Ensemble Learning

Ronald J. Williams
CSG220, Spring 2007

Main General Idea

- Train several learners (called *base learners*) L_1, L_2, ..., L_m to produce different hypotheses h_1, h_2, ..., h_m.
- Given a test input x, combine each of these hypotheses $h_1(x)$, $h_2(x)$, ..., $h_m(x)$ in some way.
- If the individual hypotheses are different from each other in some way, then the hope is that the overall result is more reliable than that given by any individual hypothesis.

Combining Multiple Hypotheses

- One common way to combine hypotheses – weighted "voting":

$$f(x) = \sigma\left(\sum_{i=1}^{m} \alpha_i h_i(x)\right)$$

- Possibilities for σ:
  - sgn function (for classification)
  - identity function (for regression)
- Possibilities for voting weights α_i :
  - all equal
  - varying with i, depending on "confidence"
  - each a function of input x

Two Different Potential Benefits of an Ensemble

- As a way to produce more reliable predictions than any of the individual base learners
- As a way to expand the hypothesis class: e.g., hypotheses of the form

$$f(x) = \sigma\left(\sum_{i=1}^{m} \alpha_i h_i(x)\right)$$

may be more general than any of the component hypotheses h_1, h_2, ..., h_m.

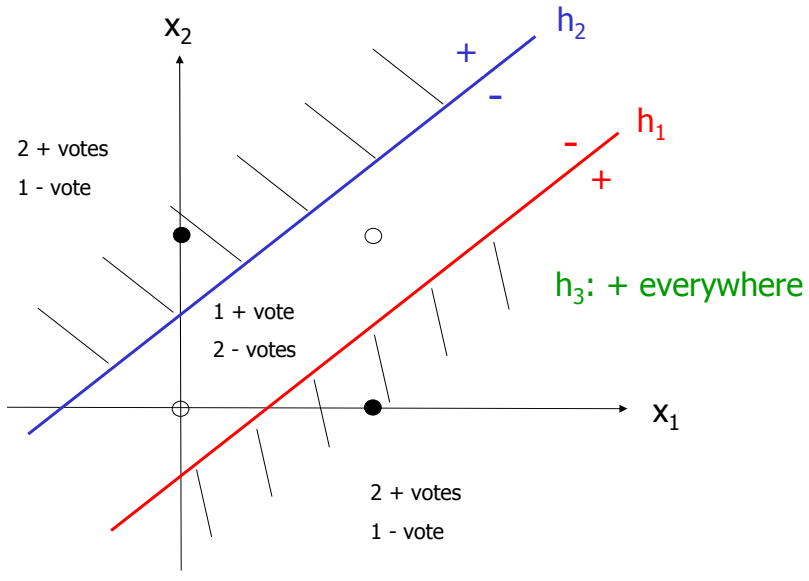
Expanding the hypothesis class

- Suppose our component hypotheses are linear separators in a 2-dimensional space.
- Let $x = (x_1, x_2)$, and consider these three linear separators:

  $h_1(x) = \text{sgn}(x_1 - x_2 - 0.5)$

  $h_2(x) = \text{sgn}(-x_1 + x_2 + 0.5)$

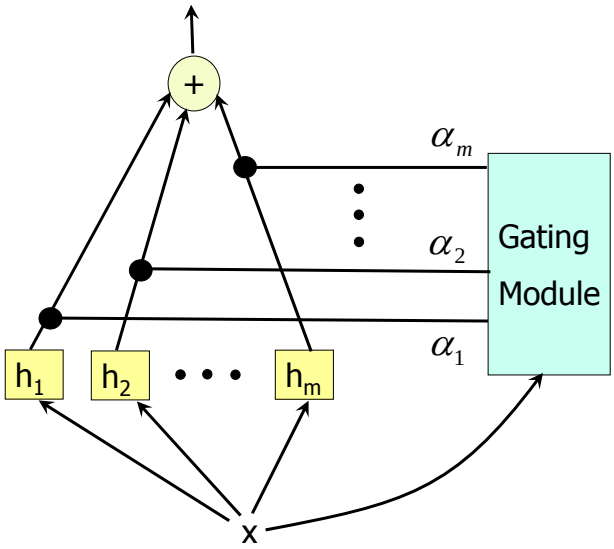
  $h_3(x) = 1$
- Then $\text{sgn}(h_1(x)+h_2(x)+h_3(x))$ computes a simple majority classification, and implements the XOR function.



How to get component hypotheses

- Use different base learners
  - e.g., decision tree, neural net, naïve Bayes, etc.
- Use different hyperparameters
- Use different input representations
  - includes possibility of different sensor modalities: visual, aural, etc.
- Use different training sets
  - bagging
  - boosting
- Use different problem decompositions
  - built in using prior knowledge
  - adaptive: mixture-of-experts

Mixture of Experts



Mixture of Experts

- The confidence weights are a function of the input
- Can take into account that some "experts" may be more accurate for certain inputs than for others
- If each confidence weight is large only in some local region of the input space, acts like a collection of local-fit models
- Both the hypotheses and the gating module (confidence weights) are trained

Bagging

- Bootstrap aggregating (Breiman, 1996)
- Given a fixed collection of R training data
  - Create multiple training sets of size R by sampling with replacement from the full set
  - Hypothesis h_i is the result of training the base learner on the i^{th} such training set
- For classification, take a majority vote
- For regression, use the mean or median
- Helps most when the base learner is *unstable*, meaning the learned hypothesis is highly sensitive to small variations in the training data
  - decision trees
  - neural nets (in this case because of sensitivity to initial weight values)

Boosting

- *Strong Learner*
  - can achieve true error rate arbitrarily close to 0 for any underlying distribution of training/test data
- *Weak Learner*
  - can achieve true error rate better than random guessing for any underlying distribution of training/test data, but not arbitrarily close to 0
- These concepts come from PAC-learning theory, but we'll forego the full technicalities involved.

Boosting

- A question posed early on in PAC-learning theory was:
  - Is it possible to turn any weak learner into a strong learner?
    - called *boosting* the weak learner
- Turns out the answer is yes
- We'll first do a simplistic analysis of one way (first proposed by Schapire, 1990)

Note: From now on, we restrict attention to 2-class classification problems.

A specific example

- Suppose we have a base learner that is able to classify randomly chosen instances correctly 80% of the time (for any distribution of instances).
- Assume given a fixed training set (of say, 100 examples). Training on this data, our base learner produces a hypothesis h_1 that classifies 80% of these correctly, and 20% of them incorrectly.
- What should we do to try to create a hypothesis with better accuracy than h_1's 80% accuracy?

Note: To keep things simple, we're pretending that training error and true error are the same thing.

Key Idea

- A key idea in boosting is to focus on data that our base learner gets wrong.
- Let's try this:
  - Run the base learner again, this time on the 20% of the training set that h_1 misclassifies.
  - Call the resulting hypothesis h_2.
- Looks promising, but will this work?

Key Idea

- A key idea in boosting is to focus on data that our base learner gets wrong.
- Let's try this:
  - Run the base learner again, this time on the 20% of the training set that h_1 misclassifies.
  - Call the resulting hypothesis h_2.
- Looks promising, but will this work?
- Well, h_2 should be correct on 80% of the 20 (out of 100) examples h_1 misclassified, so between h_1 and h_2, we're getting 96% right.

Key Idea

- A key idea in boosting is to focus on data that our base learner gets wrong.
- Let's try this:
  - Run the base learner again, this time on the 20% of the training set that h_1 misclassifies.
  - Call the resulting hypothesis h_2.
- Looks promising, but will this work?
- Well, h_2 should be correct on 80% of the 20 (out of 100) examples h_1 misclassified, so between h_1 and h_2, we're getting 96% right.
- What's the problem?

Problem with our first attempt

- Given a new instance to classify, how will we use h_1 and h_2?
- If we knew it would be misclassified by h_1, we should let h_2 classify it.
- But how can we know this?
- h_1 only tells us + or –.
- If we had additional information of the form "this is an instance that h_1 is likely to misclassify" we could take advantage of this.
- But we don't.

Problem with our first attempt

- Given a new instance to classify, how will we use h_1 and h_2?
- If we knew it would be misclassified by h_1, we should let h_2 classify it.
- But how can we know this?
- h_1 only tells us + or –.
- If we had additional information of the form "this is an instance that h_1 is likely to misclassify" we could take advantage of this.
- But we don't.

The mixture-of-experts approach is more in this spirit, however.

Another problem with our first attempt

- What if all h_2 does is produce the opposite answer to what h_1 produces?
- In this case it could get 0% error on the instances h_1 misclassifies, but it really won't help.
- Suggests a key additional insight:

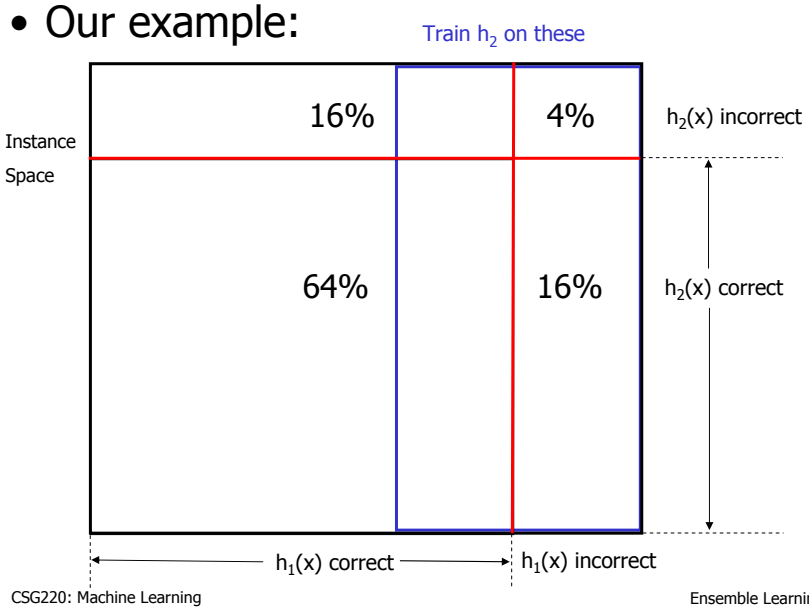
  Train on the instances h_1 misclassifies, but also include some of the other data as well.

Majority-of-Three Boosting

- Run the base learner on the original training data to get hypothesis h_1 whose error rate is $\leq \varepsilon < ½$.
- Form a new training set consisting of the data misclassified by h_1 together with an equal number of data chosen randomly from the remaining original training data.
- Run the base learner on this training set to get hypothesis h_2.
- Form another training set consisting solely of instances x for which $h_1(x) \neq h_2(x)$.
- Run the base learner on this training set to get hypothesis h_3.
- To classify any new instance, take a majority vote of h_1, h_2, and h_3.

Sketch of why this works

- Our example:



Sketch of why this works (cont.)

- h_3 trained on the 32% where h_1 and h_2 disagree.
- So h_3 expected to get 80% of these, or 25.6%, correct.
- Thus h_1 and h_2 agree with each other and are correct on 64%.
- Where they disagree, h_3 will break the tie and be correct 25.6% of the time.
- Thus the majority is correct 64%+25.6% = 89.6% of the time.
- Error rate reduced from 20% to 10.4%.

General Result on Majority-of-Three Boosting

- Schapire, 1990
- If the base learner has (true) error rate $\leq \varepsilon$ for any possible distribution of examples, then this majority-of-three classifier has true error rate $\leq 3\varepsilon^2-2\varepsilon^3$ for any distribution of examples.

Correct formal statement of this in PAC-learning framework involves confidence bounds $1-\delta$ as well, but we ignore this for simplicity.

Majority-of-Three really does boost

- This result implies that if the true error rate ε is bounded above by a number less than ½, then the error rate of this majority-of-three classifier is $< \varepsilon$:
  - Write $3\varepsilon^2-2\varepsilon^3 = \varepsilon(3\varepsilon -2\varepsilon^2)$
  - Observe that $f(\varepsilon) = 3\varepsilon -2\varepsilon^2$ is a strictly increasing function for $\varepsilon \leq ¾$
  - Conclude that $\varepsilon < ½$ implies $f(\varepsilon) < f(½) = 1$, so $\varepsilon(3\varepsilon -2\varepsilon^2) = \varepsilon f(\varepsilon) < \varepsilon$ when $\varepsilon < ½$.
- Furthermore, this process can be applied recursively to obtain arbitrarily small error rate.

AdaBoost

- Adaptive Boosting (Freund & Schapire, 1995)
- Entire training set used at every *round*
- Each round produces an additional component hypothesis
- Majority-of-three boosting is boosting by filtering
- AdaBoost strategy is boosting by reweighting
- Initially all data given equal weight
- On each successive round, the data is reweighted to emphasize hard-to-classify instances from previous rounds
  - weights of correctly classified examples are decreased
  - weights of incorrectly classified examples are increased

AdaBoost Algorithm

- Training data $\{(x_i, y_i) \mid 1 \le i \le R\}$ where each y_i is ± 1
- Have data weights $\{w_i \mid 1 \le i \le R\}$, one for each training example, normalized to sum to 1.
- Initially, $w_i = 1/R$ for all i.
- After each round t, a new hypothesis h_t is learned, along with a corresponding confidence weight α_t.
- After T rounds, the classifier classifies any instance x according to the weighted majority

$$H(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

AdaBoost Algorithm: Round t

1. Train the base learner on the weighted training set $\{(x_i, y_i, w_i) \mid 1 \le i \le R\}$. Call the resulting hypothesis h_t.
2. Compute weighted error rate $\varepsilon_t = \sum_{i : h_t(x_i) \neq y_i} w_i$
3. Set hypothesis confidence weight $\alpha_t = \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
4. For each training example, modify its weight as follows:

$$w_i \leftarrow \begin{cases} w_i(\varepsilon_t/(1-\varepsilon_t)) & \text{if } h_t(x_i) = y_i \\ w_i & \text{if } h_t(x_i) \neq y_i \end{cases}$$

5. Normalize weights so they sum to 1

Base Learners Used

- Decision trees (appropriately pruned)
- Decision stumps (special case of height 1, i.e., a tree having exactly one test node)
- Neural nets (with limited hidden nodes)
- Perceptrons
- Naïve Bayes

Weighted Training Data

- Just about every learning algorithm we've studied has a straightforward weighted data version
  - Neural nets: Try to minimize $\sum_i w_i E_i$, where

    E_i is the error on the i[th] training example
  - Decision trees, naïve Bayes: Use the sum of weights in place of counts throughout
- If no straightforward weighted version exists, two possibilities:
  - Replicate training examples so relative counts match weights (preferable)
  - Draw training data randomly (with replacement) using weights to determine distribution

Example: PlayTennis Data

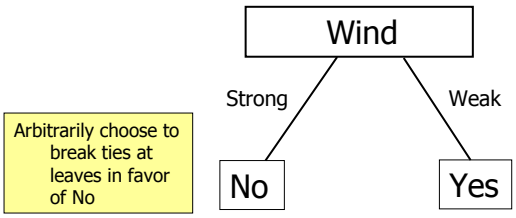
- Use decision stumps as our base learner
- Replace ID3 maximum information gain criterion with minimum weighted error criterion (done in this example purely for convenience)

It turns out that with equally weighted data, using this criterion to build the entire tree would create exactly the same tree as the one created using the ID3 criterion.

Training Data for Round 1

Day	Outlook	Temperature	Humidity	Wind	PlayTennis	Weight
D1	Sunny	Hot	High	Weak	No	1/14
D2	Sunny	Hot	High	Strong	No	1/14
D3	Overcast	Hot	High	Weak	Yes	1/14
D4	Rain	Mild	High	Weak	Yes	1/14
D5	Rain	Cool	Normal	Weak	Yes	1/14
D6	Rain	Cool	Normal	Strong	No	1/14
D7	Overcast	Cool	Normal	Strong	Yes	1/14
D8	Sunny	Mild	High	Weak	No	1/14
D9	Sunny	Cool	Normal	Weak	Yes	1/14
D10	Rain	Mild	Normal	Weak	Yes	1/14
D11	Sunny	Mild	Normal	Strong	Yes	1/14
D12	Overcast	Mild	High	Strong	Yes	1/14
D13	Overcast	Hot	Normal	Weak	Yes	1/14
D14	Rain	Mild	High	Strong	No	1/14

Potential Stump for Wind



Arbitrarily choose to break ties at leaves in favor of No

D2, D6, D7, D11, D12, D14

PlayTennis: 3/14 Yes, 3/14 No

Weighted Error = 5/14

D1, D3, D4, D5, D8, D9, D10, D13

PlayTennis: 6/14 Yes, 2/14 No

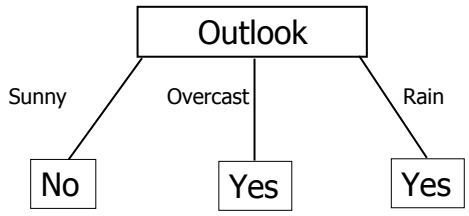
- Doing this for all 4 possible stumps yields

Attribute Tested	Weighted Error
Outlook	4/14
Temperature	5/14
Humidity	5/14
Wind	5/14

- Therefore the best choice of stump for round 1 is a test for the value of Outlook

Happens to be same result we'd get using ID3 criterion

Resulting h_1 stump



Misclassifies D9, D11

Misclassifies D6, D14

$$\varepsilon_1 = 4/14$$

$$\alpha_1 = \ln\left(\frac{1-\varepsilon_1}{\varepsilon_1}\right) = \ln(2.5) \approx 0.916$$

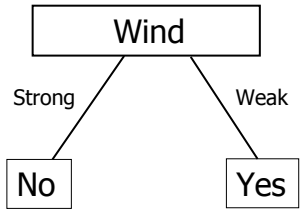
Modifying the weights for round 2

- Before normalization:
  - the weights for the incorrectly classified examples D6, D9, D11, and D14 stay at 1/14
  - the weights on the other 10 examples are multiplied by $\varepsilon/(1-\varepsilon) = (4/14)/(10/14) = 2/5$, becoming 1/35
- After normalization:
  - the weights for D6, D9, D11, and D14 become 1/8
  - the weights for the other 10 examples become 1/20

Training Data for Round 2

Day	Outlook	Temperature	Humidity	Wind	PlayTennis	Weight
D1	Sunny	Hot	High	Weak	No	1/20
D2	Sunny	Hot	High	Strong	No	1/20
D3	Overcast	Hot	High	Weak	Yes	1/20
D4	Rain	Mild	High	Weak	Yes	1/20
D5	Rain	Cool	Normal	Weak	Yes	1/20
D6	Rain	Cool	Normal	Strong	No	1/8
D7	Overcast	Cool	Normal	Strong	Yes	1/20
D8	Sunny	Mild	High	Weak	No	1/20
D9	Sunny	Cool	Normal	Weak	Yes	1/8
D10	Rain	Mild	Normal	Weak	Yes	1/20
D11	Sunny	Mild	Normal	Strong	Yes	1/8
D12	Overcast	Mild	High	Strong	Yes	1/20
D13	Overcast	Hot	Normal	Weak	Yes	1/20
D14	Rain	Mild	High	Strong	No	1/8

Potential Stump for Wind (round 2)



D2, D6, D7, D11, D12, D14
PlayTennis: 0.225 Yes, 0.300 No
Weighted Error = 0.325

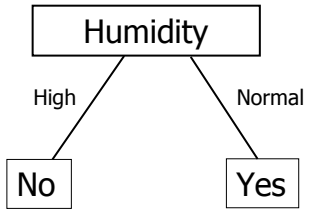
D1, D3, D4, D5, D8, D9, D10, D13
PlayTennis: 0.375 Yes, 0.100 No

- Doing this for all 4 possible stumps yields

Attribute Tested	Weighted Error
Outlook	0.300
Temperature	0.400
Humidity	0.275
Wind	0.325

- Therefore the best choice of stump in this round is a test for the value of Humidity (using our min. weighted error criterion)

Resulting h_2 stump



$$\varepsilon_2 = 0.275$$

$$\alpha_2 = \ln\left(\frac{1-\varepsilon_2}{\varepsilon_2}\right) \approx \ln(2.636) \approx 0.969$$

Modifying the weights for round 3

- Before normalization:
  - the weights for the now incorrectly classified examples D3, D4, D6, D12 are not changed
  - the weights on the other 10 examples are multiplied by ε/(1-ε) = 0.379
- After normalization, the new weights look like this:

Training Data for Round 3

Day	Outlook	Temperature	Humidity	Wind	PlayTennis	Weight
D1	Sunny	Hot	High	Weak	No	0.0345
D2	Sunny	Hot	High	Strong	No	0.0345
D3	Overcast	Hot	High	Weak	Yes	0.0909
D4	Rain	Mild	High	Weak	Yes	0.0909
D5	Rain	Cool	Normal	Weak	Yes	0.0345
D6	Rain	Cool	Normal	Strong	No	0.2273
D7	Overcast	Cool	Normal	Strong	Yes	0.0345
D8	Sunny	Mild	High	Weak	No	0.0345
D9	Sunny	Cool	Normal	Weak	Yes	0.0862
D10	Rain	Mild	Normal	Weak	Yes	0.0345
D11	Sunny	Mild	Normal	Strong	Yes	0.0862
D12	Overcast	Mild	High	Strong	Yes	0.0909
D13	Overcast	Hot	Normal	Weak	Yes	0.0345
D14	Rain	Mild	High	Strong	No	0.0862

- Etc. ...

General Observations on AdaBoost

- If $\varepsilon_t > ½$ then $\alpha_t < 0$, which is appropriate since switching the sign on the result of the corresponding hypothesis would give error rate less than ½
- If $\varepsilon_t = 0$ then $\alpha_t = \infty$, which corresponds to the result that the ensemble should consist of that single hypothesis alone
- If $\varepsilon_t = ½$ then $\alpha_t = 0$, which means
  - the corresponding hypothesis contributes nothing to the ensemble
  - the data weights will remain unchanged in every subsequent round
  - boosting cannot give zero training error
- When $\varepsilon_t = ½$ occurs we conclude that no ensemble of weak hypotheses from the given class can fit this training data
- After each round, the data weights are adjusted to concentrate exactly half the overall weight on examples misclassified in that round

Can you see how to prove this?

Reweighting Step

Can check that, as long as $0 < \varepsilon_t < 1$, step 4 of the AdaBoost algorithm can be replaced by any of these and the same weights will result after normalization:

$$w_i \leftarrow \begin{cases} w_i \varepsilon_t & \text{if } h_t(x_i) = y_i \\ w_i (1 - \varepsilon_t) & \text{if } h_t(x_i) \neq y_i \end{cases}$$

$$w_i \leftarrow \begin{cases} w_i & \text{if } h_t(x_i) = y_i \\ w_i ((1 - \varepsilon_t) / \varepsilon_t) & \text{if } h_t(x_i) \neq y_i \end{cases}$$

$$w_i \leftarrow \begin{cases} w_i \sqrt{\varepsilon_t / (1 - \varepsilon_t)} & \text{if } h_t(x_i) = y_i \\ w_i \sqrt{(1 - \varepsilon_t) / \varepsilon_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

Hypothesis Confidence Weight

- Easy to see that the same ensemble classifier results if all the α_t values are multiplied by the same positive factor.
- In particular, the overall result is equivalent if we change the definition of α_t in step 3 to

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

We'll see later why this is useful

An aside: Using unnormalized weights

- Note that it is not really necessary to perform the normalization step 5, as long as steps 1 (training the learner) and steps 2 (computing the weighted error) take this into account.
- Provides yet another computational alternative.
- For example, when combined with the use of this update in step 3,

$$w_i \leftarrow \begin{cases} w_i & \text{if } h_t(x_i) = y_i \\ w_i((1-\varepsilon_t)/\varepsilon_t) & \text{if } h_t(x_i) \neq y_i \end{cases}$$

the following is the evolution of the unnormalized weights in our PlayTennis decision stump example:

Day	Initial Weight	Weight After Round 1	Weight After Round 2
D1	1.000	1.000	1.000
D2	1.000	1.000	1.000
D3	1.000	1.000	2.636
D4	1.000	1.000	2.636
D5	1.000	1.000	1.000
D6	1.000	2.500	6.591
D7	1.000	1.000	1.000
D8	1.000	1.000	1.000
D9	1.000	2.500	2.500
D10	1.000	1.000	1.000
D11	1.000	2.500	2.500
D12	1.000	1.000	2.636
D13	1.000	1.000	1.000
D14	1.000	2.500	2.500

Analysis of AdaBoost

For purposes of this analysis, to simplify the math, we consider the equivalent AdaBoost variant in which step 3 sets the hypothesis confidence weight to

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

and the step 4 weight update is

$$w_i \leftarrow \begin{cases} w_i\sqrt{\varepsilon_t/(1-\varepsilon_t)} & \text{if } h_t(x_i) = y_i \\ w_i\sqrt{(1-\varepsilon_t)/\varepsilon_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

Reweighting Including Normalization

- Note that

$$y_i h_t(x_i) = \begin{cases} +1 & \text{if } h_t(x_i) = y_i \\ -1 & \text{if } h_t(x_i) \neq y_i \end{cases}$$

- Thus

$$e^{-\alpha_t y_i h_t(x_i)} = \begin{cases} \sqrt{\varepsilon_t / (1-\varepsilon_t)} & \text{if } h_t(x_i) = y_i \\ \sqrt{(1-\varepsilon_t)/\varepsilon_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

when

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

Full Reweighting

- Thus step 4 in the variant we're analyzing can be written as

$$w_i \leftarrow w_i e^{-\alpha_t y_i h_t(x_i)}$$

- Then, after normalization, the weight update from round t to round t+1 is

$$w_{i,t+1} = w_{i,t} e^{-\alpha_t y_i h_t(x_i)} / z_t$$

where the normalization constant is

$$z_t = \sum_{i=1}^{R} w_{i,t} e^{-\alpha_t y_i h_t(x_i)}$$

and we have introduced a second subscript on each data weight to identify the round.

AdaBoost Training Error Analysis

- We now analyze the training set error behavior of AdaBoost.
- Recall that the ensemble classifier after T rounds is the function given by $H(x) = \text{sgn}\big(f(x)\big)$ where

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

- For any proposition (boolean) *p*, define the indicator function

$$I(p) = \begin{cases} 1 & \text{if } p \text{ is true} \\ 0 & \text{if } p \text{ is false} \end{cases}$$

Training Error Analysis (cont.)

- Theorem.

$$\frac{1}{R}\sum_{i=1}^{R} I\big(H(x_i) \neq y_i\big) \leq \prod_{t=1}^{T} z_t$$

Training set error rate for the combined classifier

A quantity we'll show decays exponentially

Proof:

$$w_{i,T+1} = \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot w_{i,T}$$

Applying our earlier result on the normalized weight update rule from round T to round T+1

$$= \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot \frac{e^{-\alpha_{T-1} y_i h_{T-1}(x_i)}}{z_{T-1}} \cdot w_{i,T-1}$$

$$= \cdots$$

$$= \frac{e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x_i)}}{\prod_{t=1}^{T} z_T} \cdot w_{i,1}$$

$$= \frac{e^{-y_i f(x_i)}}{\prod_{t=1}^{T} z_T} \cdot \frac{1}{R}$$

Proof:

$$w_{i,T+1} = \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot w_{i,T}$$

Substituting the result of applying it again for the update from round T-1 to round T

$$= \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot \frac{e^{-\alpha_{T-1} y_i h_{T-1}(x_i)}}{z_{T-1}} \cdot w_{i,T-1}$$

$$= \cdots$$

$$= \frac{e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x_i)}}{\prod_{t=1}^{T} z_T} \cdot w_{i,1}$$

$$= \frac{e^{-y_i f(x_i)}}{\prod_{t=1}^{T} z_T} \cdot \frac{1}{R}$$

Proof:

$$w_{i,T+1} = \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot w_{i,T}$$

$$= \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot \frac{e^{-\alpha_{T-1} y_i h_{T-1}(x_i)}}{z_{T-1}} \cdot w_{i,T-1}$$

$$= \cdots$$

$$= \frac{e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x_i)}}{\prod_{t=1}^{T} z_T} \cdot w_{i,1}$$

Etc., all the way down to round 1

$$= \frac{e^{-y_i f(x_i)}}{\prod_{t=1}^{T} z_T} \cdot \frac{1}{R}$$

Proof:

$$w_{i,T+1} = \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot w_{i,T}$$

$$= \frac{e^{-\alpha_T y_i h_T(x_i)}}{z_T} \cdot \frac{e^{-\alpha_{T-1} y_i h_{T-1}(x_i)}}{z_{T-1}} \cdot w_{i,T-1}$$

$$= \cdots$$

$$= \frac{e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x_i)}}{\prod_{t=1}^{T} z_T} \cdot w_{i,1}$$

Using the definition of f and the fact that all weights are initialized to 1/R

$$= \frac{e^{-y_i f(x_i)}}{\prod_{t=1}^{T} z_T} \cdot \frac{1}{R}$$

Therefore

$$e^{-y_i f(x_i)} = Rw_{i,T+1} \prod_{t=1}^{T} z_T$$

Call this Fact 1

Recall that our ensemble classifier is defined by $H(x) = \text{sgn}(f(x))$

If $H(x_i) \neq y_i$, then

- $I(H(x_i) \neq y_i) = 1$
- $y_i f(x_i) \leq 0$, so $e^{-y_i f(x_i)} \geq 1$
- Thus $e^{-y_i f(x_i)} \geq I(H(x_i) \neq y_i)$ in this case.

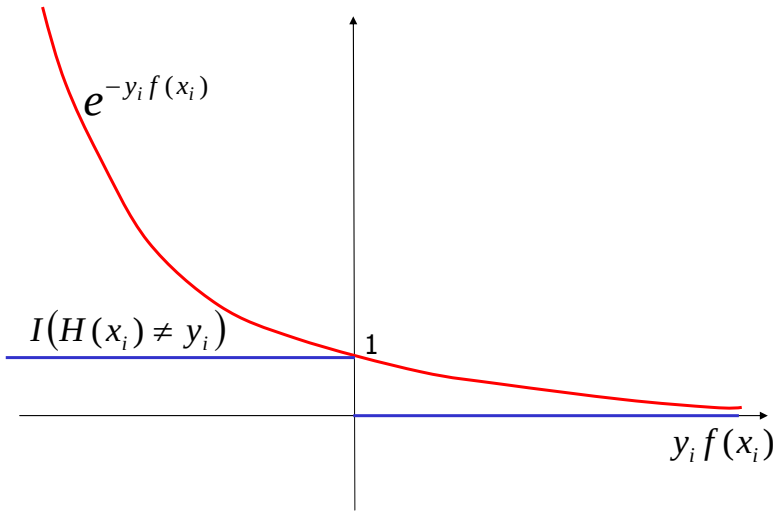
If $H(x_i) = y_i$, then

- $I(H(x_i) \neq y_i) = 0$
- $e^{-y_i f(x_i)} > 0$
- Thus $e^{-y_i f(x_i)} \geq I(H(x_i) \neq y_i)$ in this case also.

Therefore, $I(H(x_i) \neq y_i) \leq e^{-y_i f(x_i)}$

Call this Fact 2

Graphical Representation of Fact 2



Continuing the proof of the theorem

$$\frac{1}{R}\sum_{i=1}^{R} I\left(H(x_i) \neq y_i\right) \leq \frac{1}{R}\sum_{i=1}^{R} e^{-y_i f(x_i)}$$

by Fact 2

$$= \frac{1}{R}\sum_{i=1}^{R} R w_{i,T+1} \prod_{t=1}^{T} z_t$$

$$= \left(\prod_{t=1}^{T} z_t\right) \sum_{i=1}^{R} w_{i,T+1}$$

$$= \prod_{t=1}^{T} z_t$$

Continuing the proof of the theorem

$$\frac{1}{R}\sum_{i=1}^{R} I\big(H(x_i) \neq y_i\big) \leq \frac{1}{R}\sum_{i=1}^{R} e^{-y_i f(x_i)}$$

$$= \frac{1}{R}\sum_{i=1}^{R} R w_{i,T+1} \prod_{t=1}^{T} z_t$$

by Fact 1

$$= \left(\prod_{t=1}^{T} z_t\right)\sum_{i=1}^{R} w_{i,T+1}$$

$$= \prod_{t=1}^{T} z_t$$

Continuing the proof of the theorem

$$\frac{1}{R}\sum_{i=1}^{R} I\big(H(x_i) \neq y_i\big) \leq \frac{1}{R}\sum_{i=1}^{R} e^{-y_i f(x_i)}$$

$$= \frac{1}{R}\sum_{i=1}^{R} R w_{i,T+1} \prod_{t=1}^{T} z_t$$

$$= \left(\prod_{t=1}^{T} z_t\right)\sum_{i=1}^{R} w_{i,T+1}$$

$$= \prod_{t=1}^{T} z_t$$

since the weights in any round sum to 1

Continuing the proof of the theorem

$$\frac{1}{R}\sum_{i=1}^{R} I\big(H(x_i) \neq y_i\big) \leq \frac{1}{R}\sum_{i=1}^{R} e^{-y_i f(x_i)}$$

$$= \frac{1}{R}\sum_{i=1}^{R} R w_{i,T+1} \prod_{t=1}^{T} z_t$$

$$= \left(\prod_{t=1}^{T} z_t\right) \sum_{i=1}^{R} w_{i,T+1}$$

$$= \prod_{t=1}^{T} z_t \qquad \text{Q.E.D.}$$

Analysis of z_t

$$z_t = \sum_{i=1}^{R} w_{i,t} e^{-\alpha_t y_i h_t(x_i)}$$

$$= \sum_{i:h_t(x_i)=y_i} w_{i,t} e^{-\alpha_t} + \sum_{i:h_t(x_i)\neq y_i} w_{i,t} e^{\alpha_t}$$

$$= e^{-\alpha_t} \sum_{i:h_t(x_i)=y_i} w_{i,t} + e^{\alpha_t} \sum_{i:h_t(x_i)\neq y_i} w_{i,t}$$

Analysis of z_t (cont.)

By definition, $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

so

$$e^{\alpha_t} = \sqrt{(1-\varepsilon_t)/\varepsilon_t}$$

and

$$e^{-\alpha_t} = \sqrt{\varepsilon_t/(1-\varepsilon_t)}$$

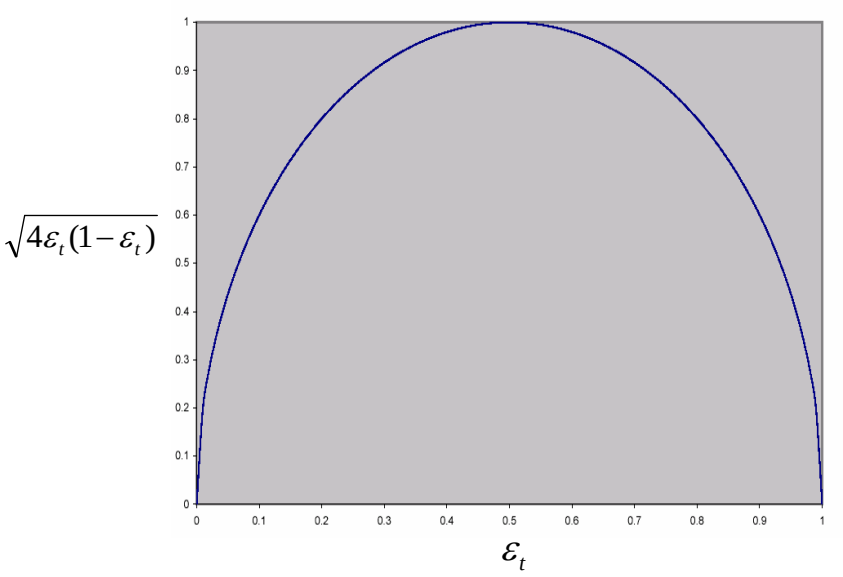
Furthermore,

$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} w_{i,t}$$

Analysis of z_t (cont.)

Therefore,

$$\begin{aligned} z_t &= e^{-\alpha_t} \sum_{i:h_t(x_i)=y_i} w_{i,t} + e^{\alpha_t} \sum_{i:h_t(x_i)\neq y_i} w_{i,t} \\ &= \sqrt{\varepsilon_t/(1-\varepsilon_t)}\cdot(1-\varepsilon_t) + \sqrt{(1-\varepsilon_t)/\varepsilon_t}\cdot\varepsilon_t \\ &= \sqrt{\varepsilon_t(1-\varepsilon_t)} + \sqrt{(1-\varepsilon_t)\varepsilon_t} \\ &= \sqrt{4\varepsilon_t(1-\varepsilon_t)} \end{aligned}$$



Exponential Convergence

- Together with the theorem, this result implies that if ε_t is bounded away from ½, z_t is bounded below 1, so the training error is bounded by a decaying exponential function
- So in this case training error eventually approaches zero exponentially as a function of the number of boosting rounds.

- What about true (generalization) error?

AdaBoost Generalization Error

- Early result by Freund and Schapire

$$\text{TESTERR} \leq \text{TRAINERR} + \tilde{O}\left(\sqrt{\frac{Td}{R}}\right)$$

true error rate

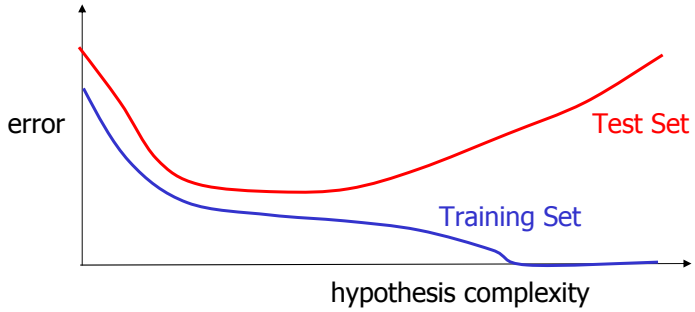
training data error rate

(with high probability) where

- T = number of rounds
- d = VC-dimension of base learner
- R = number of training examples
- $\tilde{O}$ hides log factors as well as constant factors

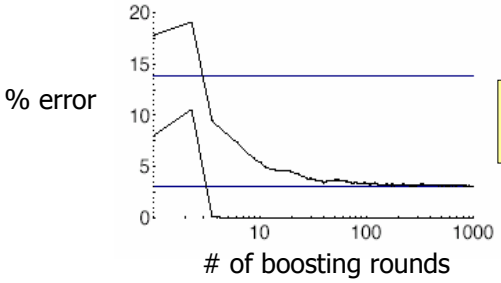
AdaBoost Generalization Error (cont.)

- Consistent with the standard wisdom that
  - boosting for more rounds (higher T) leads to a more and more complex ensemble hypothesis H
  - more complex hypotheses lead to greater chance of overfitting, and therefore generalizing poorly



Surprise!

- But many empirical studies have yielded results like this (from an actual reported experiment):



- Test error did not increase even after 1000 rounds.
- Test error continued to drop long after training error reached zero.
- How can this happen?

Margin of a Weighted-Sum Classifier

- Suppose we have an ensemble of classifiers $h_1, h_2, \ldots, h_T$ and corresponding confidence weights $\alpha_1, \alpha_2, \ldots, \alpha_T$
- Let the combined classifier be given by $H(x) = \text{sgn}(f(x))$, where

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

- Then the *margin* of the classifier H for a training example (x, y) is defined to be

$$\text{margin}(x, y) = yf(x) / \sum_{t=1}^{T} |\alpha_t|$$

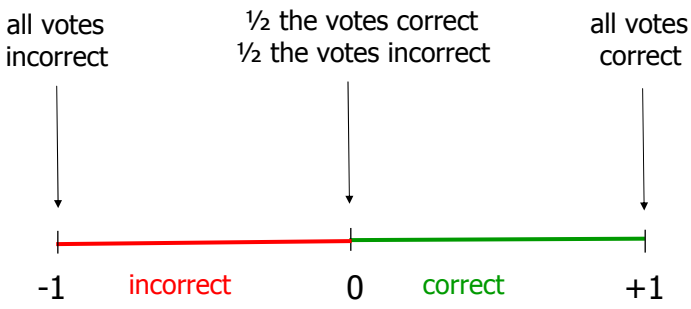
Understanding the Margin

- Underlying intutition: margin measures how close training instance is to the decision boundary
  - the larger, the better (as with SVMs)
- Note that we could more simply define it to be $yf(x)$ if the confidence weights were normalized to sum to 1 (and the absolute value is also unnecessary if we flip the output of any hypothesis whose weight is negative)
- Without some form of normalization, $yf(x)$ could trivially be increased without changing the classifier by multiplying all the confidence weights by some constant greater than 1

Understanding the Margin (cont.)

- The margin is always in [-1,+1].
- The margin is +1 if all "voters" agree on the correct classification.
- The margin is -1 if all "voters" agree on the incorrect classification.
- If the margin of (x,y) is negative, H misclassifies it.
- If the margin of (x,y) is positive, H classifies it correctly.
- If the margin of (x,y) is near zero, then the weighted votes for + are nearly balanced by the weighted votes for -.

Interpreting the Margin



Theoretical Benefit of Large Margins

- Another upper bound on generalization error: For any θ > 0,

$$\text{TESTERR} \le \frac{1}{R}\sum_{i=1}^{R} I\left(\text{margin}(x_i, y_i) \le \theta\right) + \tilde{O}\left(\sqrt{\frac{d}{R\theta^2}}\right)$$

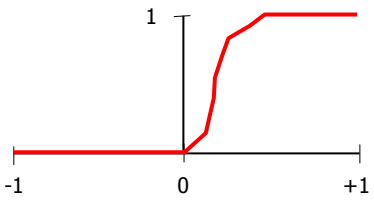
Represents a c.d.f. measuring the fraction of training examples with margin no larger than (i.e., no better than) θ

(with high probability) where d is the VC-dimension of the base learner.

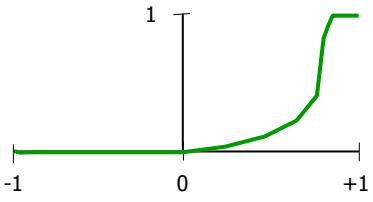
- Independent of the number T of boosting rounds
- In fact, it turns out that carrying out more rounds can continue to lower the first term well after zero error is achieved on the training set.

Margin c.d.f.'s

Poor generalization bound



Good generalization bound



Both have zero error on the training data since there are no negative margins.

Empirical Demonstration

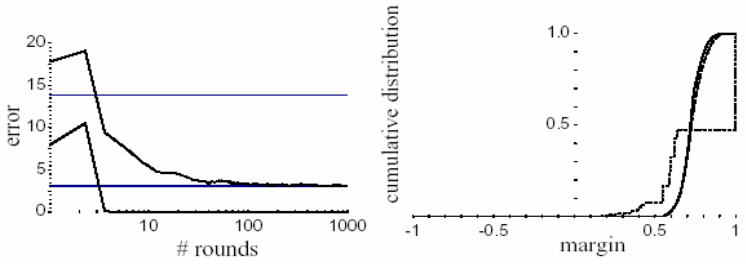


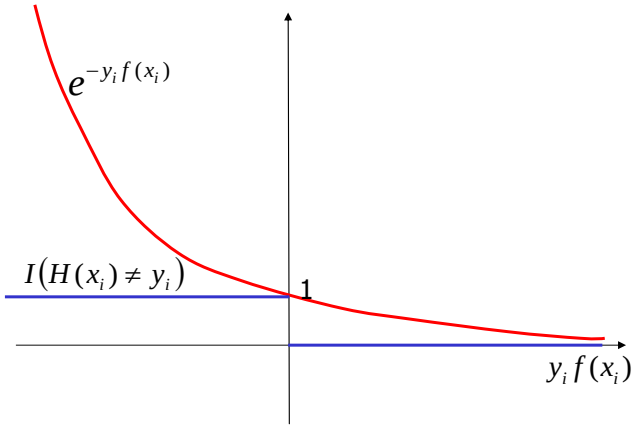
Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [69]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. *Right*: The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

From Schapire (2001)

AdaBoost and Margins

- Recall Fact 2 from our earlier analysis:

$$I\left(H(x_i) \neq y_i\right) \leq e^{-y_i f(x_i)}$$



AdaBoost Applies Exponential Penalties

- Think of each curve as representing a penalty to be applied to any training example having that value of $y_i f(x_i)$
- Applying a penalty based on the value of

$$I\left(H(x_i) \neq y_i\right)$$

would simply apply a penalty of 1 to any misclassified example and a penalty of 0 to any correctly classified example

- But AdaBoost instead applies penalties based on the value of

$$e^{-y_i f(x_i)}$$

AdaBoost Penalties and Margins

- That is, AdaBoost penalizes
  - badly misclassified examples extremely harshly
  - slightly misclassified examples less
  - even examples classified barely correctly
- AdaBoost tries to drive all the margins to 1, continually trying to increase them, regardless of whether they're > 0 or not
- This is why AdaBoost may continue to create a better-generalizing classifier long after the training error is 0

Any Downsides to AdaBoost?

- In spite of this theory, still may overfit
- May be overly sensitive to outliers
  - Concentrates more and more weight on difficult-to-classify points
  - Can be alleviated
    - Integrate outlier detection mechanisms
    - Redesign around less outlier-sensitive cost functions

What You Should Know

- That there are a variety of ensemble learning approaches
  - Mixture of experts
  - Bagging
  - Boosting
- How bagging is done
- Weak vs. strong learners
- The AdaBoost algorithm
  - Detailed steps
  - Analysis of how it reduces training set error
  - Why it may reduce true error long after training set error is zero
  - Notion of margin of an ensemble classifier and how it plays a role in promoting good generalization