HW4 is due on Thursday 10/22
- No extensions!
- We will release solutions that night, so there is enough time for you to look at it before the exam.

Midterm exam on Tuesday 10/27
- Closed books; in class; ~4 questions
- All the material covered before the midterm
- Practice midterms will be released tomorrow

Project Proposals (≤ 2 pages)
- Due tomorrow, 10/16
- Submit via Compass
Where are we?

- **Algorithms**
  - DTs
  - Perceptron + Winnow
  - Gradient Descent
  - NN

- **Theory**
  - Mistake Bound
  - PAC Learning

We have a formal notion of “learnability”

- We understand Generalization
  - How will your algorithm do on the next example?
- How it depends on the hypothesis class (VC dim)
  - and other complexity parameters

- Algorithmic Implications of the theory?
Boosting

- Boosting is (today) a general learning paradigm for putting together a Strong Learner, given a collection (possibly infinite) of Weak Learners.

- The original Boosting Algorithm was proposed as an answer to a theoretical question in PAC learning. [The Strength of Weak Learnability; Schapire, 89]

- Consequently, Boosting has interesting theoretical implications, e.g., on the relations between PAC learnability and compression.
  - If a concept class is efficiently PAC learnable then it is efficiently PAC learnable by an algorithm whose required memory is bounded by a polynomial in \( n \), size \( c \) and \( \log(1/\epsilon) \).
  - There is no concept class for which efficient PAC learnability requires that the entire sample be contained in memory at one time – there is always another algorithm that “forgets” most of the sample.
However, the key contribution of Boosting has been practical, as a way to compose a good learner from many weak learners.

It is a member of a family of Ensemble Algorithms, but has stronger guarantees than others.

A Boosting demo is available at [http://cseweb.ucsd.edu/~yfreund/adaboost/](http://cseweb.ucsd.edu/~yfreund/adaboost/)

Example

Theory of Boosting
  - Simple & insightful
Boosting Motivation

Example: “How May I Help You?”

[ Gorin et al. ]

- **goal**: automatically categorize type of call requested by phone customer
  
  (Collect, CallingCard, PersonToPerson, etc.)

  - yes I’d like to place a collect call long distance please (Collect)
  - operator I need to make a call but I need to bill it to my office (ThirdNumber)
  - yes I’d like to place a call on my master card please (CallingCard)
  - I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill (BillingCredit)

- **observation**:
  
  - easy to find “rules of thumb” that are “often” correct
    
    - e.g.: “IF ‘card’ occurs in utterance THEN predict ‘CallingCard’ ”
  
  - hard to find single highly accurate prediction rule
The Boosting Approach

**Algorithm**
- Select a small subset of examples
- Derive a rough rule of thumb
- Examine 2nd set of examples
- Derive 2nd rule of thumb
- Repeat T times
- Combine the learned rules into a single hypothesis

**Questions:**
- How to choose subsets of examples to examine on each round?
- How to combine all the rules of thumb into a single prediction rule?

**Boosting**
- General method of converting rough rules of thumb into highly accurate prediction rule
Theoretical Motivation

“Strong” PAC algorithm:
- for any distribution
- $\forall \epsilon, \delta > 0$
- Given polynomially many random examples
- Finds hypothesis with error $\leq \epsilon$ with probability $\geq (1-\delta)$

“Weak” PAC algorithm
- Same, but only for some $\epsilon \leq \frac{1}{2} - \gamma$

[Kearns & Valiant ’88]:
- Does weak learnability imply strong learnability?
- Anecdote: the importance of the distribution free assumption
  - It does not hold if PAC is restricted to only the uniform distribution, say
History

[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

[Freund & Schapire ’95]:
- Introduced “AdaBoost” algorithm
- Strong practical advantages over previous boosting algorithms

AdaBoost was followed by a huge number of papers and practical applications
A Formal View of Boosting

- Given **training set** \((x_1, y_1), \ldots, (x_m, y_m)\)
- \(y_i \in \{-1, +1\}\) is the correct label of instance \(x_i \in X\)
- For \(t = 1, \ldots, T\)
  - Construct a **distribution** \(D_t\) on \([1, \ldots, m]\)
  - Find **weak hypothesis** (“rule of thumb”)
    \[ h_t : X \rightarrow \{-1, +1\} \]
    with small error \(\epsilon_t\) on \(D_t\):
    \[ \epsilon_t = \Pr_{D_t} [h_t (x_i) \neq y_i] \]
- **Output**: **final hypothesis** \(H_{\text{final}}\)
Adaboost

Constructing $D_t$ on {1,...m}:

- $D_1(i) = 1/m$
- Given $D_t$ and $h_t$:
  - $D_{t+1} = D_t(i)/z_t \times e^{-\alpha_t}$ if $y_i = h_t(x_i)$
  - $D_{t+1} = D_t(i)/z_t \times e^{+\alpha_t}$ if $y_i \neq h_t(x_i)$

where $z_t = \text{normalization constant}$

and

$\alpha_t = \frac{1}{2} \ln \{ (1 - \epsilon_t)/\epsilon_t \}$

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

Notes about $\alpha_t$:

- Positive due to the weak learning assumption
- Examples that we predicted correctly are demoted, others promoted
- Sensible weighting scheme: better hypothesis (smaller error) $\rightarrow$ larger weight

Think about unwrapping it all the way to $1/m$
A Toy Example

\[ D_1 \]
A Toy Example

Round 1

$\hat{h}_1$

$\mathcal{D}_2$

$\epsilon_1 = 0.30$

$\alpha_1 = 0.42$
Boosting

Round 2

$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

$D_3$

$H_2$
Boosting
A cool and important note about the final hypothesis: it is possible that the combined hypothesis makes no mistakes on the training data, but boosting can still learn, by adding more weak hypotheses.
Analyzing Adaboost

- **Theorem:**
  - run AdaBoost
  - let $\epsilon_t = 1/2 - \gamma_t$
  - then

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

1. Why is the theorem stated in terms of minimizing training error? Is that what we want?
2. What does the bound mean?

\[
\epsilon_t (1- \epsilon_t) = (1/2-\gamma_t)(1/2+\gamma_t)) = 1/4 - \gamma_t^2 \\
1-(2\gamma_t)^2 \leq \exp(-(2\gamma_t)^2)
\]

- so: if $\forall t: \gamma_t \geq \gamma > 0$
  then training error$(H_{\text{final}}) \leq e^{-2\gamma^2 T}$

- **adaptive:**
  - does not need to know $\gamma$ or $T$ a priori
  - can exploit $\gamma_t \gg \gamma$
Need to prove only the first inequality, the rest is algebra.

**AdaBoost Proof (1)**

- let \( f(x) = \sum_t \alpha_t h_t(x) \Rightarrow H_{\text{final}}(x) = \text{sign}(f(x)) \)

**Step 1: unwrapping recursion:**

\[
D_{\text{final}}(i) = \frac{1}{m} \cdot \frac{\exp \left( -y_i \sum_t \alpha_t h_t(x_i) \right)}{\Pi_t Z_t}
\]

\[
= \frac{1}{m} \cdot \frac{e^{-y_i f(x_i)}}{\Pi_t Z_t}
\]

The final “weight” of the i-th example
AdaBoost Proof (2)

- Step 2: \( \text{training error}(H_{\text{final}}) \leq \prod_t Z_t \)

- Proof:
  - \( H_{\text{final}}(x) \neq y \Rightarrow yf(x) \leq 0 \Rightarrow e^{-yf(x)} \geq 1 \)
  - so:
    \[
    \text{training error}(H_{\text{final}}) = \frac{1}{m} \sum_i \left\{ \begin{array}{ll}
    1 & \text{if } y_i \neq H_{\text{final}}(x_i) \\
    0 & \text{else}
    \end{array} \right.
    \leq \frac{1}{m} \sum_i e^{-y_i f(x_i)}
    \leq \sum_i D_{\text{final}}(i) \prod_t Z_t
    = \prod_t Z_t
    \]

**The definition of training error**

Always holds for mistakes (see above)

Using Step 1

D is a distribution over the m examples
AdaBoost Proof(3)

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

\[
Z_t = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i))
\]

\[
= \sum_{i:y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} + \sum_{i:y_i = h_t(x_i)} D_t(i) e^{-\alpha_t}
\]

\[
= \epsilon_t e^{\alpha_t} + (1 - \epsilon_t) e^{-\alpha_t}
\]

\[
= 2\sqrt{\epsilon_t (1 - \epsilon_t)}
\]

By definition of \( Z_t \); it’s a normalization term

Steps 2 and 3 together prove the Theorem.
→ The error of the final hypothesis as be as low as you want.

Splitting the sum to “mistakes” and no-mistakes
The definition of \( \epsilon_t \)
The definition of \( \alpha_t \)
Boosting The Confidence

- Unlike Boosting the accuracy, Boosting the confidence is easy.

- Let’s fix the accuracy parameter to $\varepsilon$.

- Suppose that we have a learning algorithm $L$ such that for any target concept $c \in C$ and any distribution $D$, $L$ outputs $h$ s.t. $\text{error}(h) < \varepsilon$ with confidence at least $\delta_0 = 1/q(n,\text{size}(c))$, for some polynomial $q$.

- Then, if we are willing to tolerate a slightly higher hypothesis error, $\varepsilon + \gamma$ ($\gamma > 0$, arbitrarily small) then we can achieve arbitrary high confidence $1-\delta$. 
Boosting The Confidence(2)

**Idea:** Given the algorithm L, we construct a new algorithm L’ that simulates algorithm L k times (k will be determined later) on independent samples from the same distribution.

Let \( h_1, \ldots, h_k \) be the hypotheses produced. Then, since the simulations are independent, the probability that all of \( h_1, \ldots, h_k \) have error \( \geq \varepsilon \) is at most \((1-\delta_0)^k\). Otherwise, at least one \( h_j \) is good.

Solving \((1-\delta_0)^k < \frac{\delta}{2}\) yields that value of k we need,

\[
k > \left( \frac{1}{\delta_0} \right) \ln \left( \frac{2}{\delta} \right)
\]

There is still a need to show how L’ works. It would work by using the \( h_i \) that makes the fewest mistakes on the sample S; we need to compute how large S should be to guarantee that it does not make too many mistakes.  

[Kearns and Vazirani’s book]
Summary of Ensemble Methods

- Boosting
- Bagging
- Random Forests
Boosting

- **Initialization:**
  - Weigh all training samples equally

- **Iteration Step:**
  - Train model on (weighted) train set
  - Compute error of model on train set
  - Increase weights on training cases model gets wrong!!!

- Typically requires 100’s to 1000’s of iterations

- Return final model:
  - Carefully weighted prediction of each model
Boosting: Different Perspectives

- Boosting is a maximum-margin method (Schapire et al. 1998, Rosset et al. 2004)
  - Trades lower margin on easy cases for higher margin on harder cases

- Boosting is an additive logistic regression model (Friedman, Hastie and Tibshirani 2000)
  - Tries to fit the logit of the true conditional probabilities

- Boosting is an equalizer (Breiman 1998) (Friedman, Hastie, Tibshirani 2000)
  - Weighted proportion of times example is misclassified by base learners tends to be the same for all training cases

- Boosting is a linear classifier, but does not give well calibrated probability estimate.
Bagging

- Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor.

- The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class.

- The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets.
  - That is, use samples of the data, with repetition

- Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy.

- The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed then bagging can improve accuracy.
Example: Bagged Decision Trees

- Draw 100 bootstrap samples of data
- Train trees on each sample → 100 trees
- Average prediction of trees on out-of-bag samples

Average prediction

\[
\frac{0.23 + 0.19 + 0.34 + 0.22 + 0.26 + \ldots + 0.31}{\# \text{ Trees}} = 0.24
\]
Random Forests (Bagged Trees++)

- Draw **1000** bootstrap samples of data
- *Draw sample of available attributes at each split*
- Train trees on each sample/attribute set $\rightarrow$ **1000** trees
- Average prediction of trees on out-of-bag samples

\[
\text{Average prediction} = \frac{0.23 + 0.19 + 0.34 + 0.22 + 0.26 + \ldots + 0.31}{\# \text{ Trees}} = 0.24
\]