Example

- gambler wants to write computer program to accurately predict winner of horse races
- gathers historical horse-race database
- discovers:
  - easy to find “rules of thumb” that are “often” correct
  - hard to find single rule that is very highly accurate
The Gambler’s Plan

- devise mechanical procedure for deriving rough rules of thumb
- apply procedure to subset of races:
  - Belmont 1/7/94 race #2 ...
  - Saratoga 8/14/95 race #4 ...
  ...
- obtain rule of thumb:
  “Bet on horse with most favored odds”
- apply to 2nd subset of races
- obtain 2nd rule of thumb
- repeat $T$ times
Details of the Gambler’s Plan

- **how to choose races on each round?**
  - concentrate on “hardest” races
    (those most often misclassified by previous rules of thumb)
- **how to combine rules of thumb into single prediction rule?**
  - take (weighted) majority vote of rules of thumb
**Boosting**

- **boosting** = general method of converting rough rules of thumb into highly accurate prediction rule

- **more technically:**
  - given “weak” learning algorithm that can consistently find hypothesis (classifier) with error $\leq 1/2 - \gamma$
  - a boosting algorithm can **provably** construct single hypothesis with error $\leq \epsilon$

  ($\epsilon, \gamma$ small)
This Talk

- introduction to AdaBoost
- analysis of training error
- analysis of generalization error based on theory of margins
- extensions
- experiments and applications
Background

- [Valiant ’84]:
  - introduced theoretical ("PAC") model for studying machine learning

- [Kearns & Valiant ’88]:
  - open problem of finding a boosting algorithm

- [Schapire ’89], [Freund ’90]:
  - first polynomial-time boosting algorithms

- [Drucker, Schapire & Simard ’92]:
  - first experiments using boosting
Background (cont.)

- [Freund & Schapire '95]:
  - introduced “AdaBoost” algorithm
  - strong practical advantages over previous boosting algorithms

- **experiments using AdaBoost:**
  - [Drucker & Cortes ’95]
  - [Jackson & Craven ’96]
  - [Freund & Schapire ’96]
  - [Quinlan ’96]
  - [Breiman ’96]
  - [Schapire & Singer ’98]
  - [Maclin & Opitz ’97]
  - [Bauer & Kohavi ’97]
  - [Schwenk & Bengio ’98]
  - [Dietterich ’98]

- **continuing development of theory and algorithms:**
  - [Schapire, Freund, Bartlett & Lee ’97]
  - [Breiman ’97]
  - [Grove & Schuurmans ’98]
  - [Schapire & Singer ’98]
  - [Bauer & Kohavi ’97]
  - [Schwenk & Bengio ’98]
  - [Dietterich ’98]
  - [Mason, Bartlett & Baxter ’98]
  - [Friedman, Hastie & Tibshirani ’98]
A Formal View of Boosting

- given training set \((x_1, y_1), \ldots, (x_m, y_m)\)
- \(y_i \in \{-1, +1\}\) correct label of instance \(x_i \in X\)
- for \(t = 1, \ldots, T:\)
  - construct distribution \(D_t\) on \(\{1, \ldots, m\}\)
  - find weak hypothesis ("rule of thumb")
    \(h_t : X \to \{-1, +1\}\)
  - 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 $D_t$:
  - $D_1(i) = 1/m$
  - given $D_t$ and $h_t$:
    
    $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} 
    e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\
    e^{\alpha_t} & \text{if } y_i \neq h_t(x_i)
    \end{cases}$$

    $$= \frac{D_t(i)}{Z_t} \cdot \exp(-\alpha_t \cdot y_i \cdot h_t(x_i))$$

    where $Z_t = \text{normalization constant}$

    $$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) > 0$$

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

$D_1$

weak hypotheses = vertical or horizontal half-planes
$\varepsilon_1 = 0.30$
$\alpha_1 = 0.42$
$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$
Round 3

\[ \alpha_3 = 0.92 \]

\[ \varepsilon_3 = 0.14 \]
Final Hypothesis

\[ H_{\text{final}} = \text{sign} \left( \begin{array}{c}
0.42 \\
+ 0.65 \\
+ 0.92
\end{array} \right) \]
Analyzing the training error

- **Theorem:**
  - write $\epsilon_t$ as $1/2 - \gamma_t$
  - then
    \[
    \text{training error}(H_{\text{final}}) \leq \exp\left(-2 \sum_{t} \gamma_t^2\right)
    \]

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

- **AdaBoost is adaptive:**
  - does **not** need to know $\gamma$ or $T$ a priori
  - can exploit $\gamma_t \gg \gamma$
Proof intuition

- on round $t$:
  increase weight of examples incorrectly classified by $h_t$

- if $x_i$ incorrectly classified by $H_{\text{final}}$
  then $x_i$ incorrectly classified by (weighted) majority of $h_t$’s

∴ if $x_i$ incorrectly classified by $H_{\text{final}}$
  then $x_i$ must have “large” weight under final distribution $D_{T+1}$

∴ number of incorrectly classified examples “small”
  (since total weight $\leq 1$)
A First Attempt at Analyzing the Generalization Error

- expect:
  - training error to continue to drop (or reach zero)
  - test error to increase when $H_{\text{final}}$ becomes “too complex”
    - “Occam’s razor”
  - overfitting — significant problem for many machine learning systems (e.g. decision trees)
Actual Typical Run

- test error does not increase, even after 1000 rounds
  - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

<table>
<thead>
<tr>
<th># of rounds</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>8.4</td>
<td>3.3</td>
<td>3.1</td>
</tr>
</tbody>
</table>

- Occam’s razor wrongly predicts “simpler” rule is better
**A Better Story: Theory of Margins**

[with Freund, Bartlett & Lee]

- **key idea:**
  - training error only measures whether classifications are right or wrong
  - should also consider **confidence** of classifications

- can write:  \( H_{\text{final}}(x) = \text{sign}(f(x)) \)

where  \( f(x) = \frac{\sum_t \alpha_t h_t(x)}{\sum_t \alpha_t} \in [-1, 1] \)

- define **margin** of example \((x, y)\) to be  \( y \cdot f(x) \)
  = measure of confidence of classifications

- **all** \( h_t \)'s incorrect
  **all** \( h_t \)'s correct

- **all** \( h_t \)'s equally divided

- \( H_{\text{final}} \)

- incorrect

- correct

- +1
Margins for Toy Example

\[ f = \left( \begin{array} \n 0.42 \\
+ 0.65 \\
+ 0.92 \\
\end{array} \right) / (0.42 + 0.65 + 0.92) \]
The Margin Distribution

- **margin distribution**
  = cumulative distribution of margins of training examples

![Graph of error vs. # of rounds for train and test data]

<table>
<thead>
<tr>
<th># rounds</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>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>
Analyzing Boosting Using Margins

- can **prove** that:
  - generalization error can be bounded by a function of training sample margins
    - larger margins $\Rightarrow$ **better bound** on generalization error
    - bound is **independent** of number of rounds of boosting
  - **boosting tends to increase** margins of training examples by concentrating on those with smallest margin
Extensions: Confidence-rated Predictions

• useful to allow weak hypotheses to express **confidences** about predictions

• formally, allow $h_t : X \rightarrow \mathbb{R}$

$$\text{sign}(h_t(x)) = \text{prediction}$$

$$|h_t(x)| = \text{“confidence”}$$

• use **same rules**
  • updating distribution
  • combining weak hypotheses

• we give **general principle**
  • choosing $\alpha_t$’s
  • designing weak learner to find (confidence-rated) $h_t$’s

[with Singer]
Multiclass Problems

- most direct extension effective only if all weak hypotheses have error $\leq 1/2$
  - difficult to achieve for “weak” weak learners
- instead, reduce to binary problem by creating several binary questions for each example:

  “does or does not example $x$ belong to class 1?”
  “does or does not example $x$ belong to class 2?”
  \vdots
Practical Advantages of AdaBoost

- **fast**
- **simple** and easy to program
- **no parameters** to tune (except \( T \))
- **no prior knowledge** needed about weak learner
- **provably effective**, provided can consistently find rough rules of thumb
  - shift in mind set — goal now is merely to find hypotheses barely better than random guessing
- **flexible** — can combine with **any** classifier to find weak hypotheses
  - neural nets
  - decision-tree algorithms
  - very simple rules of thumb
Caveats

- performance of AdaBoost depends on data and weak learner
- consistent with theory, AdaBoost can fail if
  - weak hypotheses too complex
    → overfitting
  - weak hypotheses too weak ($\gamma_t \to 0$ too quickly)
    → underfitting
    → low margins → overfitting
- empirically, AdaBoost seems especially susceptible to noise
UCI Experiments

- tested AdaBoost on UCI benchmarks

- used:
  - **C4.5** (Quinlan’s decision tree algorithm)
  - “decision stumps”: very simple rules of thumb that test on single attributes

```plaintext
<table>
<thead>
<tr>
<th>eye color = brown?</th>
<th>height &gt; 5 feet?</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>predict +1</td>
<td>predict -1</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>predict -1</td>
<td>predict -1</td>
</tr>
<tr>
<td></td>
<td>no</td>
</tr>
<tr>
<td>predict -1</td>
<td>predict +1</td>
</tr>
</tbody>
</table>
```
UCI Results

![Diagram showing the results of boosting Stumps and boosting C4.5.](image-url)
Application: Text Categorization

- weak hypotheses are decision stumps
  - test for presence of word or short phrase in document
  - e.g.:
    "If the word Clinton appears in the document predict document is about politics"

- in our experiments, consistently beat or tied tested competitors

<table>
<thead>
<tr>
<th></th>
<th>AP</th>
<th>Reuters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Classes</td>
<td>Number of Classes</td>
<td></td>
</tr>
<tr>
<td>% Error</td>
<td>% Error</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>35</td>
<td></td>
</tr>
</tbody>
</table>

[with Singer]
Confidence-rated Predictions Help a Lot

<table>
<thead>
<tr>
<th>% error</th>
<th>round first reached</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>268</td>
<td>63.2</td>
</tr>
<tr>
<td>35</td>
<td>598</td>
<td>109.2</td>
</tr>
<tr>
<td>30</td>
<td>1,888</td>
<td>&gt;80,000</td>
</tr>
</tbody>
</table>
Application: Collaborative Filtering

- problem: rank unseen movies for user based on:
  - his rankings of movies that he did see
  - rankings provided by previous users
- to use boosting:
  - reduce to a binary problem
  - one instance for each pair of movies:
    "is or is not movie A preferred to movie B?"
  - partial feedback from user’s rankings of other movies
  - derive weak hypotheses from previous users’ rankings
- consistently beat competitors (regression and nearest neighbor)
Improved Comprehensibility: SLIPPER

- want final hypothesis that is human understandable
- build set of rules of form:
  - if \((sex = male) \land (age < 5)\) then \(-1.24\)
  - if \((sex = female) \land (income < 30K)\) then \(+2.71\)
  - if \((income > 20K)\) then \(-3.20\)
- classification = (threshold of) sum of rules that fire
- weak hypotheses ↔ rules (“abstain” if do not fire)
  - confidence-rated boosting provides principle for growing and pruning rules
SLIPPER Experimental Results

- beats competitors (RIPPER, C4.5rules, C5.0rules) on $\approx \frac{2}{3}$ of datasets
- number of rules comparable to C4.5rules and C5.0rules (but not RIPPER)
Conclusions

- **boosting is a useful new tool** for classification problems
  - grounded in rich theory
  - performs well experimentally
  - often (but not always!) resistant to overfitting
  - many applications

- **other stuff:**
  - theoretical connections to
    - game theory and linear programming
    - logistic regression
    - support-vector machines
  - tool for **data cleaning:**
    - very effective at finding outliers (mislabeled or ambiguously labeled examples)