Today

- New Machine Learning Topics:
  - Ensemble Learning
    - Bagging
    - Boosting
  - Next time two papers:
    - “Rapid Object Detection using a Boosted Cascade of Simple Features” by P. Viola and M. Jones from CVPR2001
    - “Detecting Pedestrians Using Patterns of Motion and Appearance” by P. Viola, M.J.Jones, D. Snow

Ensemble Learning: Bagging and Boosting

- So far we have talked about design of a single classifier that generalizes well (want to “learn” f(x) )
- From statistics, we know that it is good to average your predictions (reduces variance)
- Bagging
  - reshuffle your training data to create k different training sets and learn f_1(x), f_2(x), ..., f_k(x)
  - Combine the k different classifiers by majority voting
    \[ f_{\text{FINAL}}(x) = \text{sign}\left(\sum 1/k f_i(x)\right) \]
- Boosting
  - Assign different weights to training samples in a “smart” way so that different classifiers pay more attention to different samples
  - Weighted majority voting, the weight of individual classifier is proportional to its accuracy
  - Ada-boost (1996) was influenced by bagging, and it is superior to bagging

Bagging

- Generate a random sample from training set by selecting l elements (out of n elements available) with replacement
- each classifier is trained on the average of 63.2% of the training examples
  - For a dataset with N examples, each example has a probability of \(1-(1-1/N)^N\) of being selected at least once in the N samples. For \(N \rightarrow \infty\), this number converges to \((1-1/e)\) or 0.632 [Bauer and Kohavi, 1999]
- Repeat the sampling procedure, getting a sequence of k independent training sets
- A corresponding sequence of classifiers f_1(x), f_2(x), ..., f_k(x) is constructed for each of these training sets, using the same classification algorithm
- To classify an unknown sample x, let each classifier predict.
- The bagged classifier \(f_{\text{FINAL}}(x)\) then combines the predictions of the individual classifiers to generate the final outcome, frequently this combination is simple voting
### Boosting: motivation

- It is usually hard to design an accurate classifier which generalizes well
- However it is usually easy to find many “rule of thumb” weak classifiers
  - A classifier is weak if it is only slightly better than random guessing
- Can we combine several weak classifiers to produce an accurate classifier?
  - Question people have been working on since 1980’s

### Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially distribution of weights is uniform
- At successive iterations, the weight of misclassified examples is increased, forcing the weak learner to focus on the hard examples in the training set

### Ada Boost

- Let’s assume we have 2-class classification problem, with $y \in \{-1, 1\}$
- Ada boost will produce a discriminant function:
  \[
  g(x) = \sum_{t=1}^{T} \alpha_t f_t(x)
  \]
- where $f_t(x)$ is the “weak” classifier
- As usual, the final classifier is the sign of the discriminant function, that is $f_{final}(x) = sign(g(x))$

### More Comments on Ada Boost

- Ada boost is very simple to implement, provided you have an implementation of a “weak learner”
- Will work as long as the “basic” classifier $f_t(x)$ is at least slightly better than random
  - will work if the error rate of $f_t(x)$ is less than 0.5 (0.5 is the error rate of a random guessing classifier for a 2-class problem)
- Can be applied to boost any classifier, not necessarily weak
### Ada Boost

**Ada Boost (slightly modified from the original version)**

- \( d(x) \) is the distribution of weights over the \( N \) training points \( \sum d(x_i) = 1 \)
- Initially assign uniform weights \( d_i(x) = 1/N \) for all \( x_i \)
- At each iteration \( t \):
  - Find best weak classifier \( f_t(x) \) using weights \( d_t(x) \)
  - Compute the error rate \( \epsilon_t \) as
    \[
    \epsilon_t = \sum_{i=1}^{N} d_t(x_i) \cdot 1[y_i \neq f_t(x_i)]
    \]
  - Assign weight \( \alpha_t \) to the classifier \( f_t(x) \)'s in the final hypothesis
    \[
    \alpha_t = \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
    \]
  - For each \( x_i \), \( d_{t+1}(x_i) = d_t(x) \cdot \exp[\alpha_t \cdot 1(y_i \neq f_t(x_i))] \)
  - Normalize \( d_{t+1}(x_i) \) so that \( \sum_i d_{t+1}(x_i) = 1 \)
- \( f_{\text{FINAL}}(x) = \text{sign} \left[ \sum \alpha_t f_t(x) \right] \)

**Ada Boost**

- At each iteration \( t \):
  - Find best weak classifier \( f_t(x) \) using weights \( d_t(x) \)
  - Compute \( \epsilon_t \) the error rate as
    \[
    \epsilon_t = \sum d_t(x) \cdot 1[y_i \neq f_t(x_i)]
    \]
  - Assign weight \( \alpha_t \) to the classifier \( f_t(x) \)'s in the final hypothesis
    \[
    \alpha_t = \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
    \]
  - For each \( x_i \), \( d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot 1(y_i \neq f_t(x_i))] \)
  - Normalize \( d_{t+1}(x_i) \) so that \( \sum_i d_{t+1}(x_i) = 1 \)
  - \( f_{\text{FINAL}}(x) = \text{sign} \left[ \sum \alpha_t f_t(x) \right] \)

Since the weak classifier is better than random, we expect \( \epsilon_t < 1/2 \)

### Ada Boost

**Ada Boost**

- At each iteration \( t \):
  - Find best weak classifier \( f_t(x) \) using weights \( d_t(x) \)
  - Compute \( \epsilon_t \) the error rate as
    \[
    \epsilon_t = \sum_{i=1}^{N} d_t(x_i) \cdot 1[y_i \neq f_t(x_i)]
    \]
  - Assign weight \( \alpha_t \) to the classifier \( f_t(x) \)'s in the final hypothesis
    \[
    \alpha_t = \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
    \]
  - For each \( x_i \), \( d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot 1(y_i \neq f_t(x_i))] \)
  - Normalize \( d_{t+1}(x_i) \) so that \( \sum_i d_{t+1}(x_i) = 1 \)
  - \( f_{\text{FINAL}}(x) = \text{sign} \left[ \sum \alpha_t f_t(x) \right] \)

Recall that \( \epsilon_t < 1/2 \)

Thus \( (1 - \epsilon_t)/\epsilon_t > 1 \Rightarrow \alpha_t > 0 \)

The smaller is \( \epsilon_t \), the larger is \( \alpha_t \) and thus the more importance (weight) classifier \( f_t(x) \) gets in the final classifier \( f_{\text{FINAL}}(x) = \text{sign} \left[ \sum \alpha_t f_t(x) \right] \)
Ada Boost

- At each iteration t:
  - Find best weak classifier \( f_t(x) \) using weights \( d_t(x) \)
  - Compute \( \varepsilon_t \), the error rate as
    \[
    \varepsilon_t = \sum d_t(x_i) \cdot I(y_i \neq f_t(x_i))
    \]
  - Assign weight \( u_t \) the classifier \( f_t \) is in the final hypothesis
    \[
    u_t = \log \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)
    \]
  - For each \( x_i \), \( d_{t+1}(x_i) = d_t(x_i) \cdot \exp[\alpha_t \cdot I(y_i \neq f_t(x_i))] \)
  - Normalize \( d_{t+1}(x) \) so that \( \sum d_{t+1}(x) = 1 \)
  - Weight of misclassified examples is increased and the new \( d_{t+1}(x) \)'s are normalized to be a distribution again

AdaBoost Example

ROUND 1

ROUND 2

Note: in the following slides, \( h_t(x) \) is used instead of \( f_t(x) \), and \( D \) instead of \( d \)
**AdaBoost Example**

**ROUND 3**

- $\alpha_2 = 0.14$
- $\alpha_3 = 0.02$

**AdaBoost Comments**

- It can be shown that the training error drops exponentially fast, if each weak classifier is slightly better than random

$$Err_{train} \leq \exp(-2\sum \gamma_t)$$

- Here $\gamma_t = \epsilon_t - 1/2$, where $\epsilon_t$ is classification error at round $t$ (weak classifier $f_t$)

**AdaBoost Example**

- $f_{FINAL}(x)$

**AdaBoost Comments**

- But we are really interested in the generalization properties of $f_{FINAL}(x)$, not the training error
- AdaBoost was shown to have excellent generalization properties in practice
  - the more rounds, the more complex is the final classifier, so overfitting is expected as the training proceeds
  - but in the beginning researchers observed no overfitting of the data
  - It turns out it does overfit data eventually, if you run it really long
- It can be shown that boosting “aggressively” increases the margins of training examples, as iterations proceed
  - margins continue to increase even when training error reaches zero
  - Helps to explain empirically observed phenomena: test error continues to drop even after training error reaches zero
### AdaBoost Example

- The final prediction in boosting $g(x)$ can be expressed as an additive expansion of individual classifiers:
  \[
g(x) = \sum_{k=1}^{M} \alpha_k f_k(x; \gamma_k)
\]
- Typically we would try to minimize a loss function on the $N$ training examples:
  \[
  \min_{\alpha, \gamma_1, \ldots, \gamma_M} \sum_{i=1}^{N} L(y_i, \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k))
\]
- For example, under squared-error loss:
  \[
  \min_{\alpha, \gamma_1, \ldots, \gamma_M} \sum_{i=1}^{N} (y_i - \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k))^2
\]

### The Margin Distribution

<table>
<thead>
<tr>
<th>epoch</th>
<th>5</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>training error</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>test error</td>
<td>8.4</td>
<td>3.3</td>
<td>3.1</td>
</tr>
<tr>
<td>%margins&lt;0.5</td>
<td>7.7</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Minimum margin</td>
<td>0.14</td>
<td>0.52</td>
<td>0.55</td>
</tr>
</tbody>
</table>

### Boosting As Additive Model

- Forward stage-wise modeling is iterative and fits the $f_k(x, \gamma)$ sequentially, fixing the results of previous iterations:
  \[
g_t(x) = g_{t-1}(x) + \alpha_t f_t(x; \gamma_t)
\]
- Under the squared difference loss function:
  \[
  L(y_i, g_t(x_i) + \alpha_t f_t(x_i; \gamma_t)) =
  (y_i - g_{t-1}(x_i) - \alpha_t f_t(x_i; \gamma_t))^2
\]
- Forward stage-wise optimization seems to produce classifier with better generalization, doing the process stagewise seems to overfit less quickly.
**Boosting As Additive Model**

\[ g(x) = \sum_{k=1}^{n} \alpha_k f_k(x; \gamma_k) \]

- It can be shown that AdaBoost uses forward stage-wise modeling under the following loss function:
  - \( L(y, g(x)) = \exp(-y \cdot g(x)) \) -- the exponential loss function
- At stage (or iteration) \( m \), we fit:
  \[
  \sum_{i=1}^{N} \exp(-y_i \cdot \{g_{m+1}(x_i) + \alpha_m \cdot f_m(x_i)\})
  \]
  \[
  = \sum_{i=1}^{N} \exp(-y_i \cdot \{g_{m+1}(x_i) + \alpha_m \cdot f_m(x_i)\})
  \]

**Logistic Regression Model**

- It can be shown that Adaboost builds a logistic regression model:
  \[
  g(x) = \alpha \sum_{i=1}^{N} \frac{\Pr(Y = 1 | x)}{\Pr(Y = -1 | x)} x \alpha f_i(x)
  \]
- It can also be shown that the the training error on the samples is at most:
  \[
  \sum_{i=1}^{N} \exp(-y_i \cdot g(x_i)) \leq \sum_{i=1}^{N} \exp(-y_i \cdot \sum_{i=1}^{N} \alpha_i f_i(x_i))
  \]

**Exponential Loss vs. Squared Error Loss**

- \( L(y, g(x)) = \exp(-y \cdot g(x)) \)
- \( L(y, g(x)) = (y - g(x))^2 \)

- Squared Error Loss penalizes classifications that are "too correct", with \( y \cdot g(x) > 1 \), and thus it is inappropriate for classification
- Exponential loss encourages large margins, want \( y \cdot g(x) \) large

**Practical Advantages of AdaBoost**

- fast
- simple
- Has only one parameter to tune (\( T \))
- flexible: can be combined with any classifier
- provably effective (assuming weak learner)
  - shift in mind set: goal now is merely to find hypotheses that are better than random guessing
- finds outliers
  - The hardest examples are frequently the "outliers"
Caveats

- performance depends on data & weak learner
- AdaBoost can fail if
  - weak hypothesis too complex (overfitting)
  - weak hypothesis too weak ($\gamma_t \to 0$ too quickly),
    - underfitting
    - Low margins $\to$ overfitting
- empirically, AdaBoost seems especially susceptible to noise