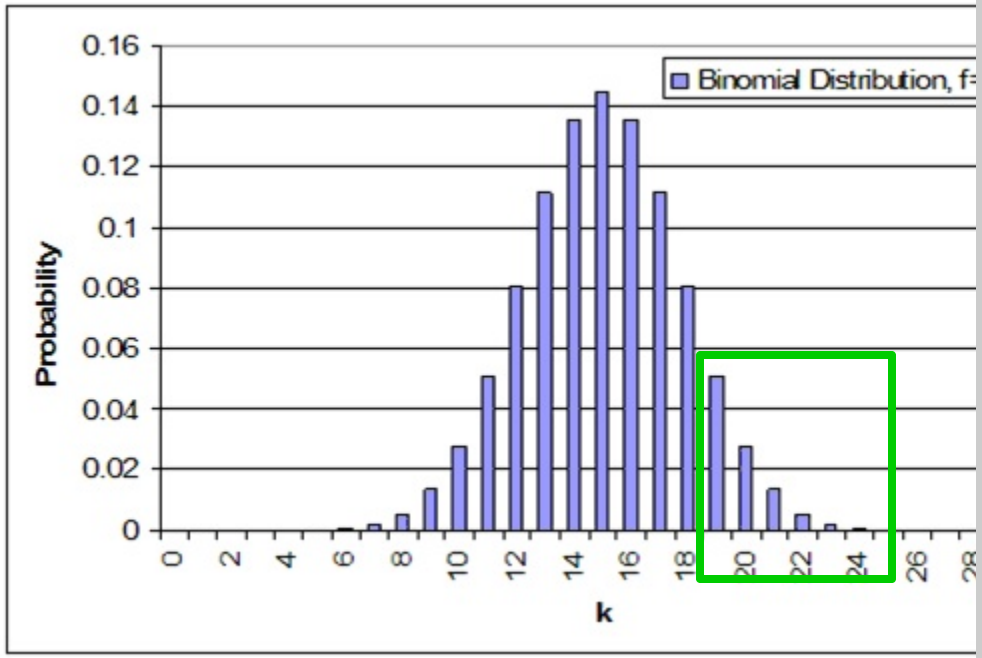


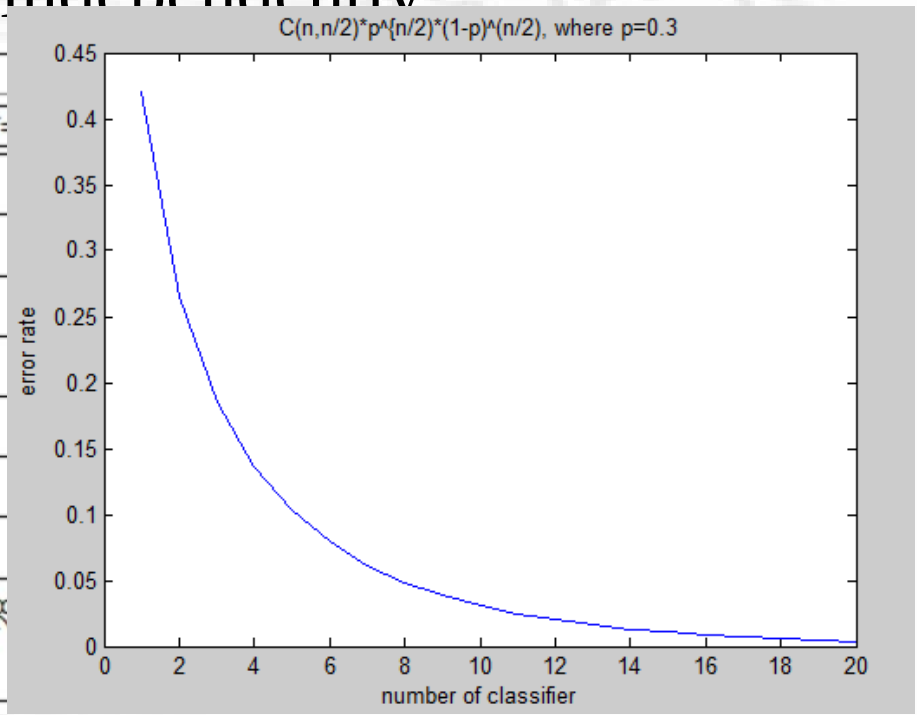
Ensemble Classifiers

- Combining simple classifiers by majority votes
- Famous ones: bagging and boosting
- Why it works:
  - Might reduce error (bias)
  - Reduce variance

Reduce Bias

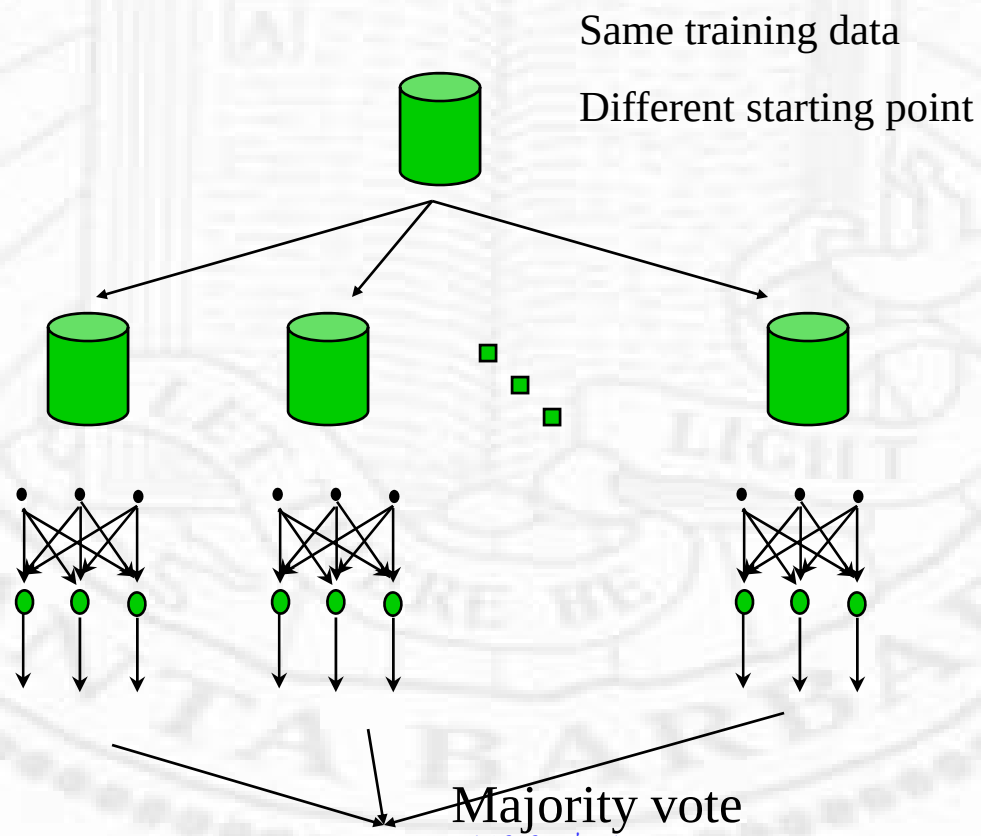
- If each classifier makes error, say 30%
- How likely it is for a committee of n classifiers to make mistake by majority rule
- Answer: Bonelli distribution
- Big IF: they must perform independently





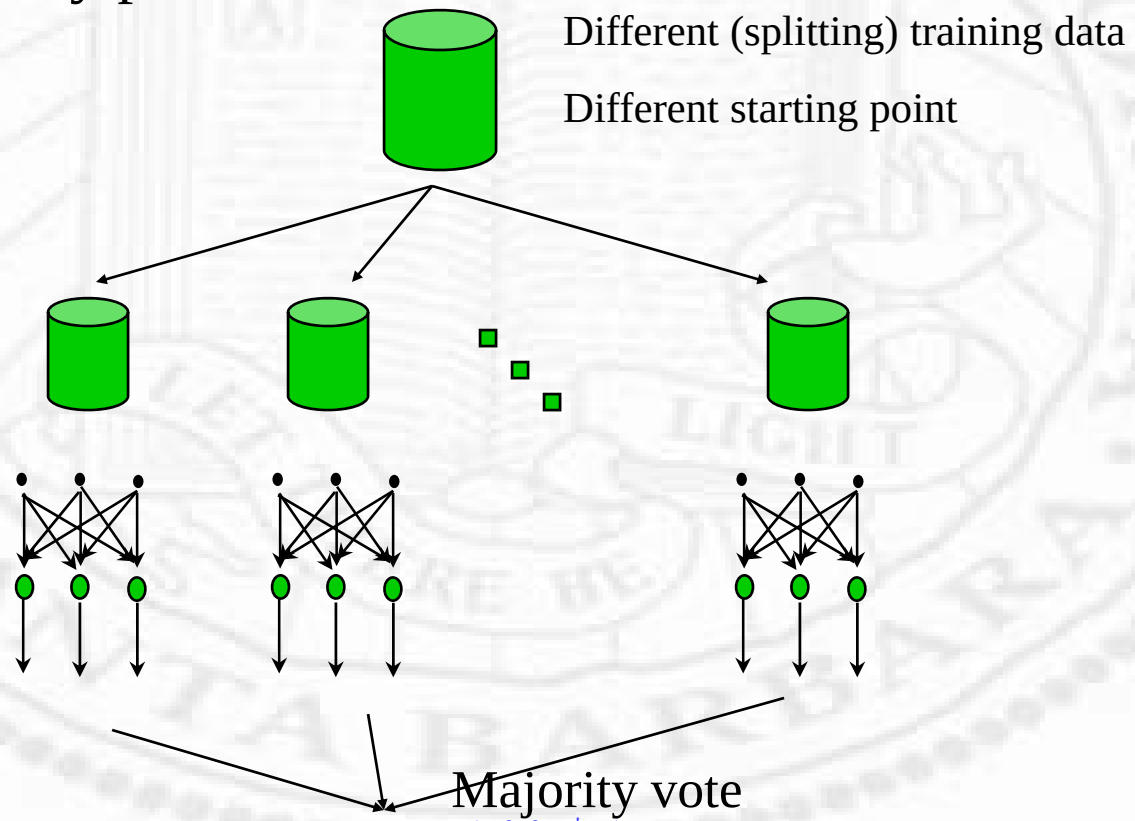
Improvement - Averaging

- Each machine reach a local minimum
- Majority vote



Improvement - Bagging

- Bootstrap AGGregation
- A simple "parallel processing" model using multiple component classifiers
- Help stability problem



Why Bagging Works?

- It can reduce both bias and variance
- Bias: not necessarily improve

$$E_D\left[(g(\mathbf{x};D)-F(\mathbf{x}))^2\right]=(E_D[g(\mathbf{x};D)-F(\mathbf{x})])^2+E_D\left[(g(\mathbf{x};D)-E_D[g(\mathbf{x};D)])^2\right]$$

$$E_D[g(\mathbf{x};D)-F(\mathbf{x})]=E_D[\frac{1}{n}\sum g_i(\mathbf{x};D)-F(\mathbf{x})]$$

$$=[\frac{1}{n}\sum E_D g_i(\mathbf{x};D)-\frac{1}{n}\sum F(\mathbf{x})]$$

$$=\frac{1}{n}[\sum E_D g_i(\mathbf{x};D)-\sum F(\mathbf{x})]=\frac{1}{n}\{\text{bias of a constituent}\}$$

$$=\text{average bias of a constituent}$$

Why Bagging Works?

- It can reduce both bias and variance
- Variance: reduce to 1/n – IF all constituents are independent

$$E_D\left[(g(\mathbf{x};D)-F(\mathbf{x}))^2\right]=(E_D[g(\mathbf{x};D)-F(\mathbf{x})])^2+E_D\left[(g(\mathbf{x};D)-E_D[g(\mathbf{x};D)])^2\right]$$

$$E_D\left[(g(\mathbf{x};D)-E_D[g(\mathbf{x};D)])^2\right]$$

$$=E_D[(\frac{1}{n}\sum g_i(\mathbf{x};D)-E_D\frac{1}{n}\sum g_i(\mathbf{x};D))^2]$$

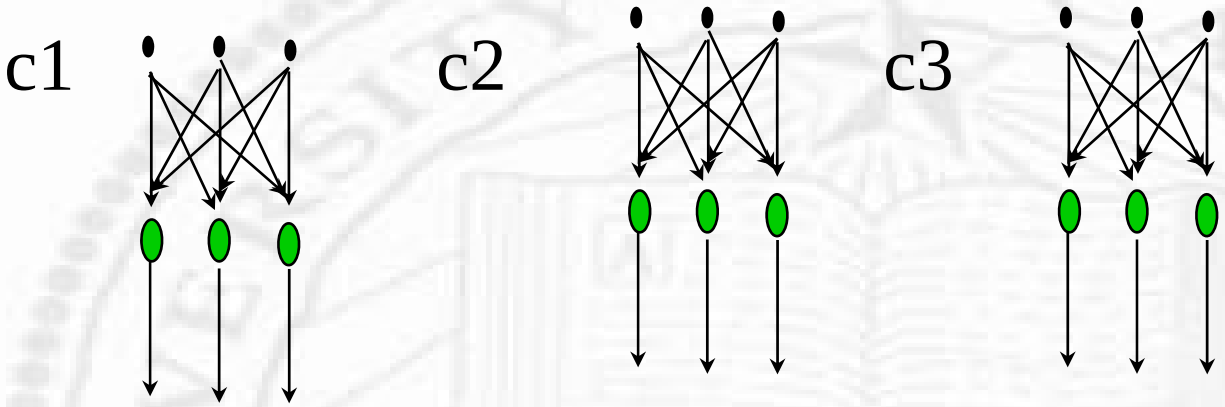
$$=\frac{1}{n}E_D[\frac{1}{n}(\sum(g_i(\mathbf{x};D)-E_D g_i(\mathbf{x};D))^2]$$

$$=\frac{1}{n}\{\text{average variance of all constituents}\}$$

Boosting by Filtering

- Bagging is *competition* model while boosting is a *collaborative* model
- Component Classifiers
  - should be introduced when needed
  - should then be trained on ambiguous samples
- Iterative refinement of results to reduce error and ambiguity

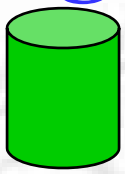
Boosting by Filtering (cont)



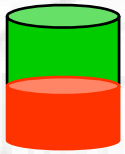
- If c1 and c2 agree, use that
- Otherwise, use c3

Boosting by Filtering (cont.)

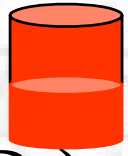
- D_1:



  - subset from D (without replacement) to train c1
- D_2:

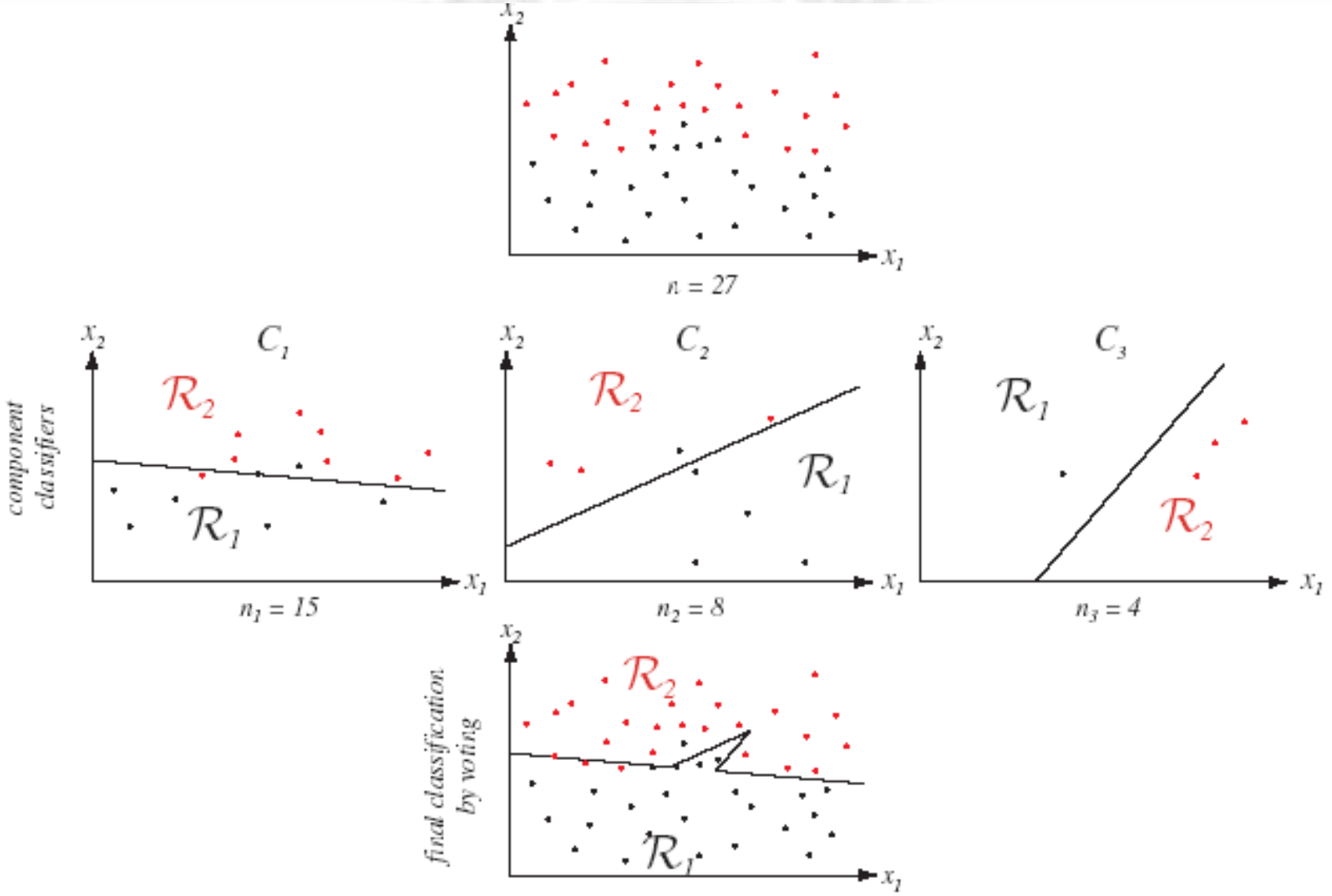


  - Head: samples from D-D_1, where c1 is wrong
  - Tail: samples from D-D_1 where c1 is correct
  - Half correct/half wrong for D_1, D_2 is learning what D_1 has difficulty with
- D_3:



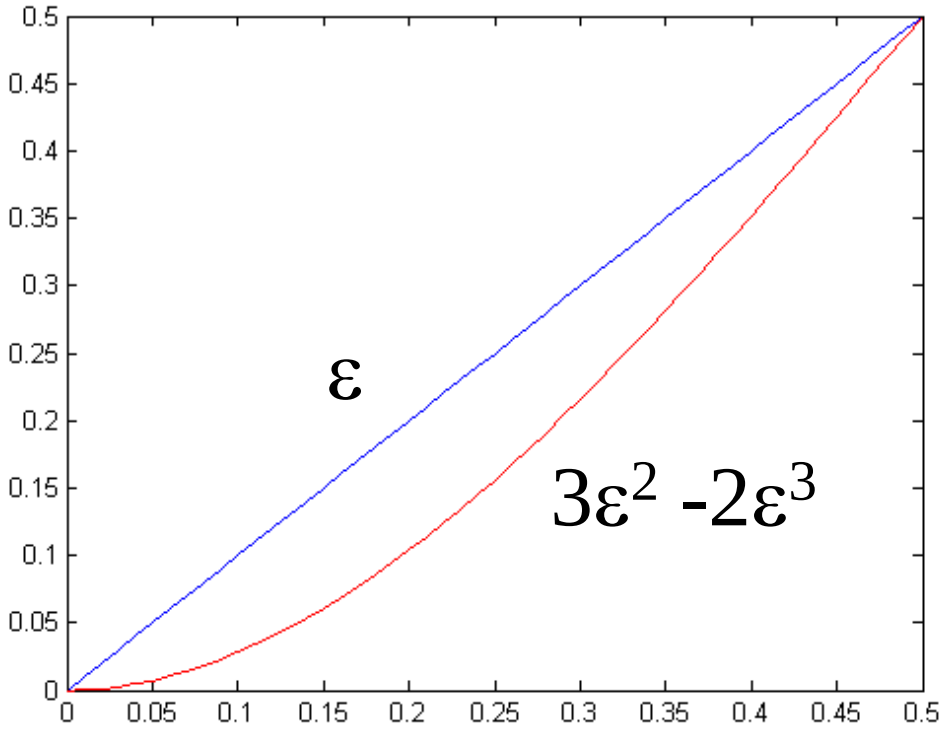
  - D-(D_1+D_2) where c1 and c2 disagree
  - D_3 is learning what the previous two cannot agree

Boosting by Filtering (cont.)



Boosting by Filtering (cont.)

- If each committee machine has an error rate of ε, then the combined machine has an error rate of $3\varepsilon^2 - 2\varepsilon^3$



AdaBoost –Adaptive Boosting

- Basic idea is very simple
  - Add more component classifiers if error is higher than preset threshold
  - Samples are weighted: if samples are accurately classified by combined component classifiers, the chance of being picked for the new classifier is reduced
  - Adaboost focuses on difficult patterns

Adaboost Algorithm

Initialization

$$D=\{(x^1,y^1),\cdots,(x^n,y^n)\},k_{\max},W_0(i)=\frac{1}{n},i=1,\cdots,n$$

Procedure

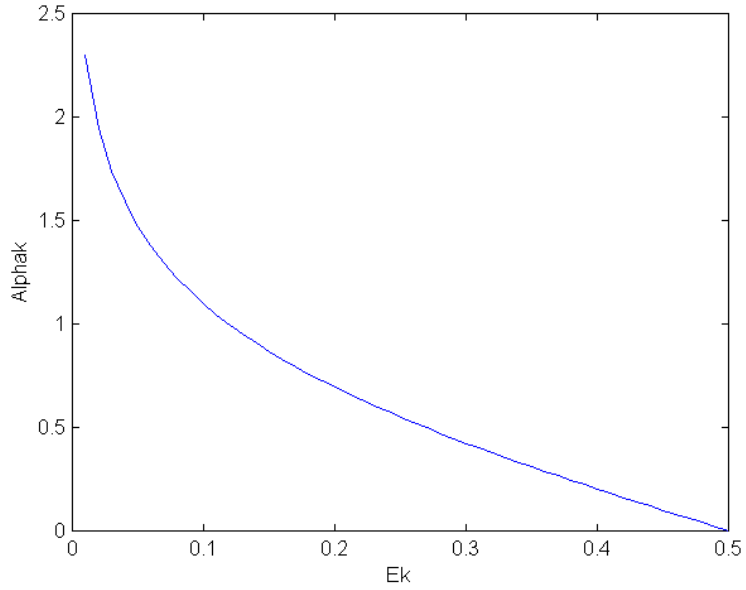
for $(k=1,\ k<k_{\max},k++)$

train weak clearner C_k using D sampled according to $W_k(i)$

$$E_k \leftarrow prob[h_k(x_i)\neq y_i]=\sum_{h_k(x_i)\neq y_i} D_t(i)$$ Error rate of C_k

$$\alpha_k \leftarrow \frac{1}{2}\ln[\frac{1-E_k}{E_k}]$$ Weighting of C_k

$$W_{k+1}(i)\leftarrow \frac{W_k(i)}{Z_k}\times\begin{cases} e^{-\alpha_k} & h_k(x_i)=y_i \\ e^{\alpha_k} & h_k(x_i)\neq y_i \end{cases}$$

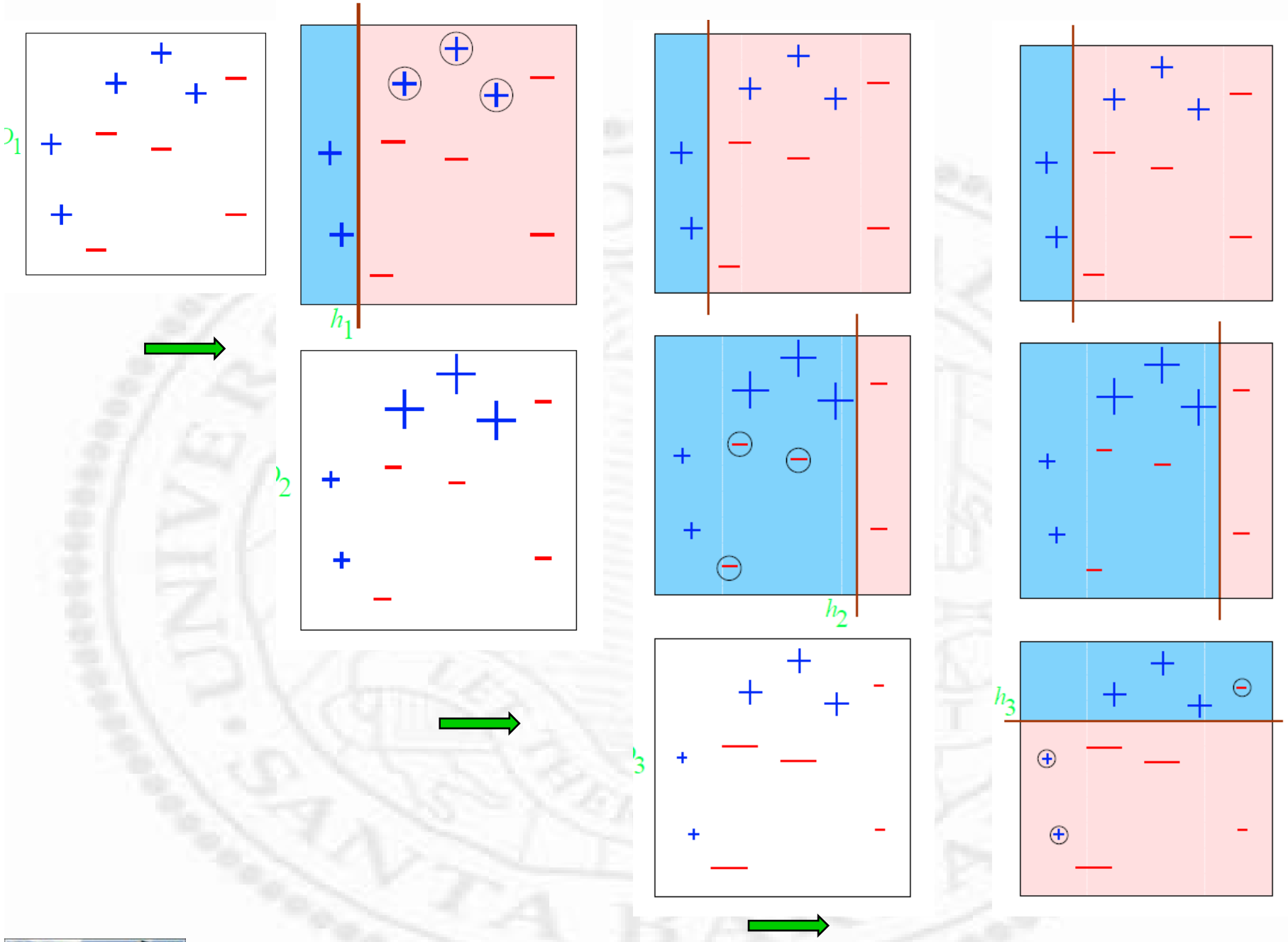


return

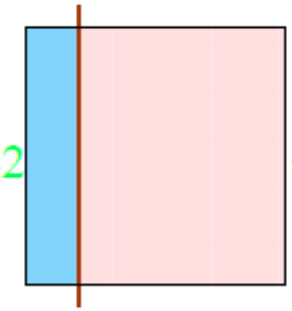
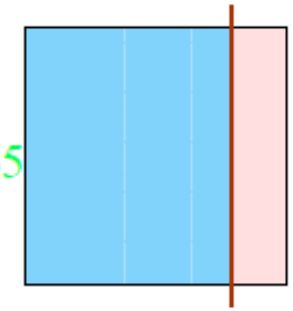
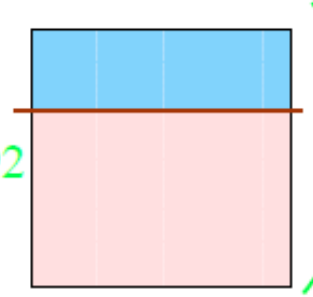
C_k and $\alpha_k, k=1,\cdots k_{\max}$

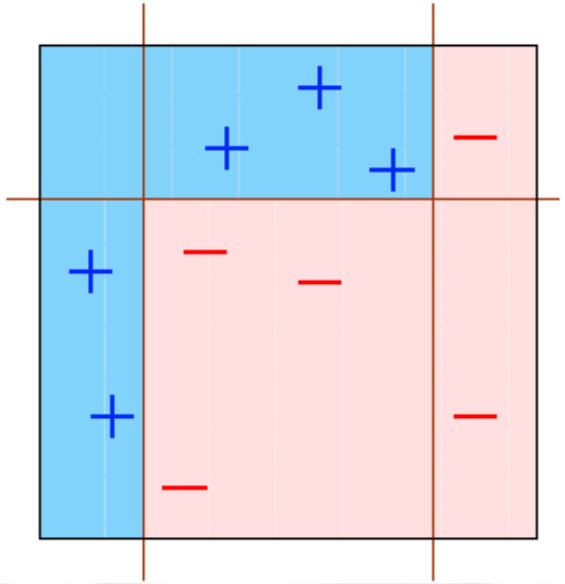
Final hypothesis

$$H(x)=sign(\sum_{k=1}^{k_{\max}}\alpha_k h_k(x))$$


D_1
h_1
D_2
h_2
D_3
h_3

H_{final}

$= \text{sign}\left(0.42 \right.$  $+ 0.65$  $+ 0.92$  $\left. \right)$

$=$ 

Comparison

Multilayer Perceptron

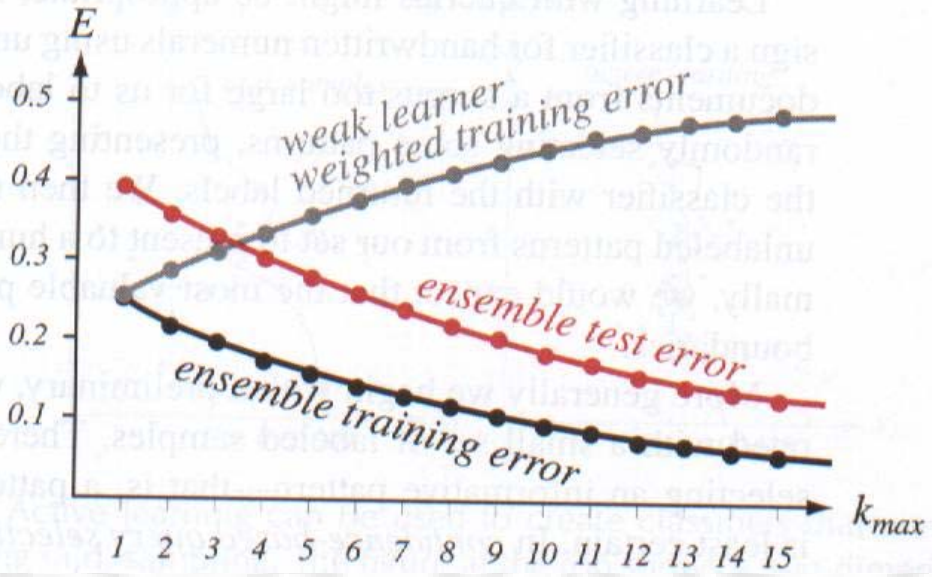
- Many neurons
- Train together
- Hard to train
- Same data set
- Require nonlinearity in thresholding
- Complicated decision boundary
- Overfitting likely

Boosting

- Many weak classifiers
- Trained individually
- Easy to train
- Fine-tuned data set
- Require different data sets
- Simple decision boundary
- Less susceptible to overfitting

Adaboost Algorithm (cont.)

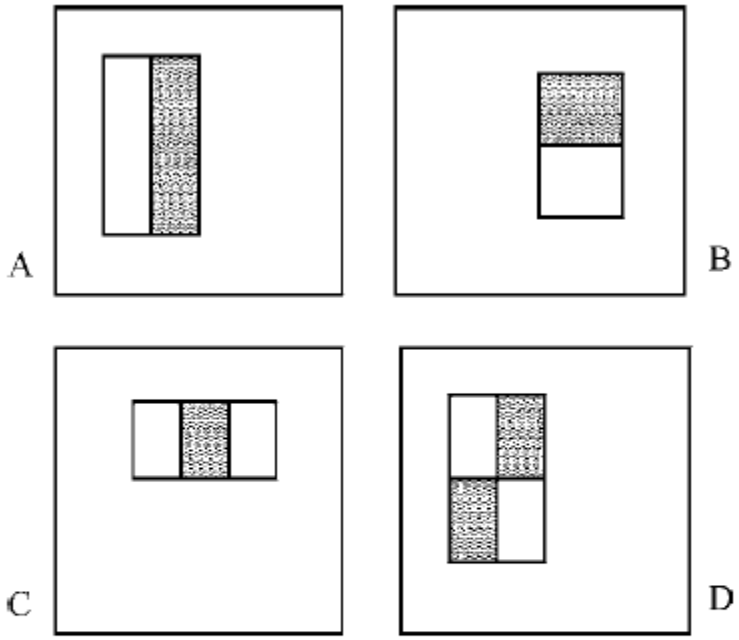
- As long as each individual weak learner has a better than chance performance, Adaboost can boost the performance arbitrarily well (Freund and Schapire, JCSS 1997)

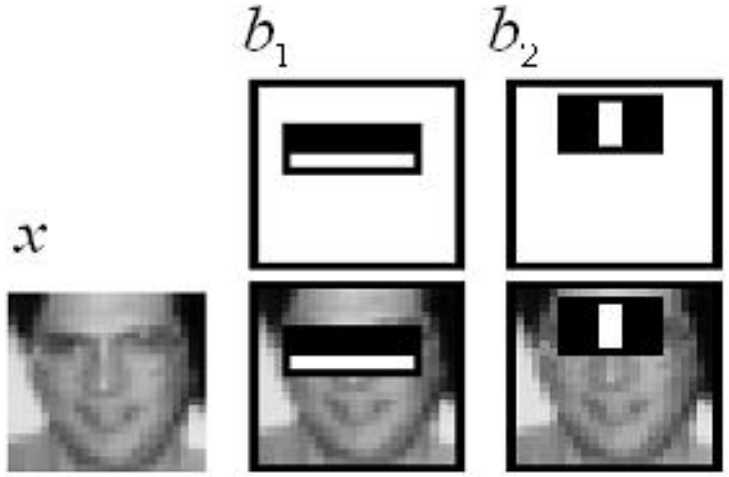


Applications

- Adaboost has been used successfully in many applications
- One famous one is to the use in face detection from images

Viola and Jones





$$h_k(x): \quad x \cdot b_k > \theta_k$$

Figure 1. Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

Fast Feature Computation

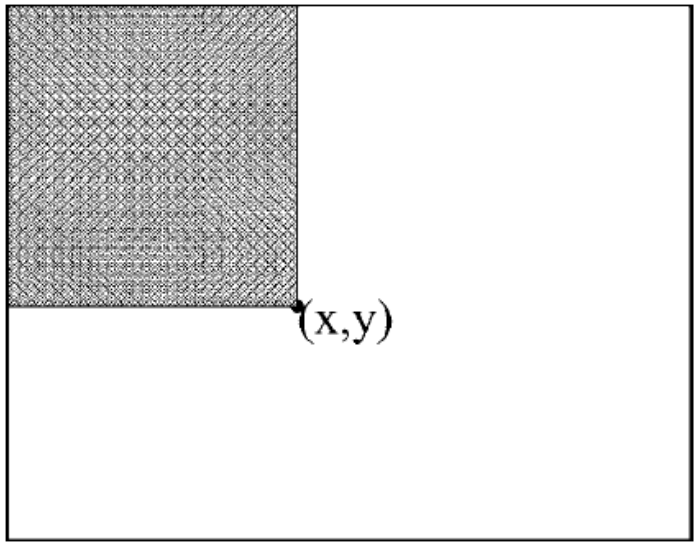


Figure 2. The value of the integral image at point (x, y) is the sum of all the pixels above and to the left.

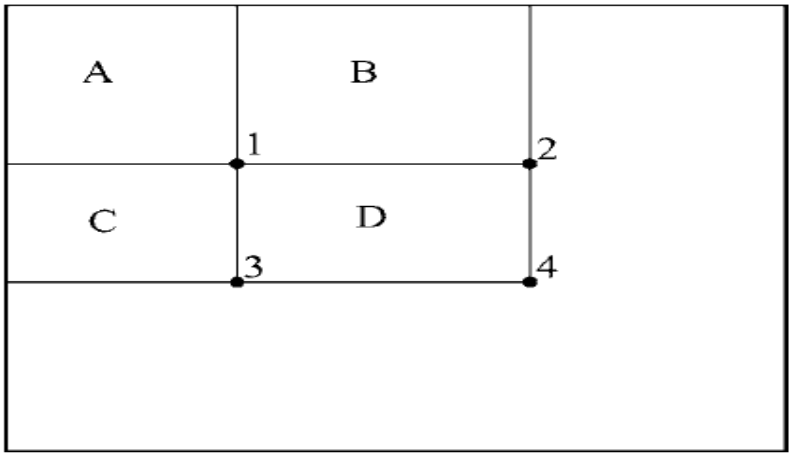


Figure 3. The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

Classifier

- Adaboost idea – greedy feature (classifier) selection
- Weak leaner (h) – select a single rectangular feature
  - f: feature
  - θ: threshold
  - p: polarity
  - X: 24x24 ixel sub window

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases}$$

Algorithm

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:

  1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  2. Select the best weak classifier with respect to the weighted error

     $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \left| h(x_i, f, p, \theta) - y_i \right| .$$

     See Section 3.1 for a discussion of an efficient implementation.
  3. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t, and θ_t are the minimizers of ϵ_t.
  4. Update the weights:

     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

Efficiency

- Use attention cascade

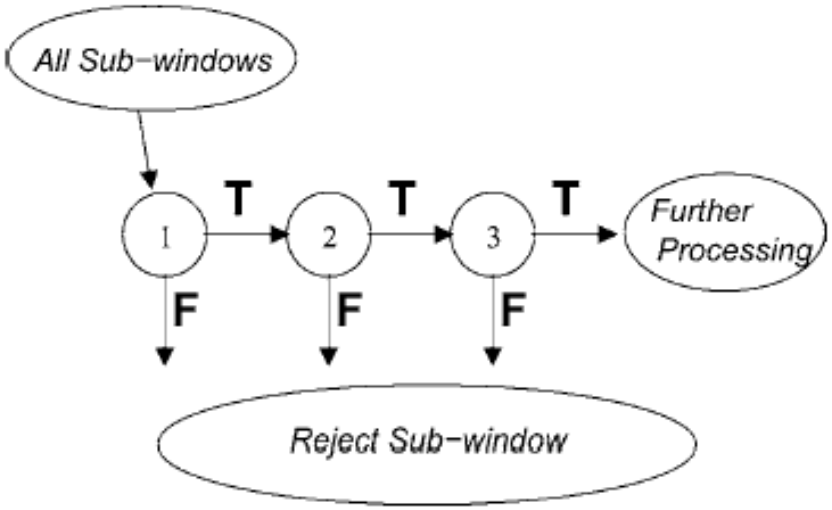
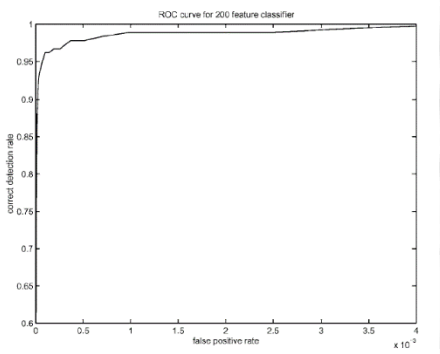
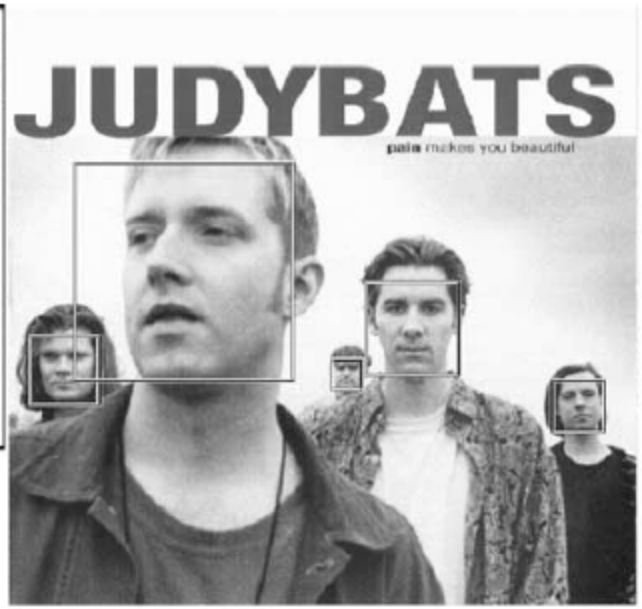
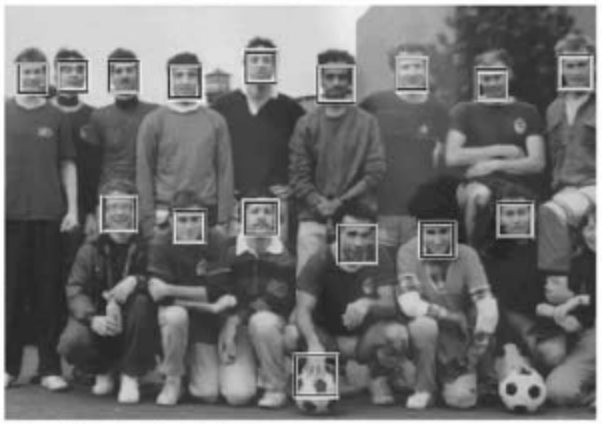


Figure 6. Schematic depiction of a the detection cascade. A series of classifiers are applied to every sub-window. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negatives but require additional computation. After several stages of processing the number of sub-windows have been reduced radically. Further processing can take any form such as additional stages of the cascade (as in our detection system) or an alternative detection system.

Results







University of California Santa Barbara

Results

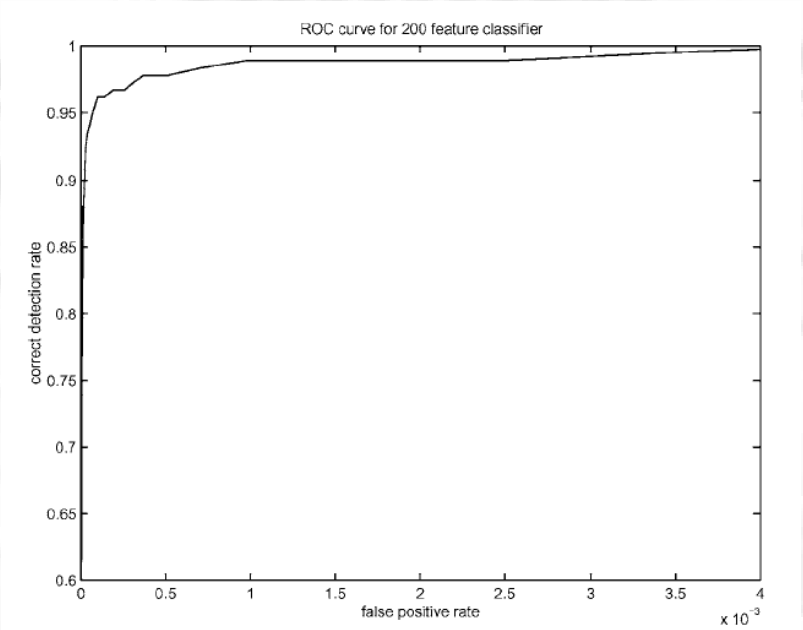


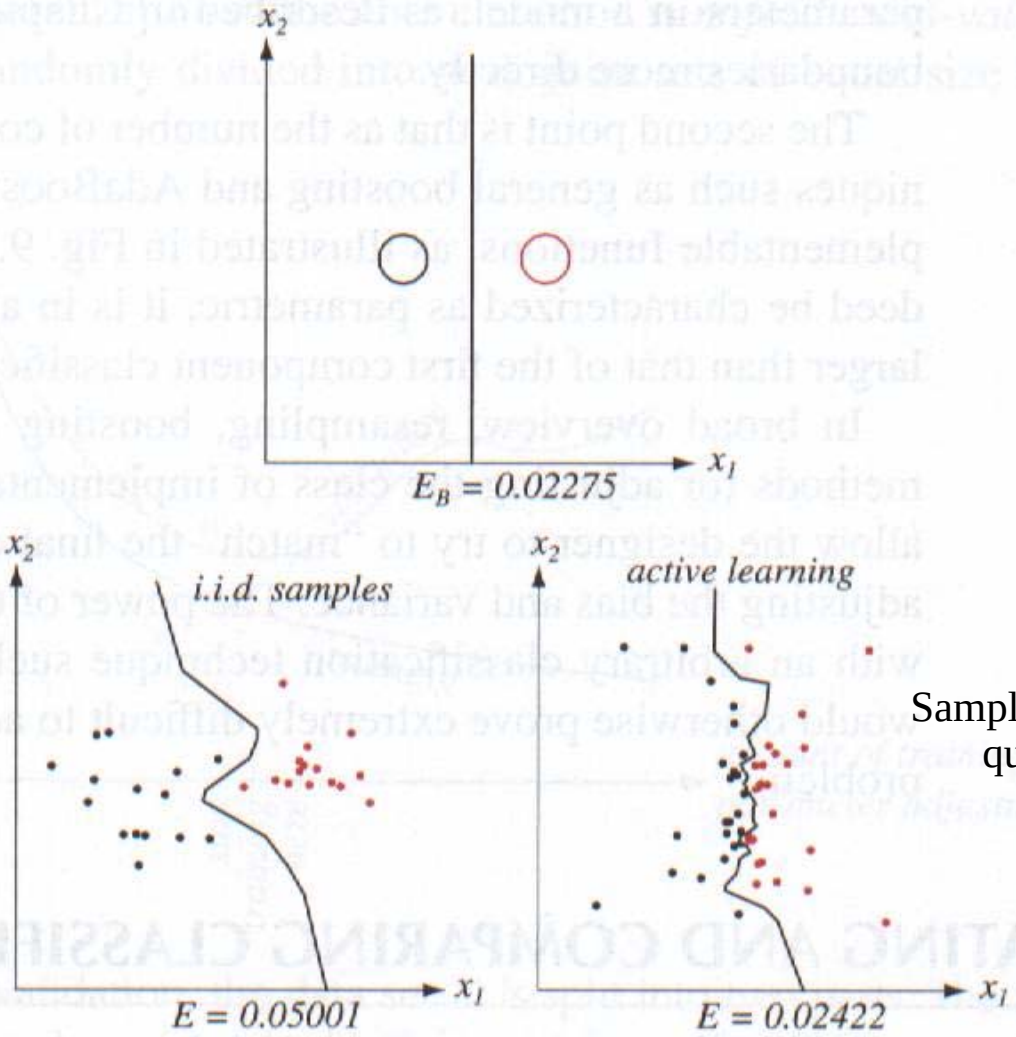
Table 3. Detection rates for various numbers of false positives on the MIT + CMU test set containing 130 images and 507 faces.

	False detections							
Detector	10	31	50	65	78	95	167	422
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%	94.1%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2%	93.7%	–
Rowley-Baluja-Kanade	83.2%	86.0%	–	–	–	89.2%	90.1%	89.9%
Schneiderman-Kanade	–	–	–	94.4%	–	–	–	–
Roth-Yang-Ahuja	–	–	–	–	(94.8%)	–	–	–

Learning with Queries

- Given a weak classifier or several weak component classifiers
- Find out where ambiguity is
  - Where weak classifier gives high reading for the top two discrimination functions (e.g., in a linear machine)
  - Where component classifiers yield the greatest disagreement
- Train the classifiers with those ambiguous samples

Learning with Queries (cont.)



Samples generated with queries