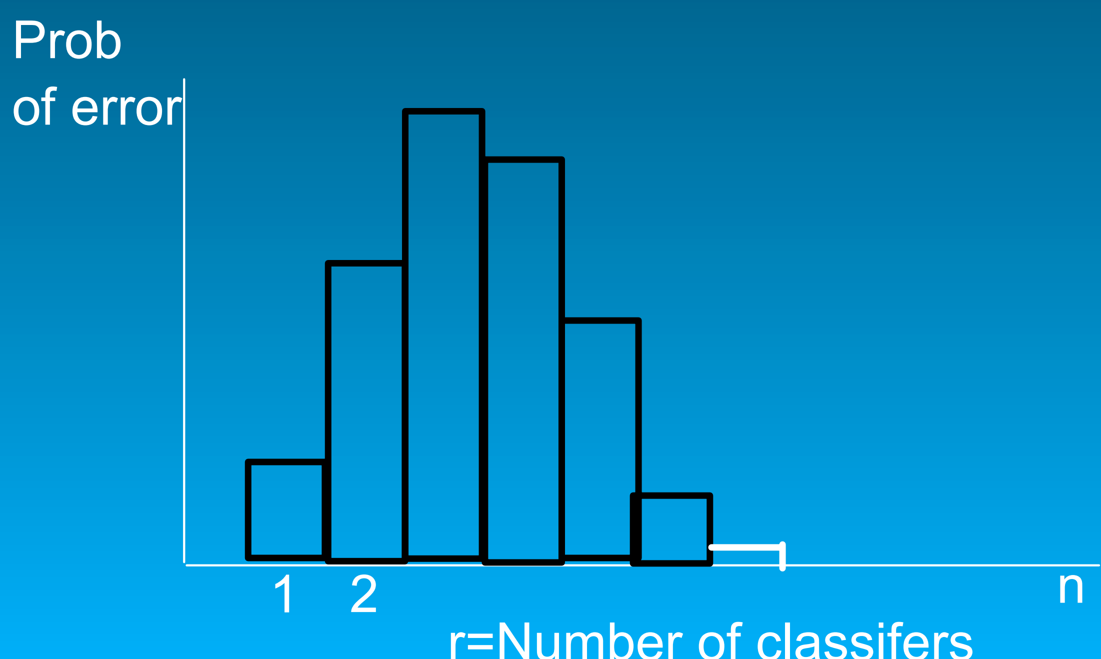
Combining Classifiers

- Generic methods of generating and combining multiple classifiers
  - Bagging
  - Boosting
- References:
  - Duda, Hart & Stork, pg 475-480.
  - Hastie, Tibsharini, Friedman, pg 246-256 and Chapter 10.
  - http://www.boosting.org/
    - Bulletin Board
      - "Is there a book available on boosting?"
- Stacking
  - "meta-learn" which classifier does well where
- Error-correcting codes
  - going from binary to multi-class problems

Why Combine Classifiers?

- Combine several classifiers to produce a more accurate single classifier
- If C_2 and C_3 are correct where C_1 is wrong, etc, majority vote will do better than each C_i individually
- Suppose
  - each C_i has error rate $p<0.5$
  - errors of different C_i are uncorrelated
- Then Pr(r out of n classifiers are wrong) = ${}_nB_r\, p^r(1-p)^{n-r}$



Pr(majority of n classifiers are wrong)
= right-half of binomial distribution
is small if:
n is large
p is small

Bagging

- "Bootstrap aggregation"
- Bootstrap estimation - generate data set by randomly selecting from training set
  - with replacement (some points may repeat)
  - repeat *B* times
  - use as estimate the average of individual estimates
- Bagging
  - generate *B* equal size training sets
  - each training set
    - is drawn randomly, with replacement, from the data
    - is used to generate a different component classifier f_i
      - usually using same algorithm (e.g. decision tree)
  - final classifier decides by voting among component classifiers
- Leo Breiman, 1996.

Bagging (contd)

- Suppose there are k classes
  - Each $f_i(x)$ predicts 1 of the classes
  - Equivalently, $f_i(x)$ = (0, 0, ..., 0, 1, 0, ..., 0)
- Define $f_{bag}(x) = (1/B)\Sigma_{i=1}^{B} f_i(x)$

  $= (p_1(x), ..., p_k(x))$,

  $p_j(x)$ = proportion of f_i predicting class j at x
- Bagged prediction is
  - arg max$_k$ $f_{bag}(x)$
- Reduces variance
  - always (provable) for squared-error
  - not always for classification (0/1 loss)
  - In practice usually most effective if classifiers are "unstable" - depend sensitively on training points.
- However may lose interpretability
  - a bagged decision tree is not a single decision tree

Boosting

- Generate the component classifiers so that each does well where the previous ones do badly
  - Train classifier C_1 using (some part of) the training data
  - Train classifier C_2 so that it performs well on points where C_1 performs badly
  - Train classifer C_3 to perform well on data classified badly by C_1 and C_2, etc.
- Overall classifier C classifies by weighted voting among the component classifiers C_i
- The same algorithm is used to generate each C_i - only the data used for training changes

AdaBoost

- "Adaptive Boosting"
- Give each training point (x_i, $y_i = \pm 1$) in D a weight w_i (initialized uniformly)
- Repeat:
  - Draw a training set D_m at random from D according to the weights w_i
  - Generate classifier C_m using training set D_m
  - Measure error of C_m on D
    - Increase weights of misclassified training points
    - Decrease weights of correctly classified points
- Overall classification is determined by
  - $C_{boost}(x) = \mathrm{Sign}(\Sigma_m\, \alpha_m C_m(x))$, where
  - α_m measures the "quality" of C_m
- Terminate when $C_{boost}(x)$ has low error

AdaBoost (Details)

- Initialize weights uniformly: $w_i^1 = 1/N$ (N=training set size)
- Repeat for m=1,2, ..., M
  - Draw random training set D_m from D according to weights w_i^m
  - Train classifier C_m using training set D_m
  - Compute $err_m = Pr_{i\sim Dm} [C_m(x_i) \neq y_i]$
    - error rate of C_m on (weighted) training points
  - Compute $\alpha_m = 0.5 \log((1-err_m)/err_m)$
    - $\alpha_m = 0$ when $err_m = 0.5$
    - $\alpha_m \to \infty$ as $err_m \to 0$
  - $w_i^{m*} = w_i^m \exp(\alpha_m) = w_i^m \sqrt{(1-err_m)/err_m}$ if x_i is incorrectly classified
    $w_i^m \exp(-\alpha_m) = w_i^m \sqrt{err_m/(1-err_m)}$ if x_i is correctly classified
  - $w_i^{m+1} = w_i^{m*}/Z_m$
    - $Z_m = \Sigma_i w_i^{m*}$ is a normalization factor so that $\Sigma_i w_i^{m+1} = 1$
- Overall classification is determined by
  - $C_{boost}(x) = \mathrm{Sign}(\Sigma_m \alpha_m C_m(x))$

Theory

- If:
  - each component classifier C_m is a "weak learner"
    - performs better than random chance (err_m<0.5)
- Then:
  - the TRAINING SET ERROR of C_{boost} can be made arbitrarily small as M (the number of boosting rounds) ->∞
- Proof (see Later)
- Probabilistic bounds on the TEST SET ERROR can be obtained as a function of training set error, sample size, number of boosting rounds, and "complexity" of the classifiers C_m
- If Bayes Risk is high, it may become impossible to continually find C_m which perform better than chance.
- "In theory theory and practice are the same, but in practice they are different"

Practice

- Use an independent test set to determine stopping point
- Boosting performs very well in practice
  - Fast
  - Boosting decision "stumps" is competitive with decision trees
  - Test set error may continue to fall even after training set error=0
  - Does not (usually) overfit
  - Sometimes vulnerable to outliers/noise
  - Result may be difficult to interpret
- "AdaBoost with trees is the best off-the-shelf classifier in the world" - Breiman, 1996.

History

- Robert Schapire, 1989
  - Weak classifier could be boosted
- Yoav Freund, 1995
  - Boost by combining many weak classifiers
  - Required bound on error rate of weak classifier
- Freund & Schapire, 1996
  - AdaBoost - adapts weights based on error rate of weak classifier
- Many extensions since then
  - Boosting Decision Trees, Naive Bayes, ...
  - More robust to noise
  - Improving interpretability of boosted classifier
  - Incorporating prior knowledge
  - Extending to multi-class case
  - "Balancing between Boosting and Bagging using Bumping"
  - .....

Proof

- Claim: If $err_m<0.5$ for all m, then Training Set Error of C_{boost} ->0 as M->∞
- Note: $y_i C_m(x_i) = 1$ if x_i is correctly classified by C_m
  $= -1$ if x_i is incorrectly classified by C_m,
  similarly for $C_{boost}(x) = \text{sign}(\Sigma_m \alpha_m C_m(x))$
- Training Set Error of classifer $C_{boost}(x)$ is
  $err_{boost} = |\{i: C_{boost}(x_i) \neq y_i\}| / N$
- $C_{boost}(x_i) \neq y_i$ if and only if $y_i \Sigma_m \alpha_m C_m(x)<0$
  if and only if $-y_i \Sigma_m \alpha_m C_m(x))>0$
- Hence $C_{boost}(x_i) \neq y_i \Rightarrow \exp(-y_i \Sigma_m \alpha_m C_m(x))>1$
  so $err_{boost} < [\Sigma_i \exp(-y_i \Sigma_m \alpha_m C_m(x))]/N$
- By definition, $w_i^{m+1} = w_i^m \exp(-y_i \alpha_m C_m(x)) / Z_m$
- So $\exp(-y_i \alpha_m C_m(x)) = Z_m w_i^{m+1} / w_i^m$
- Now insert the "sum" into the exponential:

$$\begin{aligned}\exp(-y_i \Sigma_m \alpha_m C_m(x)) &= \Pi_m \exp(-y_i \alpha_m C_m(x)) \\ &= \Pi_m Z_m w_i^{m+1} / w_i^m \\ &= w_i^{M+1} / w_i^1 \, \Pi_m Z_m \\ &= N w_i^{M+1} \, \Pi_m Z_m\end{aligned}$$

Proof *(continued)*

- Thus $[\Sigma_i \exp(-y_i \Sigma_m \alpha_m C_m(x))]/N = \Sigma_i w_i^{M+1} \Pi_m Z_m$
  $= \Pi_m Z_m$
  because $\Sigma_i w_i^{M+1}=1$ (having been normalized by Z_M)
- Nothing has been said so far about the choice of α_m
- Set $\alpha_m = 0.5 \log((1-err_m)/err_m)$
- Then $w_i^{m*} = w_i^m \sqrt{(1-err_m)/err_m}$ if x_i is incorrectly classified
  $w_i^m \sqrt{err_m/(1-err_m)}$ if x_i is correctly classified
- To normalize, set $Z_m = \Sigma_i w_i^{m*}$
  $= \Sigma_i w_i^m[err_m(\sqrt{(1-err_m)/err_m}) + (1-err_m)\sqrt{err_m/(1-err_m)}]$
  $= \Sigma_i w_i^m[\sqrt{err_m(1-err_m)} + \sqrt{err_m(1-err_m)}]$
  $= 2\sqrt{err_m(1-err_m)}$
  because $\Sigma_i w_i^m=1$
- So $err_{boost} < [\Sigma_i \exp(-y_i \Sigma_m \alpha_m C_m(x))]/N = \Pi_m Z_m = \Pi_m 2\sqrt{err_m(1-err_m)}$
  NOTE: D, H & S, pg 479, says $err_{boost} = \Pi_m 2\sqrt{err_m(1-err_m)}$

Proof *(continued)*

- Let $\gamma_m = 0.5 - err_m > 0$ for all m
  - γ_m is the "edge" of C_m over random guessing
- Then $2\sqrt{err_m(1-err_m)} = 2\sqrt{(0.5-\gamma_m)(0.5+\gamma_m)}$
  $= \sqrt{1-4\gamma_m^2}$
- So $err_{boost} < \Pi_m \sqrt{1-4\gamma_m^2}$
  $< \Pi_m(1-2\gamma_m^2)$ since $(1-x)^{0.5} = 1-0.5x-\ldots.$
  $< \Pi_m \exp(-2\gamma_m^2)$ since $1+x < \exp(x)$
  $= \exp(-2\Sigma_m\gamma_m^2)$
- If:
  - $\gamma_m > \gamma > 0$ for all m
- Then
  - $err_{boost} < \exp(-2\Sigma_m\gamma^2)$
    $= \exp(-2M\gamma^2)$

which tends to zero exponentially fast as $M\text{->}\infty$

Why Boosting Works

- "The success of boosting is really not very mysterious." - Jerome Friedman, 2000.
- Additive models:
  - $f(x) = \Sigma_m \alpha_m b(x;\theta_m)$
    - Classify using Sign(f(x))
  - b = "basis" function parametrized by θ
  - α_m are weights
- Examples:
  - neural networks
    - b = activation function, θ = input-to-hidden weights
  - support vector machines
    - b = kernel function, appropriately parametrized
  - boosting
    - b = weak classifier, appropriately parametrized

Fitting Additive Models

- To fit $f(x) = \Sigma_m \alpha_m b(x;\theta_m)$, usually α_m, θ_m are found by minimizing a loss function (e.g. squared error) over the training set
- Forward Stagewise fitting:
  - Add new basis functions to the expansion one-by-one
  - Do not modify previous terms
- Algorithm:
  - $f_0(x) = 0$
  - For m=1 to M:
    - Find α_m, θ_m by $\min_{\alpha,\theta} \Sigma_i L(y_i, f_{m-1}(x)+\alpha b(x_i;\theta))$
    - Set $f_m(x) = f_{m-1}(x) + \alpha_m b(x;\theta_m)$
- AdaBoost is Forward Stagewise fitting applied to the weak classifier with an EXPONENTIAL loss function

AdaBoost (Derivation)

- $L(y,f(x)) = \exp(-yf(x))$ exponential loss
- $\alpha_m, C_m = \arg\min_{\alpha,c} \Sigma_i \exp(-y_i(f_{m-1}(x_i)+\alpha C(x_i)))$

  $= \arg\min_{\alpha,c} \Sigma_i \exp(-y_i(f_{m-1}(x_i)))\exp(-\alpha y_i C(x_i))$

  $= \arg\min_{\alpha,c} \Sigma_i w_i^m \exp(-\alpha y_i C(x_i))$

  where $w_i^m = \exp(-y_i(f_{m-1}(x_i)))$

  w_i^m depends on neither α nor C.
- Note: $\Sigma_i w_i^m \exp(-\alpha y_i C(x_i))$

  $= e^{-\alpha}\Sigma_{yi=C(xi)} w_i^m + e^{\alpha}\Sigma_{yi\neq C(xi)} w_i^m$

  $= e^{-\alpha}\Sigma_i w_i^m + (e^{\alpha}-e^{-\alpha})\Sigma_i w_i^m \mathrm{Ind}(y_i \neq C(x_i))$
- For $\alpha>0$, pick $C_m = \arg\min_c \Sigma_i w_i^m \mathrm{Ind}(y_i \neq C(x_i))$

  $= \arg\min_c err_m$

AdaBoost (Derivation) *(continued)*

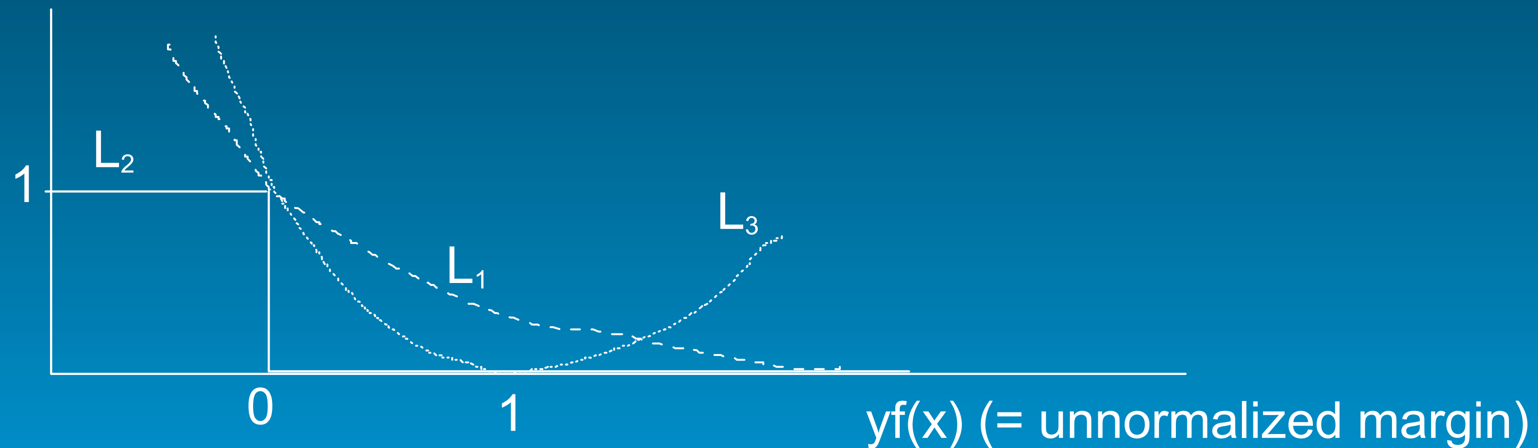
- **Substitute back:**
  - yields $e^{-\alpha}\Sigma_i w_i^m + (e^{\alpha} - e^{-\alpha}) err_m$
    - a function of α only
  - $\arg\min_{\alpha} e^{-\alpha}\Sigma_i w_i^m + (e^{\alpha} - e^{-\alpha}) err_m$ can be found
    - differentiate, etc - Exercise!
  - giving $\alpha_m = 0.5\log((1-err_m)/err_m$
- **The model update is:** $f_m(x) = f_{m-1}(x) + \alpha_m C_m(x_i)$

$$
\begin{aligned}
w_i^{m+1} &= \exp(-y_i(f_m(x_i))) \\
&= \exp(-y_i(f_{m-1}(x_i) + \alpha_m C_m(x_i))) \\
&= \exp(-y_i(f_{m-1}(x_i)))\exp(-y_i\alpha_m C_m(x_i)) \\
&= w_i^m \exp(-\alpha_m y_i C_m(x_i))
\end{aligned}
$$

deriving the weight update rule.

Exponential Loss

- $L_1(y,f(x)) = \exp(-yf(x))$ exponential loss
- $L_2(y,f(x)) = \mathrm{Ind}(yf(x)<0)$ 0/1 loss
- $L_3(y,f(x)) = (y-f(x))^2$ squared error



Exponential loss puts heavy weight on examples with large negative margin
These are difficult, atypical, training points - boosting is sensitive to outliers

Boosting and SVMs

- The margin of (x_i, y_i) is $(y_i \Sigma_m \alpha_m C_m(x_i)) / \Sigma_m |\alpha_m|$
  $= y_i(\alpha \bullet C(x_i)) / \|\alpha\|$
  - lies between -1 and 1
  - >0 if and only if x_i is classified correctly
- Large margins on the training set yield better bounds on generalization error
- It can be argued that boosting attempts to (approximately) maximize the minimum margin
  - $\max_\alpha \min_i y_i(\alpha \bullet C(x_i)) / \|\alpha\|$
  - same expression as SVM, but 1-norm instead of 2-norm

Stacking

- Stacking = "stacked generalization"
- Usually used to combine models l_1, ..., l_r of different types
  - e.g. l_1=neural network,
  - l_2=decision tree,
  - l_3=Naive Bayes,
  - ...
- Use a "meta-learner" L to learn which classifier is best where
- Let x be an instance for the component learners
- Training instance for L is of the form
  - $(l_1(x), \ldots, l_r(x))$,
    - $l_i(x)$ = class predicted by classifier l_i

  OR
  - $(l_{11}(x), \ldots, l_{1k}(x), \ldots, l_{r1}(x) \ldots, l_{rk}(x))$,
    - $l_{ij}(x)$ = probability x is in class j according to classifier l_i

Stacking *(continued)*

- What should class label for L be?
  - actual label from data
    - may prefer classifiers that overfit
  - use a "hold-out" data set which is not used to train the $l_1, \ldots, l_r$
    - wastes data
  - use cross-validation
    - when x occurs in the test set, use it as a training instance for L
    - computationally expensive
- Use simple linear models for L
- David Wolpert, 1992.

Error-correcting Codes

- Using binary classifiers to predict multi-class problem
- Generate one binary classifier C_i for each class vs every other class

class	C_1	C_2	C_3	C_4
a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

class	C_1	C_2	C_3	C_4	C_5	C_6	C_7
a	1	1	1	1	1	1	1
b	0	0	0	0	1	1	1
c	0	0	1	1	0	0	1
d	0	1	0	1	0	1	0

- Each binary classifier C_i predicts the i^{th} bit
  - LHS: Predictions like "1 0 1 0" cannot be "decoded"
  - RHS: Predictions like "1 0 1 1 1 1 1" are class "a" (C_2 made a mistake)

Hamming Distance

- Hamming distance H between codewords = number of single-bit corrections needed to convert one into the other
  - H(1000,0100) = 2
  - H(1111111,0000111) = 4
- (d-1)/2 single-bit errors can be corrected if d=minumum Hamming distance between any pair of code-words
  - LHS: d=2
    - No error-correction
  - RHS: d=4
    - Corrects all single-bit errors
- Tom Dietterich and Ghulum Bakiri, 1995.