Ensemble Methods

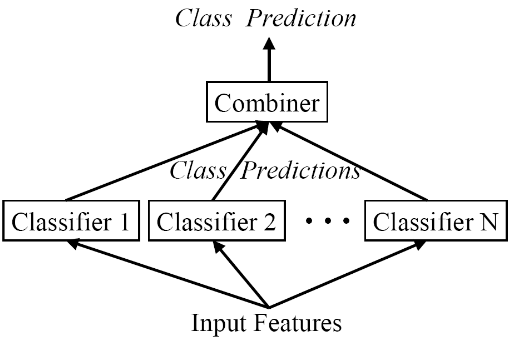
Mingmin Chi

SCS Fudan University, Shanghai, China

Outline

What's an ensemble

Ensemble methods are learning algorithms that construct a set of (base) classifiers and then classify new data points by taking a vote of their predictions



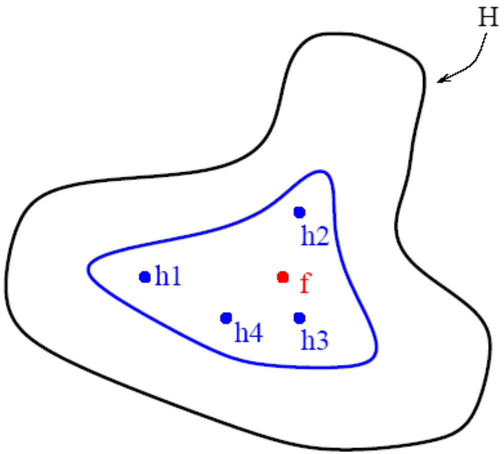
Hansen and Salamon, 1990

If we can assume classifiers are random in predictions and accuracy $>$ 50%, can push accuracy arbitrarily high by combining more classifiers

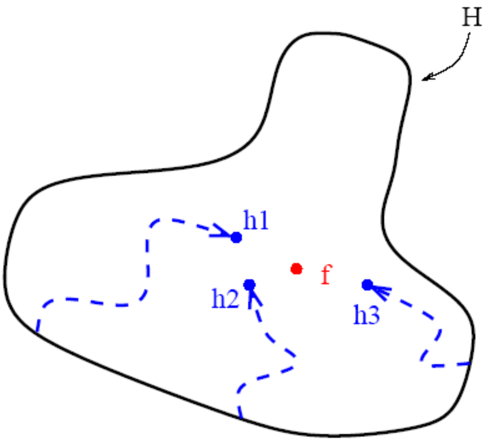
Key assumption: classifiers are independent in their predictions

- not a very reasonable assumption
- more realistic: for data points where classifiers predict with $>$ 50% accuracy, can push accuracy arbitrarily high (some data points just too hard)

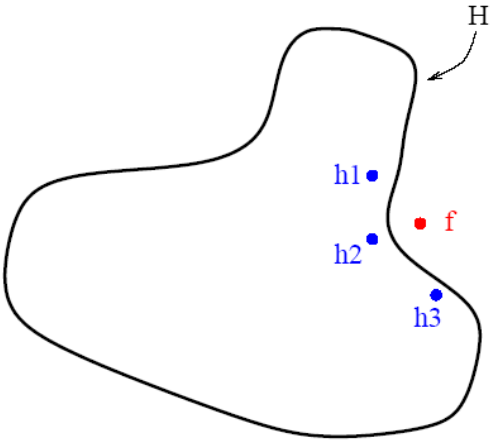
Statistical Reason



Computational Reason



Representational Reason



Variance and Bias

- A learning algorithm that suffers from the statistical problem is said to have high “variance"
- An algorithm that exhibits the computational problem is sometimes described has having “computational variance"
- An learning algorithm that suffers from the representational problem is said to have high “bias"

Hence, ensemble methods can reduce both the bias and the variance of learning algorithms

Theoretical Proofs

Hansen and Salamon (1990) proved that

- if the average error rate for an example is less than 50%
- the component classifiers in the ensemble are independent in the production of their errors

the predicted error for that example can be reduced to zero as the number of classifiers combined goes to infinity

Theoretical Proofs

Hansen and Salamon (1990) proved that

- if the average error rate for an example is less than 50%
- the component classifiers in the ensemble are independent in the production of their errors

the predicted error for that example can be reduced to zero as the number of classifiers combined goes to infinity

However, such assumption rarely hold in practice

Theoretical Proofs

Krogh and Vedelsby, 1995 proved that the ensemble error can be divided into

- a term measuring the average generalization error of each individual classifier and
- a term measuring disagreement among the classifiers

It can show that the accuracy of an ensemble is mathematically related:

$$\hat{E} = \bar{E} - D$$

where $\hat{E}$ is the error of the entire ensemble, $\bar{E}$ is the average error of the component classifiers, D is a term measuring the diversity of the components

Theoretical Proofs

Krogh and Vedelsby, 1995 proved that the ensemble error can be divided into

- a term measuring the average generalization error of each individual classifier and
- a term measuring disagreement among the classifiers

It can show that the accuracy of an ensemble is mathematically related:

$$\hat{E} = \bar{E} - D$$

where $\hat{E}$ is the error of the entire ensemble, $\bar{E}$ is the average error of the component classifiers, D is a term measuring the diversity of the components

Effective ensembles have accurate and diverse components

Enumerating the Hypotheses

In a Bayesian probabilistic setting,

- each hypothesis h defines a conditional probability distribution

$$h(\mathbf{x}) = P(f(\mathbf{x}) = y|\mathbf{x}, h)$$

- Given a new data point $\mathbf{x}$ and a training set $\mathbf{X}_l$, the problem of predicting the value of $f(\mathbf{x})$ can be viewed as the problem of computing $P(f(\mathbf{x}) = y|\mathbf{X}_l, \mathbf{x})$

- We can rewrite this as weighted sum over all hypotheses in $\mathcal{H}$:

$$P(f(\mathbf{x}) = y|\mathbf{X}_l, \mathbf{x}) = \sum_{h \in \mathcal{H}} h(\mathbf{x}) P(h|\mathbf{X}_l)$$

- We can view this as an ensemble method in which the ensemble consists of all of the hypotheses in $\mathcal{H}$, each weighted by its posterior probability $P(h|\mathbf{X}_l)$

Enumerating the Hypotheses

By Bayes rule, the posterior probability is proportional to the likelihood of the training data times the prior probability of *h*:

$$P(h|\mathbf{X}_l) \propto P(\mathbf{X}_l)P(h)$$

In some learning problems, it is possible to completely enumerate each $h \in \mathcal{H}$, compute $P(\mathbf{X}_l|h)$ and P(h), and evaluate the Bayesian “committee".

If the true function *f* is drawn from $\mathcal{H}$ according to $P(h)$, then the Bayesian voting scheme is optimal

- When the training set is small, many hypotheses *h* will have significantly large posterior probabilities, and the voting process can average these to “marginalize away" the remaining uncertainty about *f*
- When the training set is large, typically only one hypothesis has substantial posterior probability and the “ensemble" effectively

Problems

- In complex problems where $\mathcal{H}$ cannot be enumerated, it is sometimes possible to approximate Bayesian voting by drawing a random sample of hypotheses distributed according to $P(h|\mathbf{X}_l)$. E.g., Markov chain Monte Carlo methods seek to develop a set of tools for this task
- In practice, it is often difficult to construct a space $\mathcal{H}$ and assign a prior $P(h)$ that captures our prior knowledge adequately. In such case, the Bayesian committee is not optimal, and other ensemble methods may produce better results. In particular, the Bayesian approach does not address the computational and representational problems in any significant way

Manipulating the Training Examples

One can subsample training samples to generate multiple classifiers

- The learning algorithm is run several times, each time with a different subset of the training points
- This technique works especially well for unstable learning algorithms:

Manipulating the Training Examples

One can subsample training samples to generate multiple classifiers

- The learning algorithm is run several times, each time with a different subset of the training points
- This technique works especially well for unstable learning algorithms:
  - algorithms whose output classifier undergoes major changes in response to small changes in the training data
  - unstable algorithms:

Manipulating the Training Examples

One can subsample training samples to generate multiple classifiers

- The learning algorithm is run several times, each time with a different subset of the training points
- This technique works especially well for unstable learning algorithms:
  - algorithms whose output classifier undergoes major changes in response to small changes in the training data
  - unstable algorithms: decision tree, neural network, rule learning algorithms ...
  - stable algorithms:

Manipulating the Training Examples

One can subsample training samples to generate multiple classifiers

- The learning algorithm is run several times, each time with a different subset of the training points
- This technique works especially well for unstable learning algorithms:
  - algorithms whose output classifier undergoes major changes in response to small changes in the training data
  - unstable algorithms: decision tree, neural network, rule learning algorithms ...
  - stable algorithms: linear regression, nearest neighbor, linear threshold algorithms...

Manipulating the Training Examples

- The most straightforward way of manipulating the training set is called *Bagging*
- Another training set sampling method is to construct the training sets by leaving out disjoint subsets of the training data, e.g., cross-validated committee
- The third method for manipulating the training set is illustrated by the boosting

Manipulating the Input Features

The third general technique for generating multiple classifiers is to manipulate the set of input features available to the learning algorithm, e.g.,

- in a project to identify volcanoes on Venus, Cherkauer (1995) trained an ensemble of 32 neural networks
- the 32 neural networks were based on 8 different subsets of the 119 available input features and 4 different network sizes
- the input feature subsets were selected by hand to group features that were based on different image processing operations (such as PCA and the fast fourier transform)

This technique only works when the input features are highly redundant

Injection Randomness

The last general purpose method for generating ensembles of classifiers is to inject randomness into the learning algorithm

E.g., back-propagation algorithm for training neural networks

- the initial weights of the network are set randomly
- if the algorithm is applied to the same training set but with different initial weights, the resulting classifier can be quite different

Denote the measurement vector used by the j-th classifier by $\mathbf{x}^j$. In the measurement space each class ω_k is modeled by the probability density function $p(\mathbf{x}^j|\omega_w)$ and its a priori probability of occurrence is denoted by $P(\omega_k)$

- the models should be mutually exclusive: only one model can be associated with each measurement
- According to the Bayesian theory, given measurements $\mathbf{x}^j, j = 1, \cdots, J$, the pattern, $\mathbf{x}$, should be assigned to class ω_c provided the a posteriori probability of that interpretation is maximum, i.e.,

$$\begin{aligned} &\text{assign} \quad \mathbf{x} \rightarrow \mathbf{w}_m \text{ if} \\ &\quad P(\mathbf{w}_m|\mathbf{x}^1, \cdots, \mathbf{x}^J) = \max_c P(\omega_c|\mathbf{x}^1, \cdots, \mathbf{x}^J) \end{aligned}$$

The computation of the a posteriori probability functions would depend on the knowledge of high-order measurement statistics described in terms of joint probability density functions $p(\mathbf{x}^1, \cdots, \mathbf{x}^J|\omega_k)$ which would be difficult to infer

Therefore, we should attempt to simplify the above rule and express it in terms of decision support computations performed by the individual classifiers, each exploiting only the information conveyed by vector $\mathbf{x}^j$

- Product rule
- Sum rule
- min rule
- max rule
- median rule
- majority voting

Using the Bayes theorem, we can rewrite the a posteriori probability $P(\mathbf{w}_c|\mathbf{x}^1, \cdots, \mathbf{x}^J)$

$$P(\mathbf{w}_c|\mathbf{x}^1, \cdots, \mathbf{x}^J) = \frac{p(\mathbf{x}^1, \cdots, \mathbf{x}^J|\omega_c)P(\omega_c)}{p(\mathbf{x}^1, \cdots, \mathbf{x}^J)}$$

where $p(\mathbf{x}^1, \cdots, \mathbf{x}^J)$ is the unconditional measurement joint probability density. This can be expressed in terms of the conditional measurement distributions as

$$p(\mathbf{x}^1, \cdots, \mathbf{x}^J) = \sum_{c=1}^{C} p(\mathbf{x}^1, \cdots, \mathbf{x}^J|\omega_c)P(\omega_c)$$

Product Rule

Assuming that the representations by the classifiers are conditionally statistically independent, so

$$p(\mathbf{x}^1, \cdots, \mathbf{x}^J | \omega_c) = \prod_{j=1}^{J} p(\mathbf{x}^j | \omega_c)$$

where $p(\mathbf{x}^j|\omega_c)$ is the measurement process model of the j-th representation. Substituting this to the above equation, we have:

$$P(\mathbf{w}_c|\mathbf{x}^1, \cdots, \mathbf{x}^J) = \frac{P(\omega_c) \prod_{j=1}^{J} p(\mathbf{x}^j|\omega_c)}{\sum_{c=1}^{C} P(\omega_c) \prod_{j=1}^{J} p(\mathbf{x}^j|\omega_c)}$$

We can obtain the decision rule

$$\text{assign} \quad \mathbf{x} \rightarrow \mathbf{w}_m \text{ if}$$

$$P(\omega_m) \prod_{j=1}^{J} p(\mathbf{x}^j|\omega_m) = \max_c P(\omega_c) \prod_{j=1}^{J} p(\mathbf{x}^j|\omega_c)$$

Product Rule

In terms of the a posteriori probabilities yielded by the respective classifiers, we can obtain the decision rule

$$\text{assign} \quad \mathbf{x} \rightarrow \mathbf{w}_m \text{ if}$$

$$P^{-(J-1)}(\omega_m) \prod_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c P^{-(J-1)}(\omega_c) \prod_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

since

$$p(\mathbf{x}^j|\omega_c) = \frac{P(\omega_c|\mathbf{x}^j)}{P(\omega_c)}$$

Sum Rule

- In some applications it may be appropriate further to assume that the a posteriori probabilities computed by the respective classifiers will not deviate dramatically from the prior probabilities.
- This is a rather strong assumption that it may be readily satisfied when the available observational discriminatory information is highly ambiguous due to high levels of noise

In such a situation we can assume that the a posteriori probabilities can be expressed as

$$P(\omega_c|\mathbf{x}^j) = P(\omega_c)(1 + \delta_{cj})$$

where δ_{cj} satisfies $\delta_{cj} << 1$

Sum Rule

$$P^{-(J-1)}(\omega_c)\prod_{j=1}^{J} P(\omega_c|\mathbf{x}^j) = P(\omega_c)\prod_{j=1}^{J}(1+\delta_{cj})$$

If we expand the product and neglect any terms of second and higher order, we can approximate the right-hand side of the above equation as

$$P(\omega_c)\prod_{j=1}^{J}(1+\delta_{cj}) = P(\omega_c) + P(\omega_c)\prod_{j=1}^{J}\delta_{cj}$$

Finally, we can obtain a sum decision rule

assign $\mathbf{x} \rightarrow \mathbf{w}_m$ if

$$(1-J)P(\omega_m) + \sum_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \left[(1-J)P(\omega_c) + \sum_{j=1}^{J} P(\omega_c|\mathbf{x}^j)\right]$$

Classifier Combination Strategies

The decision product and sum rules constitute the basic schemes for classifier combination. Many commonly used classifier combination strategies can be developed from these rules by noting that

$$\prod_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \leq \min_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

Classifier Combination Strategies

The decision product and sum rules constitute the basic schemes for classifier combination. Many commonly used classifier combination strategies can be developed from these rules by noting that

$$\prod_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \leq \min_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$
$$\leq \frac{1}{J}\sum_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \leq \max_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

Furthermore, the hardening of the a posteriori probabilities $P(\omega_c|\mathbf{x}^j)$ to produce binary valued function δ_{cj} as

$$\delta_{cj} = \begin{cases} 1 & \text{if } P(\omega_c|\mathbf{x}^j) = \max_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \\ 0 & \text{otherwise} \end{cases}$$

Max Rule

Starting from the sum rule

assign $\mathbf{x} \rightarrow \mathbf{w}_m$ if

$$(1-J)P(\omega_m) + \sum_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \left[(1-J)P(\omega_c) + \sum_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \right]$$

and approximating the sum by the maximum of the posterior probabilities, we obtain

assign $\mathbf{x} \rightarrow \mathbf{w}_m$ if

$$\begin{aligned} &(1-J)P(\omega_m) + J \max_{j=1}^{J} P(\omega_m|\mathbf{x}^j) \\ &= \max_c \left[(1-J)P(\omega_c) + J \max_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \right] \end{aligned}$$

Max Rule

$$\text{assign} \quad \mathbf{x} \to \mathbf{w}_m \text{ if}$$
$$(1-J)P(\omega_m) + J\max_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \left[(1-J)P(\omega_c) + J\max_{j=1}^{J} P(\omega_c|\mathbf{x}^j)\right]$$

Under the assumption of equal priors, the above simplifies to

$$\text{assign} \quad \mathbf{x} \to \mathbf{w}_m \text{ if}$$
$$\max_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \max_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

Min Rule

Starting from the product rule

assign $\mathbf{x} \rightarrow \mathbf{w}_m$ if

$$P^{-(J-1)}(\omega_m)\prod_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c P^{-(J-1)}(\omega_c)\prod_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

and bounding the product of posterior probabilities from the above we obtain

assign $\mathbf{x} \rightarrow \mathbf{w}_m$ if

$$P^{-(J-1)}(\omega_m)\min_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c P^{-(J-1)}(\omega_c)\min_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

Median Rule

Note that under the equal prior assumption, the sum rule can be viewed to be computing the average a posteriori probabilities for each class over all the classifier outputs, i.e.,

$$\text{assign} \quad \mathbf{x} \to \mathbf{w}_m \text{ if}$$
$$\frac{1}{J}\sum_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \frac{1}{J}\sum_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

If any of the classifiers outputs an a posteriori probability for some class which is an outlier, it will affect the average and this in turn could lead to an incorrect decision.

Median Rule

It is well known that a robust estimate of the mean is the median. It could therefore be more appropriate to base the combined decision on the median of the a posteriori probabilities. This then leads to the following rule:

$$\text{assign} \quad \mathbf{x} \rightarrow \mathbf{w}_m \text{ if}$$
$$\text{med}_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \text{med}_{j=1}^{J} P(\omega_c|\mathbf{x}^j)$$

Majority Voting

Starting from the sum rule

assign $\mathbf{x} \to \mathbf{w}_m$ if

$$(1-J)P(\omega_m) + \sum_{j=1}^{J} P(\omega_m|\mathbf{x}^j) = \max_c \left[(1-J)P(\omega_c) + \sum_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \right]$$

under the assumption of equal priors and by hardening the probabilities according to

$$\delta_{cj} = \begin{cases} 1 & \text{if } P(\omega_c|\mathbf{x}^j) = \max_{j=1}^{J} P(\omega_c|\mathbf{x}^j) \\ 0 & \text{otherwise} \end{cases}$$

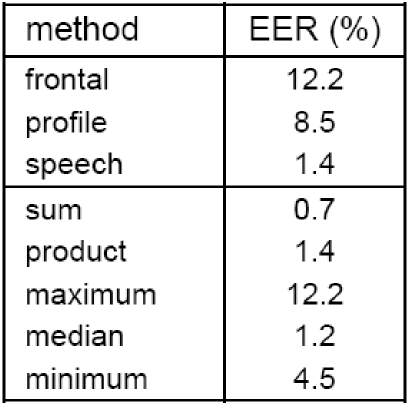
we find

assign $\mathbf{x} \to \mathbf{w}_m$ if

$$\sum_{j=1}^{J} \delta_{mj} = \max_c \sum_{j=1}^{J} \delta_{cj}$$

Experimental Results

EQUAL ERROR RATES



method	EER (%)
frontal	12.2
profile	8.5
speech	1.4
sum	0.7
product	1.4
maximum	12.2
median	1.2
minimum	4.5

The Boosting Approach

- devise computer program for deriving rough rules of thumb
- apply procedure to subset of examples
- obtain rule of thumb
- apply to 2nd subset of examples
- obtain 2nd rule of thumb
- repeat T times

Detail

- how to choose examples on each round?
  - concentrate on “hardest" examples (those most often misclassified by previous rules of thumb)
- how to combine rules of thumb into single prediction rule?
  - take (weighted) majority vote of rules of thumb

- boosting = general method of converting rough rules of thumb into highly accurate prediction rule
- technically:
    - assume given “weak" learning algorithm that can consistently find classifiers (“rules of thumb") at least slightly better than random, say, accuracy $\geq 55\%$ (in two-class setting)
    - given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%

Early Boosting Algorithms

- Let $h_1, \cdots, h_B$ be a set of hypotheses, and consider the composite ensemble hypothesis

$$f(\mathbf{x}) = \sum_{b=1}^{B} \alpha_b h_b(\mathbf{x})$$

- α_b and the learner or hypothesis h_b are to be learned within the Boosting procedure
- Kearns and Valiant were the first to pose the question of whether a “weak" learning algorithm which performs just slightly better than random guessing can be “boosted" into an arbitrarily accurate “strong" learning algorithms

Early Boosting Algorithms (contd)

- [Schapire '89]:
  - first provable boosting algorithm
    - call weak learner three times on three modified distributions
    - get slight boost in accuracy
    - apply recursively
- [Freund '90]:
  - "optimal" algorithm that "boosts by majority"
- [Drucker, Schapire & Simard '92]:
  - first experiments using boosting
  - limited by practical drawbacks

Boosting Problem

- “strong" PAC algorithm
  - for any distribution
  - $\forall\ \epsilon > 0,\ \delta > 0$
  - given polynomially many random examples
  - finds classifier with error $\leq\ \epsilon$ with probability $\geq\ 1 - \delta$
- “Weak" PAC algorithm
  - same, but only for $\epsilon \geq 1/2 - \gamma$
- does weak learnability imply strong learnability? [Kearns & Valiant’88]

A Formal Description of Boosting

- given training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$:
  - construct distribution D_t on $\{1, \ldots, m\}$
  - find weak classifier ("rule of thumb")

    $$h_t : X \to \{-1, +1\}$$

    with small error ϵ_t on D_t:

    $$\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$$
- output final classifier H_{final}

- Use bootstrapping to generate *B* training sets of bootstrap samples: selecting a set of examples (with replacement) from the original training set
- Train one base-learner with each bootstrapped training set
- Use voting (Average or median with regression)

Algorithm 1 The Bagging algorithm

```
Require: $(\mathrm{x}_i)_{i=1}^n, (y_i)_{i=1}^n, B$
1: for $b = 1$ to $B$ do
2:     The $j$-the set of training samples $T^b$ = bootstrapped samples of the original training set with replacement
3:     Creating a base classifier using $T^b$ as a training set
4: end for
5: Combining classifications using simple voting
```

Instability is an essential ingredient for bagging to improve accuracy.

- Unstable classifiers such as trees characteristically have high variance and low bias
- Stable classifiers like linear discriminant analysis have low variance, but can have high bias

The main effect of bagging is to reduce variance

AdaBoost

- constructing D_t:
  - $D_1(i) = 1/m$
  - given D_t and h_t:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

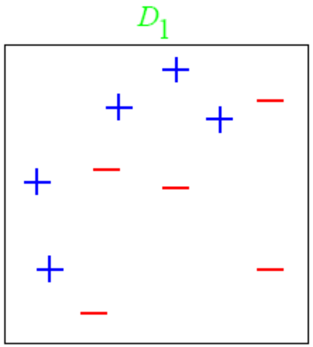
$$= \frac{D_t(i)}{Z_t} \exp(-\alpha_t \, y_i \, h_t(x_i))$$

where $Z_t =$ normalization constant

$$\alpha_t = \tfrac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

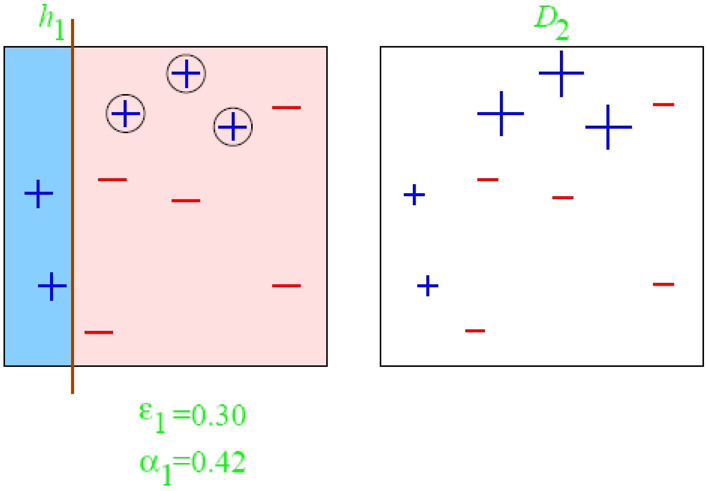
- final classifier:
  - $H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t h_t(x) \right)$

Toy Example

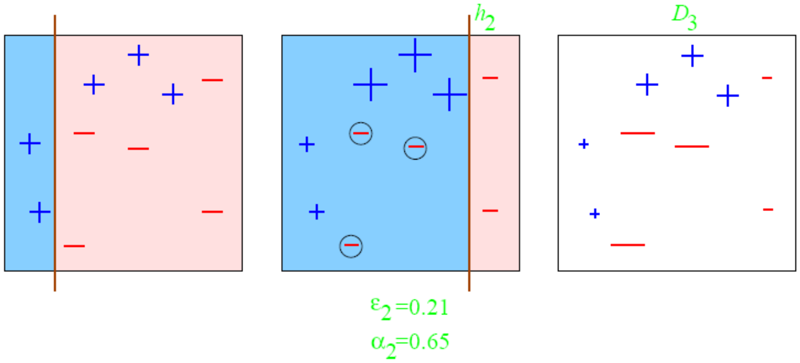


weak classifiers = vertical or horizontal half-planes

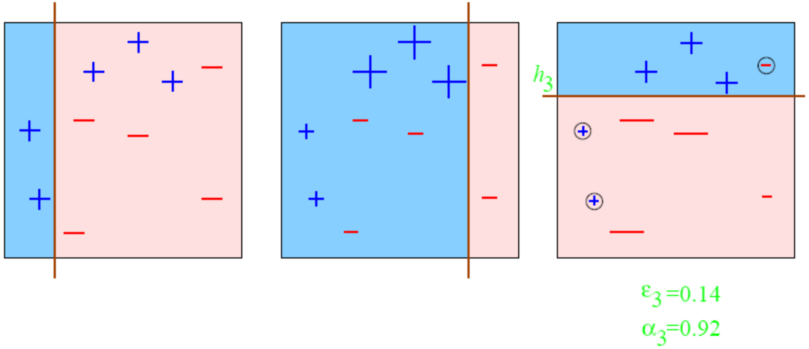
Toy Example - Round1



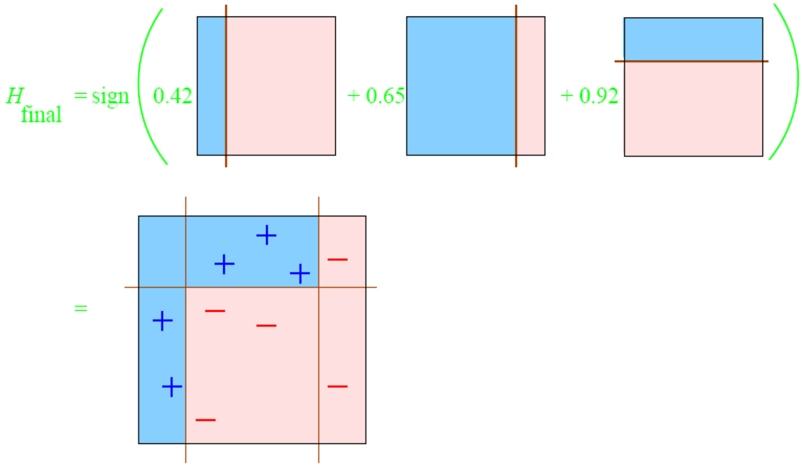
Toy Example - Round 2



Toy Example - Round 3



Toy Example - Final Classifier


$H_{\text{final}} = \text{sign}\left(0.42 \quad + 0.65 \quad + 0.92 \quad\right)$
=

Analyzing Training Error

- Theorem:
  - write ϵ_t as $1/2 - \gamma_t$
  - then

$$\begin{aligned}\text{training error}(H_{\text{final}}) &\leq \prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] \\ &= \prod_t \sqrt{1-4\gamma_t^2} \\ &\leq \exp\left(-2\sum_t \gamma_t^2\right)\end{aligned}$$

- so: if $\forall t: \ \gamma_t \geq \gamma > 0$
  then training error$(H_{\text{final}}) \leq e^{-2\gamma^2 T}$
- AdaBoost is adaptive:
  - does not need to know γ or T a priori
  - can exploit $\gamma_t \gg \gamma$

Training Error - Proof: Step 1

- let $f(x) = \sum_t \alpha_t h_t(x) \Rightarrow H_{\text{final}}(x) = \text{sign}(f(x))$
- *Step 1*: unwrapping recurrence:

$$D_{\text{final}}(i) = \frac{1}{m} \frac{\exp\left(-y_i \sum_t \alpha_t h_t(x_i)\right)}{\prod_t Z_t}$$

$$= \frac{1}{m} \frac{\exp\left(-y_i f(x_i)\right)}{\prod_t Z_t}$$

Training Error - Proof: Step 2

- *Step 2*: training error$(H_{\text{final}}) \le \prod_t Z_t$
- Proof:

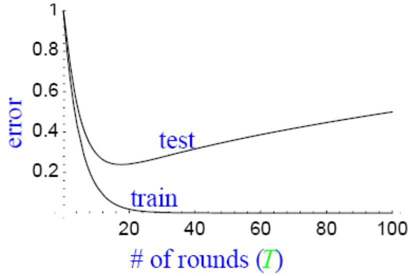
$$\begin{aligned}
\text{training error}(H_{\text{final}}) &= \frac{1}{m}\sum_i \begin{cases} 1 & \text{if } y_i \neq H_{\text{final}}(x_i) \\ 0 & \text{else} \end{cases} \\
&= \frac{1}{m}\sum_i \begin{cases} 1 & \text{if } y_i f(x_i) \le 0 \\ 0 & \text{else} \end{cases} \\
&\le \frac{1}{m}\sum_i \exp(-y_i f(x_i)) \\
&= \sum_i D_{\text{final}}(i) \prod_t Z_t \\
&= \prod_t Z_t
\end{aligned}$$

Training Error - Proof: Step 3

- *Step 3*: $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
- Proof:

$$\begin{aligned} Z_t &= \sum_i D_t(i) \exp(-\alpha_t \, y_i \, h_t(x_i)) \\ &= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} \\ &= \epsilon_t \, e^{\alpha_t} + (1-\epsilon_t) \, e^{-\alpha_t} \\ &= 2\sqrt{\epsilon_t(1-\epsilon_t)} \end{aligned}$$

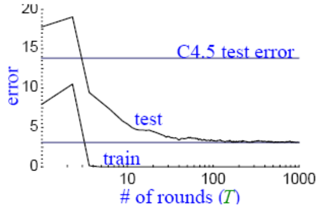
How Will Test Error Behavior?



expect:

- training error to continue to drop (or reach zero)
- test error to increase when H_{final} becomes "too complex"
  - "Occam's razor"
  - overfitting
    - hard to know when to stop training

Actual Typical Run



(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

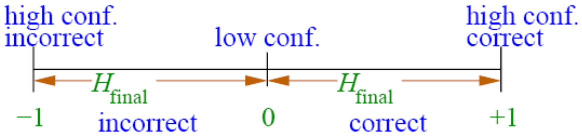
rounds

	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

- Occam's razor wrongly predicts "simpler" rule is better

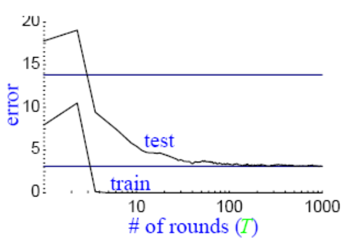
Theory of Margin

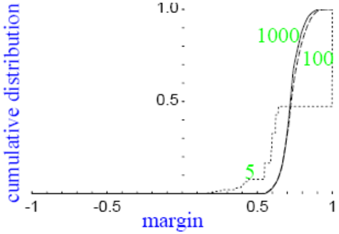
- key idea:
  - training error only measures whether classifications are right or wrong
  - should also consider confidence of classifications
- recall: H_{final} is weighted majority vote of weak classifiers
- measure confidence by margin = strength of the vote
  = (fraction voting correctly) − (fraction voting incorrectly)



Empirical Evidence: The Margin Distribution

- margin distribution
  = cumulative distribution of margins of training examples





	# rounds		
	5	100	1000
train 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

Theoretical Evidence: Analyzing Boosting Using Margins

- Theorem: large margins $\Rightarrow$ better bound on generalization error (independent of number of rounds)
  - proof idea: if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)
- Theorem: boosting tends to increase margins of training examples (given weak learning assumption)
  - proof idea: similar to training error proof
- so:
  although final classifier is getting larger,
  margins are likely to be increasing,
  so final classifier actually getting close to a simpler classifier,
  driving down the test error

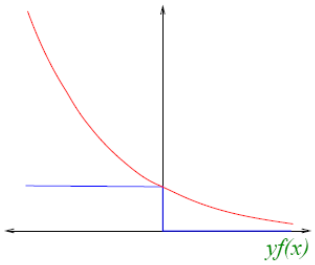
AdaBoost and Exponential Loss

- many (most?) learning algorithms minimize a "loss" function
  - e.g. least squares regression
- training error proof shows AdaBoost actually minimizes

$$\prod_t Z_t = \frac{1}{m}\sum_i \exp(-y_i f(x_i))$$

where $f(x) = \sum_t \alpha_t h_t(x)$

- on each round, AdaBoost greedily chooses α_t and h_t to minimize loss
- exponential loss is an upper bound on 0-1 (classification) loss
- AdaBoost provably minimizes exponential loss



Multiclass Problem

- say $y \in Y = \{1, \ldots, k\}$
- direct approach (AdaBoost.M1):

$$h_t : X \to Y$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$H_{\text{final}}(x) = \arg\max_{y \in Y} \sum_{t: h_t(x) = y} \alpha_t$$

- can prove same bound on error if $\forall t : \epsilon_t \leq 1/2$
  - in practice, not usually a problem for "strong" weak learners (e.g., C4.5)
  - significant problem for "weak" weak learners (e.g., decision stumps)
- instead, reduce to binary

Reducing Multiclass to Binary Problem

- say possible labels are $\{a, b, c, d, e\}$
- each training example replaced by five $\{-1, +1\}$-labeled examples:

$$x \ , \ c \rightarrow \begin{cases} (x, a) \ , \ -1 \\ (x, b) \ , \ -1 \\ (x, c) \ , \ +1 \\ (x, d) \ , \ -1 \\ (x, e) \ , \ -1 \end{cases}$$

- predict with label receiving most (weighted) votes

Multiclass Problem - Multi-label Problem

AdaBoost.MH

- can prove:

$$\text{training error}(H_{\text{final}}) \leq \frac{k}{2} \cdot \Pi\, Z_t$$

  - reflects fact that small number of errors in binary predictors can cause overall prediction to be incorrect

- extends immediately to multi-label case (more than one correct label per example)

Boosting: Conclusion

- boosting is a practical tool for classification and other learning problems
  - grounded in rich theory
  - performs well experimentally
  - often (but not always!) resistant to overfitting
  - many applications and extensions
- many ways to think about boosting
  - none is entirely satisfactory by itself, but each useful in its own way
  - considerable room for further theoretical and experimental work

Comparison between Bagging and AdaBoost

- In the presence of noise, AdaBoost overfits the data badly while Bagging works well
- In low-noise cases, AdaBoost gives good performance as it is able to optimize the ensemble without overfitting

Adaboost

constructs a new algorithm to eliminate “residual" errors that have not been properly handled by the weighted vote of the previous classifier

- it is directly trying to optimize the weighted vote
- it is making a direct assault on the representational problem
- directly optimizing an ensemble can increase the risk of overfitting as the space of ensembles is usually larger than the hypothesis space of the original algorithm

Comparison between Bagging and AdaBoost (contd)

Bagging

Bagging constructs base classifier independently of the others by manipulating the input data:

- it is sampling from the space of all possible hypotheses with a bias toward hypotheses that give good accuracy on the training set
- its main effect will address the statistical problem and to a lesser extent, the computational problem
- In large datasets, Bagging cannot create significant diversity as bootstrap replicates of a large training set are very similar to the training set itself, and hence, the learned algorithm will not be very diverse