Announcement

- A1 due in 2 days!

- Extension?
“Adults do not need to choose”

• Which algorithm to use for my problem?

• Cheap answers
  • Deep learning, but then which architecture?
  • I don’t know; whatever my boss tells me
  • Whatever I can find in scikit-learn
  • I try many and pick the “best”

• Why don’t we combine a few algs? But how?
The Power of Aggregation

- Train $h_t$ on data set $D_t$, say each with accuracy $p > \frac{1}{2}$
- Assuming $D_t$ are independent, hence $h_t$ are independent
- Predict with the majority label among $(2T+1) h_t$’s
- What is the accuracy?

\[
\Pr(\sum_{t=1}^{2T+1} B_t \geq T + 1) \approx 1 - \Phi\left(\frac{T + 1 - (2T + 1) p}{\sqrt{(2T + 1)p(1 - p)}}\right)
\]
Bootstrap Aggregating (Breiman'96)

- Can’t afford to have many independent training sets
- Bootstrapping!

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad \ldots \quad n-1 \quad n \]

Sample with replacement:

\[ 1 \quad 1 \quad 3 \quad 4 \quad 4 \quad \ldots \quad n-1 \quad n \]

\[ h_1 \quad \ldots \quad h_T \]

E.g. majority vote

\[ 2 \quad 2 \quad 2 \quad 4 \quad 5 \quad \ldots \quad n-1 \quad n \]
Bagging for Regression

- Simply average the outputs of $h_t$, each trained on a bootstrapped training set

- With $T$ independent $h_t$, averaging would reduce variance by a factor of $T$
When Bagging Works

• Bagging is essentially averaging

• Beneficial if base classifiers have high variance (instable); e.g., performance changes a lot if training set is slightly perturbed

• Like decision trees but not k-NNs
Randomizing Output (Breiman’00)

• For regression, add small Gaussian noise to each $y_i$ (leaving $x_i$ untouched)
• Train many $h_t$ and average their outputs

• For classification
  • Use one-hot encoding and reduce to regression
  • Randomly flip a small proportion of labels in training set
• Train many $h_t$ and majority vote
Random Forest (Breiman’01)

- A collection of tree-structured classifiers \{h(x; \Theta_t), t=1, ..., T\}, where \Theta_t are iid random vectors.

- Bagging: random samples

- Random feature split

- Both
Leo Breiman (1928-2005)
Boosting

• Given many classifiers $h_t$, each slightly better than random guessing
• Is it possible to construct a classifier with nearly optimal accuracy?

• Yes! First shown by Schapire (1990)
def $EX_2()$
{ flip coin
  if heads, return the first instance $v$ from $EX$ for which $h_1(v) = c(v)$
  else return the first instance $v$ from $EX$ for which $h_1(v) \neq c(v)$
}$h_2 \leftarrow \text{Learn}(\alpha, \delta/5, EX_2)$
{$\tau_2 \leftarrow (1 - 2\alpha)\epsilon/8$
let $\hat{e}$ be an estimate of $e = \text{Pr}_v[c|D][h_2(v) \neq c(v)]$
choose a sample sufficiently large that $|e - \hat{e}| \leq \tau_2$ with probability $\geq 1 - \delta/5$
if $\hat{e} \leq \epsilon - \tau_2$ then return $h_2$

def $EX_3()$
{ return the first instance $v$ from $EX$ for which $h_1(v) \neq h_2(v)$
}$h_3 \leftarrow \text{Learn}(\alpha, \delta/5, EX_3)$

def $h(v)$
{ $b_1 \leftarrow h_1(v)$, $b_2 \leftarrow h_2(v)$
  if $b_1 = b_2$ then return $b_1$
  else return $h_3(v)$
}
return $h$

(Freund, 1995)
Algorithm Hedge($\beta$)

**Parameters:** $\beta \in [0, 1]$  
initial weight vector $w^1 \in [0, 1]^N$ with $\sum_{i=1}^{N} w_i^1 = 1$  
number of trials $T$

**Do for** $t = 1, 2, ..., T$

1. Choose allocation

   $$p^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t}$$

2. Receive loss vector $\ell^t \in [0, 1]^N$ from environment.
3. Suffer loss $p^t \cdot \ell^t$.
4. Set the new weights vector to be

   $$w_{i}^{t+1} = w_i^t \beta^t$$
What Guarantee?

# of rounds

\[ \sum_{t=1}^{T} \mathbf{p}^t \cdot \mathbf{e}^t \leq \left( \ln N - \ln \beta \cdot \min_{i=1,\ldots,N} \sum_{t=1}^{T} \ell_i^t \right) \cdot \frac{1}{1 - \beta} \rightarrow 0 \text{ as } T \to \infty \]

# of experts

total loss of best expert

your total loss

goes to 0 as \( T \to \infty \)

choose beta appropriately

\[
\frac{1}{T} \sum_{t=1}^{T} \mathbf{p}^t \cdot \mathbf{e}^t \leq \min_{i=1,\ldots,N} \frac{1}{T} \sum_{t=1}^{T} \ell_i^t + \sqrt{\frac{2 \ln N}{T} \cdot \min_{i=1,\ldots,N} \frac{1}{T} \sum_{t=1}^{T} \ell_i^t + \frac{\ln N}{T}}
\]
Ada

ptive Boost (Freund & Schapire’97)

Input: sequence of $N$ labeled examples $\langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$ 
- distribution $D$ over the $N$ examples 
- weak learning algorithm WeakLearn 
- integer $T$ specifying number of iterations 

Initialize the weight vector: $w_i^1 = D(i)$ for $i = 1, \ldots, N$. 

Do for $t = 1, 2, \ldots, T$

1. Set

$$p_t = \frac{w_t}{\sum_{i=1}^{N} w_i^t}$$

2. Call WeakLearn, providing it with the distribution $p_t$; get back a hypothesis $h_t: X \rightarrow [0, 1]$. 
3. Calculate the error of $h_t$: $e_t = \sum_{i=1}^{N} p_t \cdot \left| h_t(x_i) - y_i \right|$. 
4. Set $\beta_t = e_t / (1 - e_t)$. 
5. Set the new weights vector to be 

$$w_{i+1} = w_i \beta_t \left| h_t(x_i) - y_i \right|$$

Output the hypothesis

$$h_f(x) = \begin{cases} 
1 & \text{if } \sum_{t=1}^{T} (\log 1/\beta_t) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log 1/\beta_t \\
0 & \text{otherwise.}
\end{cases}$$

Given: $(x_1, y_1), \ldots, (x_m, y_m)$; $x_i \in X, y_i \in \{-1, +1\}$ 
Initialize $D_1(i) = 1/m$. 

For $t = 1, \ldots, T$:

- Train weak learner using distribution $D_t$. 
- Get weak hypothesis $h_t: X \rightarrow \mathbb{R}$. 
- Choose $\alpha_t \in \mathbb{R}$. 
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution). 

Output the final hypothesis:

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

(Schapire & Singer, 1999)
Look Closely

**Initialize** the weight vector: $w_i^1 = D(i)$ for $i = 1, \ldots, N$.

**Do for** $t = 1, 2, \ldots, T$

1. Set
   
   $$h_f(x) = \begin{cases} 
   1 & \text{if } \sum_{t=1}^{T} (\log 1/\beta_t) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log 1/\beta_t \varepsilon_t^{7} \\
   0 & \text{otherwise.} 
   \end{cases}$$

   $$p^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t}$$

2. Call **WeakLearn**, providing it with the distribution $p^t$; get back a hypothesis $h_t: X \rightarrow [0, 1]$.

3. Calculate the error of $h_t$: $\varepsilon_t = \sum_{i=1}^{N} (p_i^t|h_t(x_i) - y_i|)$.

4. Set $\beta_t = \varepsilon_t/(1 - \varepsilon_t)$.

5. Set the new weights vector to be

   $$w_i^{t+1} = w_i^t \beta_t^{1 - |h_t(x_i) - y_i|}$$

   if $\varepsilon_t \leq \frac{1}{2}$

   **bigger $\varepsilon_t$, bigger $\beta_t$, smaller coefficient**

   **can optimize**

   **y = 1 or 0**

   **expected error of h**

   *adaptive*

   **h(x) closer to y, bigger exponent, discount**

   when $w_i^t = 0$?
Does It Work?
Exponential decay of ... training error

\[
\Pr_{i \sim D}[h_f(x_i) \neq y_i] \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1 - \epsilon_t)}
\]

training error!  

\[
\epsilon_t \leq \frac{1}{2} - \gamma \rightarrow \leq \exp(-2T\gamma^2)
\]

- Basically a form of gradient descent to minimize exponential loss: \( \sum_i e^{-\gamma_i h(x_i)} \), h in conic hull of h_t’s
- Overfitting? Use simple base classifiers (e.g., decision stumps)
Will Adaboost Overfit?

Bagging

Boosting

margin: y h(x)
### Seriously? (Grove & Schuurmans’98; Breiman’99)

<table>
<thead>
<tr>
<th>Data set</th>
<th>C4.5 error%</th>
<th>C4.5 win%</th>
<th>Adaboost error%</th>
<th>Adaboost win%</th>
<th>LP-Adaboost error%</th>
<th>LP-Adaboost win%</th>
<th>DualLPboost error%</th>
<th>DualLPboost win%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audiology</td>
<td>22.70</td>
<td>17.0</td>
<td>16.39</td>
<td>0.446</td>
<td>16.48</td>
<td>49.0</td>
<td>0.501</td>
<td>18.09</td>
</tr>
<tr>
<td>Banding</td>
<td>25.58</td>
<td>12.5</td>
<td>15.00</td>
<td>0.528</td>
<td>15.42</td>
<td>45.5</td>
<td>0.565</td>
<td>22.50</td>
</tr>
<tr>
<td>Chess</td>
<td>4.18</td>
<td>12.5</td>
<td>2.70</td>
<td>0.657</td>
<td>2.74</td>
<td>46.5</td>
<td>0.730</td>
<td>2.97</td>
</tr>
<tr>
<td>Colic</td>
<td>14.46</td>
<td>67.5</td>
<td>17.03</td>
<td>0.051</td>
<td>18.97</td>
<td>31.5</td>
<td>0.182</td>
<td>18.16</td>
</tr>
<tr>
<td>Glass</td>
<td>30.91</td>
<td>22.0</td>
<td>23.95</td>
<td>0.513</td>
<td>23.91</td>
<td>49.5</td>
<td>0.624</td>
<td>26.86</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>21.06</td>
<td>38.0</td>
<td>18.94</td>
<td>0.329</td>
<td>17.56</td>
<td>59.0</td>
<td>0.596</td>
<td>20.00</td>
</tr>
<tr>
<td>Labor</td>
<td>15.33</td>
<td>43.0</td>
<td>12.83</td>
<td>0.535</td>
<td>13.83</td>
<td>47.0</td>
<td>0.684</td>
<td>15.17</td>
</tr>
<tr>
<td>Promoter</td>
<td>21.09</td>
<td>10.5</td>
<td>7.55</td>
<td>0.599</td>
<td>8.00</td>
<td>47.0</td>
<td>0.694</td>
<td>13.55</td>
</tr>
<tr>
<td>Sonar</td>
<td>28.81</td>
<td>16.0</td>
<td>18.10</td>
<td>0.628</td>
<td>18.62</td>
<td>48.0</td>
<td>0.685</td>
<td>25.00</td>
</tr>
<tr>
<td>Soybean</td>
<td>8.86</td>
<td>28.5</td>
<td>6.97</td>
<td>-0.005</td>
<td>6.55</td>
<td>62.0</td>
<td>0.017</td>
<td>8.41</td>
</tr>
<tr>
<td>Splice</td>
<td>16.18</td>
<td>0.0</td>
<td>6.83</td>
<td>0.535</td>
<td>7.00</td>
<td>25.0</td>
<td>0.569</td>
<td>11.01</td>
</tr>
<tr>
<td>Vote</td>
<td>4.95</td>
<td>51.0</td>
<td>5.02</td>
<td>0.723</td>
<td>5.30</td>
<td>44.5</td>
<td>0.795</td>
<td>5.27</td>
</tr>
<tr>
<td>Wine</td>
<td>9.11</td>
<td>27.0</td>
<td>4.61</td>
<td>0.869</td>
<td>4.89</td>
<td>47.5</td>
<td>0.912</td>
<td>4.50</td>
</tr>
</tbody>
</table>

- **Adaboost test error**
- **Adaboost train error**
- **LP min margin**
- **Adaboost min margin**
Pros and Cons

- “Straightforward” way to boost performance
- Flexible with any base classifier
- Less interpretable
- Longer training time
  - hard to parallelize (in contrast to bagging)

https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/
Extensions

• LogitBoost
• GradBoost
• L2Boost
• ... you name it

• Multi-class
• Regression
• Ranking
Face Detection (Viola & Jones’01)

- Each detection window results in ~160k features
- Speed is crucial for real-time detection!
Examples
Questions?
Cascading

38 layers with ~6k features selected
Asymmetry (Viola & Jones’02)

- Too many non-faces vs few faces
- Trivial to achieve small false positives by sacrificing true positives
- Asymmetric loss

\[
\begin{array}{c|cc}
\hat{y} = 1 & \hat{y} = -1 \\
\hline
y = 1 & 0 & \sqrt{k} \\
y = -1 & 1/\sqrt{k} & 0 \\
\end{array}
\]

confusion matrix