Lecture 5

Boosting

Some slides are due to Robin Dhamankar
Vandi Verma & Sebastian Thrun
Today

- New Machine Learning Topics:
  - Ensemble Learning
    - Bagging
    - Boosting
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 is based on ensemble learning ideas
• Boosting was inspired by bagging
Bagging

• Generate a random sample from training set by selecting \( I \) elements (out of \( N \) elements available) with replacement

• New sampled dataset has, on average, 63.2% of training examples
  • each example has a probability of \( 1-(1-1/N)^N \) of being selected at least once.
    For \( N \to \infty \), this converges to \((1-1/e)\) or 0.632 [Bauer and Kohavi, 1999]

• Repeat the sampling procedure, getting a sequence of \( k \) independent training sets

• Train classifiers \( f_1(x), f_2(x), \ldots, f_k(x) \) for each of these training sets, using the same classification algorithm

• To classify an unknown sample \( x \), let each classifier predict

• The bagged classifier \( f_{\text{FINAL}}(x) \) combines predictions of individual classifiers, frequently by simple voting

\[
f_{\text{FINAL}}(x) = \text{sign}[1/k \sum f_i(x)]
\]
Boosting: Motivation

- Hard to design accurate classifier which generalizes well
- Easy to find many rule of thumb or weak classifiers
  - a classifier is weak if it is slightly better than random guessing
  - example: if an email has word “money” classify it as spam, otherwise classify it as not spam
    - likely to be better than random guessing
- How combine weak classifiers to produce an accurate classifier?
  - Question people have been working on since 1980’s
  - Ada-Boost (1996) was the first practical boosting algorithm
- 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 was influenced by bagging, and it is superior to bagging
Ada Boost

• Assume 2-class problem, with labels +1 and -1
  • $y^i$ in {-1, 1}

• Ada boost produces a discriminant function:

$$g(x) = \sum_{t=1}^{T} \alpha_t h_t(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots + \alpha_T h_T(x)$$

• Where $h_t(x)$ is a weak classifier, for example:

$$h_t(x) = \begin{cases} 
-1 & \text{if email has word “money”} \\
1 & \text{if email does not have word “money”}
\end{cases}$$

• The final classifier is the sign of the discriminant function

$$f_{\text{final}}(x) = \text{sign}[g(x)]$$
Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially weights are equal
- Main Idea: at successive iterations, the weight of misclassified examples is increased
- This forces the algorithm to concentrate on examples that have not been classified correctly so far
Idea Behind Ada Boost

• Examples of high weight are shown more often at later rounds
• Face/nonface classification problem:

Round 1

best weak classifier:  

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

change weights:  

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1/7</td>
<td>1/7</td>
<td>1/7</td>
<td>1/7</td>
<td>1/7</td>
<td>1/7</td>
<td>1/7</td>
</tr>
</tbody>
</table>

Round 2

best weak classifier:  

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

change weights:  

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8</td>
<td>1/32</td>
<td>11/32</td>
<td>1/2</td>
<td>1/8</td>
<td>1/32</td>
<td>1/32</td>
</tr>
</tbody>
</table>
Idea Behind Ada Boost

Round 3

- out of all available weak classifiers, we choose the one that works best on the data we have at round 3
- we assume there is always a weak classifier better than random (better than 50% error)
- image is half of the data given to the classifier
- chosen weak classifier has to classify this image correctly
More Comments on Ada Boost

• Ada boost is simple to implement, provided you have an implementation of a “weak learner”

• Will work as long as the “basic” classifier $h_t(x)$ is at least slightly better than random
  • will work if the error rate of $h_t(x)$ is less than 0.5
  • 0.5 is the error rate of a random guessing for a 2-class problem

• Can be applied to boost any classifier, not necessarily weak
  • but there may be no benefits in boosting a “strong” classifier
Ada Boost for 2 Classes

**Initialization step:** for each example $x$, set

$$D(x) = \frac{1}{N}, \text{ where } N \text{ is the number of examples}$$

**Iteration step** (for $t = 1...T$):

1. Find best weak classifier $h_t(x)$ using weights $D(x)$

2. Compute the error rate $\varepsilon_t$ as

$$\varepsilon_t = \sum_{i=1}^{N} D(x^i) \cdot [y^i \neq h_t(x^i)]$$

3. Compute weight $\alpha_t$ of classifier $h_t$

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

4. For each $x^i$, $D(x^i) = D(x^i) \cdot \exp(\alpha_t \cdot [y^i \neq h_t(x^i)])$

5. Normalize $D(x^i)$ so that $\sum_{i=1}^{N} D(x^i) = 1$

$$f_{\text{final}}(x) = \text{sign} \left[ \sum \alpha_t h_t(x) \right]$$
Ada Boost: Step 1

1. Find best weak classifier $h_t(x)$ using weights $D(x)$
   - some classifiers accept weighted samples, but most don’t
   - if classifier does not take weighted samples, sample from the training samples according to the distribution $D(x)$

   ![Sampled images](image)

   $\frac{1}{16} \quad \frac{1}{4} \quad \frac{1}{16} \quad \frac{1}{16} \quad \frac{1}{4} \quad \frac{1}{16} \quad \frac{1}{4}$

   • Draw $k$ samples, each $x$ with probability equal to $D(x)$:

   ![Re-sampled images](image)

   re-sampled examples
Adaboost: Step 1

1. Find best weak classifier $h_t(x)$ using weights $D(x)$

- Give to the classifier the re-sampled examples:
- To find the best weak classifier, go through all weak classifiers, and find the one that gives the smallest error on the re-sampled examples

<table>
<thead>
<tr>
<th>weak classifiers</th>
<th>$h_1(x)$</th>
<th>$h_2(x)$</th>
<th>$h_3(x)$</th>
<th>..........</th>
<th>$h_m(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>errors</td>
<td>0.46</td>
<td>0.36</td>
<td>0.16</td>
<td>0.43</td>
<td></td>
</tr>
</tbody>
</table>

the best classifier $h_t(x)$ to choose at iteration $t$
2. Compute $\epsilon_t$ the error rate as

$$\epsilon_t = \sum_{i=1}^{N} D(x^i) \cdot I[y^i \neq h_t(x^i)] = \begin{cases} 1 & \text{if } y^i \neq h_t(x^i) \\ 0 & \text{otherwise} \end{cases}$$

- $\epsilon_t$ is the weight of all misclassified examples added
- the error rate is computed over original examples, not the re-sampled examples
- If a weak classifier is better than random, then $\epsilon_t < \frac{1}{2}$
Ada Boost: Step 3

3. compute weight $\alpha_t$ of classifier $h_t$

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

In example from previous slide:

$$\varepsilon_t = \frac{5}{16} \quad \Rightarrow \quad \alpha_t = \log \frac{1 - \frac{5}{16}}{\frac{5}{16}} = \log \frac{11}{5} \approx 0.8$$

- Recall that $\varepsilon_t < \frac{1}{2}$
- Thus $(1 - \varepsilon_t)/\varepsilon_t > 1 \quad \Rightarrow \quad \alpha_t > 0$
- The smaller is $\varepsilon_t$, the larger is $\alpha_t$, and thus the more importance (weight) classifier $h_t(x)$

$$\text{final}(x) = \text{sign} \left[ \sum \alpha_t h_t(x) \right]$$
Ada Boost: Step 4

4. For each $x^i$, $D(x^i) = D(x^i) \cdot \exp(\alpha_t \cdot I[y^i \neq h_t(x^i)])$

from previous slide $\alpha_t = 0.8$

• weight of misclassified examples is increased
5. Normalize $D(x^i)$ so that $\sum D(x^i) = 1$

from previous slide:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1/16</td>
<td>1/4</td>
<td>1/16</td>
<td>0.14</td>
<td>0.56</td>
<td>1/16</td>
<td>1/4</td>
<td></td>
</tr>
</tbody>
</table>

• after normalization

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.18</td>
<td>0.05</td>
<td>0.10</td>
<td>0.40</td>
<td>0.05</td>
<td>0.18</td>
<td></td>
</tr>
</tbody>
</table>
AdaBoost Example

- Initialization: all examples have equal weights

from “A Tutorial on Boosting” by Yoav Freund and Rob Schapire
AdaBoost Example

ROUND 1

\[ h_1(x) = \text{sign}(3 - x_1) \]

\[ \varepsilon_1 = 0.30 \]

\[ \alpha_1 = 0.42 \]
AdaBoost Example

ROUND 2

\[ h_2(x) = \text{sign}(7 - x_1) \]

\[ \varepsilon_2 = 0.21 \]

\[ \alpha_2 = 0.65 \]
AdaBoost Example

ROUND 3

\[ h_3(x) = \text{sign}(x_2 - 4) \]

\[ \varepsilon_3 = 0.14 \]

\[ \alpha_3 = 0.92 \]
AdaBoost Example

\[ f_{\text{final}}(x) = \text{sign} \left( \frac{211 + 7650 + 3420}{x_{\text{sign}} \cdot x_{\text{sign}} \cdot x_{\text{sign}} \cdot x_{\text{sign}}} \right) \]

- note non-linear decision boundary
AdaBoost Comments

• Can show that training error drops exponentially fast

\[
\text{Err}_{\text{train}} \leq \exp\left(-2 \sum_t \gamma_t^2\right)
\]

• Here \( \gamma_t = \varepsilon_t - 1/2 \), where \( \varepsilon_t \) is classification error at round \( t \)

• Example: let errors for the first four rounds be, 0.3, 0.14, 0.06, 0.03, 0.01 respectively. Then

\[
\text{Err}_{\text{train}} \leq \exp\left[-2\left(0.2^2 + 0.36^2 + 0.44^2 + 0.47^2 + 0.49^2\right)\right] \\
\approx 0.19
\]
AdaBoost Comments

- 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 increases the margins of training examples, as iterations proceed:
  - larger margins help better generalization.
  - 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.
• zero training error

new (test) example

keep training

• zero training error
• larger margins helps better generalization
### Margin Distribution

<table>
<thead>
<tr>
<th>Iteration number</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 ≤ 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>
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:

$$\min_{\alpha_1, \gamma_1, \ldots, \gamma_M, \alpha_M} \sum_{i=1}^{N} L \left( y_i, \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k) \right)$$

For example, under squared-error loss:

$$\min_{\alpha_1, \gamma_1, \ldots, \gamma_M, \alpha_M} \sum_{i=1}^{N} \left( y_i - \sum_{k=1}^{M} \alpha_k f_k(x_i; \gamma_k) \right)^2$$
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

\[
g_t(x) = g_{t-1}(x) + \alpha_t f_t(x; \gamma_t)
\]

• Under the squared difference loss function:

\[
L(y_i, g_{t-1}(x_i) + \alpha_t f_t(x_i; \gamma_t)) =
\]

\[
= (y_i - g_{t-1}(x_i) - \alpha_t f_t(x_i; \gamma_t))^2
\]

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

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

- It can be shown that AdaBoost uses forward stage-wise modeling under the following loss function:
  - \( L(y, g(\mathbf{x})) = \exp(-y \cdot g(\mathbf{x})) \)
    - the exponential loss function
  - At stage (or iteration) \( m \), we fit:

\[
\begin{align*}
\arg \min_{\alpha_m, f_m} \sum_{i=1}^{N} L(y_i, g(\mathbf{x}_i)) &= \\
= \arg \min_{\alpha_m, f_m} \sum_{i=1}^{N} \exp(-y_i \cdot [g_{m-1}(\mathbf{x}_i) + \alpha_m \cdot f_m(\mathbf{x}_i)]) &= \\
= \arg \min_{\alpha_m, f_m} \sum_{i=1}^{N} \exp(-y_i \cdot g_{m-1}(\mathbf{x}_i)) \cdot \exp(-y_i \cdot \alpha_m \cdot f_m(\mathbf{x}_i))
\end{align*}
\]
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.
Logistic Regression Model

- It can be shown that Adaboost builds a logistic regression model:

\[
g(x) = \log \frac{\Pr(Y = 1 \mid x)}{\Pr(Y = -1 \mid x)} = \sum_{k=1}^{M} \alpha_m f_m(x)
\]

- It can also be shown that the training error on the samples is at most:

\[
\sum_{i=1}^{N} \exp(-y_i \cdot g(x_i)) = \sum_{i=1}^{N} \exp \left( -y_i \cdot \sum_{k=1}^{M} \alpha_m f_m(x_i) \right)
\]
Practical Advantages of AdaBoost

- Can construct arbitrarily complex decision regions
- 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
Caveats

• AdaBoost can fail if
  • weak hypothesis too complex (overfitting)
  • weak hypothesis too weak ($\gamma_t \rightarrow 0$ too quickly),
    • underfitting
• empirically, AdaBoost seems especially susceptible to noise
  • noise is the data with wrong labels