Today

- New Machine Learning Topics:
  1) Performance evaluation methods
     - cross-validation
  2) 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
Linear Regression

Univariate Linear regression with a constant term:

\[
\begin{align*}
X & \quad Y \\
3 & \quad 7 \\
1 & \quad 3 \\
\vdots & \\
\end{align*}
\]

\[x_1 = (3, \ldots, y_1 = 7, \ldots)\]

\[
\begin{align*}
X & \quad Y \\
3 & \quad 7 \\
1 & \quad 3 \\
\vdots & \\
\end{align*}
\]

\[x_1 = (3, \ldots, y_1 = 7, \ldots)\]

\[
\beta = (Z^T Z)^{-1} (Z^T y)
\]

\[y_{\text{est}} = \beta_0 + \beta_1 x\]

from Andrew Moore (CMU)

Quadratic Regression

from Andrew Moore (CMU)
**Quadratic Regression**

\[ X | Y \\
3 | 7 \\
1 | 3 \\
\]

\[ Z = \\
1 | 3 | 9 \\
1 | 1 | 1 \\
\]

\[ y = \\
7 \\
3 \\
\]

\[ y_{est} = \beta_0 + \beta_1 x + \beta_2 x^2 \]

**Join-the-dots**

Also known as piecewise linear nonparametric regression if that makes you feel better.

**Which is best?**

Why not choose the method with the best fit to the data?

**What do we really want?**

Why not choose the method with the best fit to the data?

“How well are you going to predict future data drawn from the same distribution?”
1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

*Linear regression example*
Mean Squared Error = 2.4

*Quadratic regression example*
Mean Squared Error = 0.9

From Andrew Moore (CMU)
The test set method

1. Randomly choose 30% of the data to be in a test set
2. The remainder is a training set
3. Perform your regression on the training set
4. Estimate your future performance with the test set

(Join the dots example)
Mean Squared Error = 2.2

Good news:
- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:
- Wastes data: we get an estimate of the best method to apply to 30% less data
  - if we don’t have much data, our test-set might just be lucky or unlucky

LOOCV (Leave-one-out Cross Validation)

For k=1 to R
1. Let \((x_k, y_k)\) be the \(k^{th}\) record

\[
\begin{align*}
\text{y} & \quad \text{x} \\
\bullet & \quad \bullet & \quad \bullet \\
\bullet & \quad \bullet & \quad \bullet \\
\end{align*}
\]
**LOOCV (Leave-one-out Cross Validation)**

For $k=1$ to $R$
1. Let $(x_k,y_k)$ be the $k$th record
2. Temporarily remove $(x_k,y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k,y_k)$

When you've done all points, report the mean error.

---

**LOOCV (Leave-one-out Cross Validation)**

For $k=1$ to $R$
1. Let $(x_k,y_k)$ be the $k$th record
2. Temporarily remove $(x_k,y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k,y_k)$

---

**LOOCV (Leave-one-out Cross Validation)**

For $k=1$ to $R$
1. Let $(x_k,y_k)$ be the $k$th record
2. Temporarily remove $(x_k,y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k,y_k)$

---

**LOOCV (Leave-one-out Cross Validation)**

For $k=1$ to $R$
1. Let $(x_k,y_k)$ be the $k$th record
2. Temporarily remove $(x_k,y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k,y_k)$

When you've done all points, report the mean error.
**LOOCV (Leave-one-out Cross Validation)**

For $k=1$ to $R$
1. Let $(x_k, y_k)$ be the $k$th record
2. Temporarily remove $(x_k, y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k, y_k)$

When you've done all points, report the mean error

$\text{MSE}_{\text{LOOCV}} = 2.12$

from Andrew Moore (CMU)

**LOOCV for Join The Dots**

For $k=1$ to $R$
1. Let $(x_k, y_k)$ be the $k$th record
2. Temporarily remove $(x_k, y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k, y_k)$

When you've done all points, report the mean error

$\text{MSE}_{\text{LOOCV}} = 3.33$

from Andrew Moore (CMU)

**LOOCV for Quadratic Regression**

For $k=1$ to $R$
1. Let $(x_k, y_k)$ be the $k$th record
2. Temporarily remove $(x_k, y_k)$ from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error $(x_k, y_k)$

When you've done all points, report the mean error

$\text{MSE}_{\text{LOOCV}} = 0.962$

from Andrew Moore (CMU)

**Which kind of Cross Validation?**

<table>
<thead>
<tr>
<th>Test-set</th>
<th>Downside</th>
<th>Upside</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leave-one-out</td>
<td>Expensive</td>
<td>Doesn't waste data</td>
</tr>
<tr>
<td>LOOCV</td>
<td>Variance: unreliable estimate of future performance</td>
<td>Cheap</td>
</tr>
</tbody>
</table>

..can we get the best of both worlds?
Randomly break the dataset into $k$ partitions (in our example we'll have $k=3$ partitions colored Red, Green, and Blue).

**For the red partition:** Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

**For the green partition:** Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

**For the blue partition:** Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

**For the yellow partition:** Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

$k$-fold Cross Validation from Andrew Moore (CMU)
Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Yellow)

For the red partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the yellow partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

Then report the mean error

\[ \text{MSE}_{3\text{FOLD}} = 2.05 \]

from Andrew Moore (CMU)

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Yellow Green and Blue)

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the gray partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

Then report the mean error

\[ \text{MSE}_{3\text{FOLD}} = 1.11 \]

from Andrew Moore (CMU)

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Yellow Green and Blue)

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the gray partition: Train on all the points not in the gray partition. Find the test-set sum of errors on the gray points.

Then report the mean error

\[ \text{MSE}_{3\text{FOLD}} = 2.93 \]

from Andrew Moore (CMU)

<table>
<thead>
<tr>
<th>Which kind of Cross Validation?</th>
<th>Downside</th>
<th>Upside</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test-set</td>
<td>Variance: unreliable estimate of future performance</td>
<td>Cheap</td>
</tr>
<tr>
<td>Leave-one-out</td>
<td>Expensive</td>
<td>Doesn’t waste data</td>
</tr>
<tr>
<td>10-fold</td>
<td>Wastes 10% of the data. Only wastes 10%. Only 10 times more expensive instead of R times.</td>
<td></td>
</tr>
<tr>
<td>3-fold</td>
<td>Wastier than 10-fold. Slightly better than test-set</td>
<td></td>
</tr>
<tr>
<td>N-fold</td>
<td>Identical to Leave-one-out</td>
<td></td>
</tr>
</tbody>
</table>

from Andrew Moore (CMU)
We’re trying to decide which algorithm to use.
- We train each machine and make a table...

### Example: Choosing number of hidden units in a one-hidden-layer neural net.

#### Step 1: Compute 10-fold CV error for six different model classes:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10-FOLD-CV-ERR</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 hidden units</td>
<td>[f_1]</td>
<td>[f_2]</td>
</tr>
<tr>
<td>1 hidden units</td>
<td>[f_3]</td>
<td>[f_4]</td>
</tr>
<tr>
<td>2 hidden units</td>
<td>[f_5]</td>
<td>[f_6]</td>
</tr>
<tr>
<td>3 hidden units</td>
<td>[f_7]</td>
<td>[f_8]</td>
</tr>
<tr>
<td>4 hidden units</td>
<td>[f_9]</td>
<td>[f_10]</td>
</tr>
<tr>
<td>5 hidden units</td>
<td>[f_11]</td>
<td>[f_12]</td>
</tr>
</tbody>
</table>

#### Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you’ll use.

---

### Example: Choosing “k” for a k-nearest-neighbor regression.

#### Step 1: Compute LOOCV error for six different model classes:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10-fold-CV-ERR</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=1</td>
<td>[f_1]</td>
<td>[f_2]</td>
</tr>
<tr>
<td>K=2</td>
<td>[f_3]</td>
<td>[f_4]</td>
</tr>
<tr>
<td>K=3</td>
<td>[f_5]</td>
<td>[f_6]</td>
</tr>
<tr>
<td>K=4</td>
<td>[f_7]</td>
<td>[f_8]</td>
</tr>
<tr>
<td>K=5</td>
<td>[f_9]</td>
<td>[f_10]</td>
</tr>
<tr>
<td>K=6</td>
<td>[f_11]</td>
<td>[f_12]</td>
</tr>
</tbody>
</table>

#### Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you’ll use.

---

CV-based Model Selection

- We’re trying to decide which algorithm to use.
- We train each machine and make a table...

### Example: Choosing number of hidden units in a one-hidden-layer neural net.

#### Step 1: Compute 10-fold CV error for six different model classes:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10-FOLD-CV-ERR</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 hidden units</td>
<td>[f_1]</td>
<td>[f_2]</td>
</tr>
<tr>
<td>1 hidden units</td>
<td>[f_3]</td>
<td>[f_4]</td>
</tr>
<tr>
<td>2 hidden units</td>
<td>[f_5]</td>
<td>[f_6]</td>
</tr>
<tr>
<td>3 hidden units</td>
<td>[f_7]</td>
<td>[f_8]</td>
</tr>
<tr>
<td>4 hidden units</td>
<td>[f_9]</td>
<td>[f_10]</td>
</tr>
<tr>
<td>5 hidden units</td>
<td>[f_11]</td>
<td>[f_12]</td>
</tr>
</tbody>
</table>

#### Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you’ll use.

---

CV-based Model Selection

- Example: Choosing “k” for a k-nearest-neighbor regression.

#### Step 1: Compute LOOCV error for six different model classes:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>10-fold-CV-ERR</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=1</td>
<td>[f_1]</td>
<td>[f_2]</td>
</tr>
<tr>
<td>K=2</td>
<td>[f_3]</td>
<td>[f_4]</td>
</tr>
<tr>
<td>K=3</td>
<td>[f_5]</td>
<td>[f_6]</td>
</tr>
<tr>
<td>K=4</td>
<td>[f_7]</td>
<td>[f_8]</td>
</tr>
<tr>
<td>K=5</td>
<td>[f_9]</td>
<td>[f_10]</td>
</tr>
<tr>
<td>K=6</td>
<td>[f_11]</td>
<td>[f_12]</td>
</tr>
</tbody>
</table>

#### Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you’ll use.
CV-based Model Selection

- Can you think of other decisions we can ask Cross Validation to make for us, based on other machine learning algorithms in the class so far?

CV-based Algorithm Choice

- Example: Choosing which regression algorithm to use
- Step 1: Compute 10-fold-CV error for six different model classes:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TRAINErr</th>
<th>10-fold-CV-Err</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Reg'n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quad reg'n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LWR, KW=0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LWR, KW=0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Step 2: Whichever algorithm gave best CV score: train it with all the data, and that’s the predictive model you’ll use.

Cross-validation for classification

- Instead of computing the sum squared errors on a test set, you should compute…

Cross-validation for classification

- Instead of computing the sum squared errors on a test set, you should compute…

The total number of misclassifications on a testset.
Cross-validation for classification

- Instead of computing the sum squared errors on a test set, you should compute the total number of misclassifications on a test set.
  - What's LOOCV of 1-NN?
  - What's LOOCV of 3-NN?
  - What's LOOCV of 22-NN?

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 \( f_1(x), f_2(x), \ldots, f_k(x) \).
  - Combine the \( k \) different classifiers by majority voting
    \[ f_{\text{FINAL}}(x) = \text{sign} \left( \sum f_i(x) \right) \]
- 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.

Cross-Validation for classification

- Choosing \( k \) for \( k \)-nearest neighbors
- Choosing \( h \) for the Parzen windows
- Any other "free" parameter of a classifier
- Choosing which classifier to use
- Choosing Features to use

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 \( N \to \infty \), 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 \( f_1(x), f_2(x), \ldots, f_k(x) \) 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 $f_{\text{final}}(x) = \text{sign}[g(x)]$

### 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 (slightly modified from the original version)

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

---

Ada Boost

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

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

---

**Ada Boost**

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

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 
    \[ \varepsilon_t = \sum d_t(x) \cdot I(y_i \neq f_t(x)) \]
  - Assign weight \( \omega_i \) to the classifier \( f_t \) in the final hypothesis 
    \[ \omega_i = \log \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) \]
  - 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

From "A Tutorial on Boosting" by Yoav Freund and Rob Schapire

Round 1

Original Training set: equal weights to all training samples

Note: in the following slides, \( h_t(x) \) is used instead of \( f_t(x) \), and \( D \) instead of \( d \)
It can be shown that the training error drops exponentially fast, if each weak classifier is slightly better than random
\[ \text{Err}_{\text{train}} \leq \exp\left(-2\sum \gamma_t \right) \]
Here \( \gamma_t = \epsilon_t - 1/2 \), where \( \epsilon_t \) is classification error at round \( t \) (weak classifier \( f_t \))

But we are really interested in the generalization properties of \( f_{\text{FINAL}}(x) \), 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
**Boosting As Additive Model**

- 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
  \[ \min_{\alpha_1, \gamma_1, \ldots, \alpha_M, \gamma_M} \sum_{i=1}^{N} \lambda_i \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k) \]

- For example, under squared-error loss:
  \[ \min_{\alpha_1, \gamma_1, \ldots, \alpha_M, \gamma_M} \sum_{i=1}^{N} \left( y_i - \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k) \right)^2 \]

---

**The Margin Distribution**

<table>
<thead>
<tr>
<th>epoch</th>
<th>5</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>train error</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>test error</td>
<td>6.4</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 $f_k(x; \gamma_k)$ sequentially, fixing the results of previous iterations
  \[ g_i(x) = g_{i-1}(x) + \alpha_i f_i(x; \gamma_i) \]

- Under the squared difference loss function:
  \[ L(y_i, g_{i-1}(x_i) + \alpha_i f_i(x_i; \gamma_i)) = \left( y_i - g_{i-1}(x_i) - \alpha_i f_i(x_i; \gamma_i) \right)^2 \]

- Forward stage-wise optimization seems to produce classifier with better generalization, doing the process stagewise seems to overfit less quickly
**Boosting As Additive Model**

\[ g(x) = \sum_{m=1}^{M} \alpha_m f_m(x; y_k) \]

- 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:
  \[
  \arg \min_{\alpha_m} \sum_{i=1}^{N} \exp(-y_i \cdot \{g_{m-1}(x_i) + \alpha_m \cdot f_m(x_i)\})
  \]
  \[
  = \arg \min_{\alpha_m} \sum_{i=1}^{N} \exp(-y_i \cdot g_{m-1}(x_i)) \cdot \exp(-y_i \cdot \alpha_m \cdot f_m(x_i))
  \]

**Logistic Regression Model**

- It can be shown that AdaBoost builds a logistic regression model:
  \[
  g(x) = \log \frac{P(Y = 1 | x)}{P(Y = -1 | x)} = \sum_{m=1}^{M} \alpha_m f_m(x)
  \]
- It can also be shown that the the training error on the samples is at most:
  \[
  \sum_{i=1}^{N} \exp(-y_i \cdot g(x_i)) = \sum_{i=1}^{N} \exp(-y_i \cdot \sum_{m=1}^{M} \alpha_m f_m(x_i))
  \]

**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 ($γ_t \rightarrow 0$ too quickly),
    - underfitting
    - Low margins $\rightarrow$ overfitting
- empirically, AdaBoost seems especially susceptible to noise