Sometimes a single classifier (e.g., neural network, decision tree) won’t perform well, but a weighted combination of them will. When asked to predict the label for a new example, each classifier (inferred from a base learner) makes its own prediction, and then the master algorithm (or meta-learner) combines them using the weights for its own prediction. If the classifiers themselves cannot learn (e.g., heuristics) then the best we can do is to learn a good set of weights (e.g., Weighted Majority). If we are using a learning algorithm (e.g., ANN, dec. tree), then we can rerun the algorithm on different subsamples of the training set and set the classifiers’ weights during training.

### Bagging

Bagging = Bootstrap aggregating

Bootstrap sampling: given a set \( X \) containing \( N \) training examples:
- Create \( X_j \) by drawing \( N \) examples uniformly at random with replacement from \( X \)
- Expect \( X_j \) to omit \( \approx 37\% \) of examples from \( X \)

#### Experiment

[Breiman, ML Journal, 1996]

Given sample \( \mathcal{X} \) of labeled data, Breiman did the following 100 times and reported avg:
- Divide \( \mathcal{X} \) randomly into test set \( T \) (10%) and train set \( D \) (90%)
- Learn decision tree from \( D \) and let \( e_S \) be error rate on \( T \)
- Do 50 times: Create bootstrap set \( \mathcal{X} \) and learn decision tree (so ensemble size = 50). Then let \( e_B \) be the error of a majority vote of the trees on \( T \)

<table>
<thead>
<tr>
<th>Data Set</th>
<th>( e_S )</th>
<th>( e_B )</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>waveform</td>
<td>29.0</td>
<td>19.4</td>
<td>33%</td>
</tr>
<tr>
<td>heart</td>
<td>10.0</td>
<td>5.3</td>
<td>47%</td>
</tr>
<tr>
<td>breast cancer</td>
<td>6.0</td>
<td>4.2</td>
<td>30%</td>
</tr>
<tr>
<td>ionosphere</td>
<td>11.2</td>
<td>8.6</td>
<td>23%</td>
</tr>
<tr>
<td>diabetes</td>
<td>23.4</td>
<td>18.8</td>
<td>20%</td>
</tr>
<tr>
<td>glass</td>
<td>32.0</td>
<td>24.9</td>
<td>27%</td>
</tr>
<tr>
<td>soybean</td>
<td>14.5</td>
<td>10.6</td>
<td>27%</td>
</tr>
</tbody>
</table>
**Boosting**

Same experiment, but using a nearest neighbor classifier, where prediction of new example \(x\)'s label is that of \(x\)'s nearest neighbor in training set, where distance is e.g., Euclidean distance.

**Results**

<table>
<thead>
<tr>
<th>Data Set</th>
<th>(\bar{e}_S)</th>
<th>(\bar{e}_p)</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>waveform</td>
<td>26.1</td>
<td>26.1</td>
<td>0%</td>
</tr>
<tr>
<td>heart</td>
<td>6.3</td>
<td>6.3</td>
<td>0%</td>
</tr>
<tr>
<td>breast cancer</td>
<td>4.9</td>
<td>4.9</td>
<td>0%</td>
</tr>
<tr>
<td>ionosphere</td>
<td>35.7</td>
<td>35.7</td>
<td>0%</td>
</tr>
<tr>
<td>diabetes</td>
<td>16.4</td>
<td>16.4</td>
<td>0%</td>
</tr>
<tr>
<td>glass</td>
<td>16.4</td>
<td>16.4</td>
<td>0%</td>
</tr>
</tbody>
</table>

What happened?

---

**When Does Bagging Help?**

- When learner is **unstable**, i.e., if small change in training set causes large change in hypothesis produced:
  - Decision trees, neural networks
  - Not nearest neighbor

Experimentally, bagging can help substantially for unstable learners; can somewhat degrade results for stable learners.

---

**Boosting** (Schapire & Freund Book)

Similar to bagging, but don’t always sample uniformly; instead adjust resampling distribution \(p_j\) over \(X\) to focus attention on previously misclassified examples.

Final classifier weights learned classifiers, but not uniform; instead weight of classifier \(d_j\) depends on its performance on data it was trained on.

Final classifier is weighted combination of \(d_1, \ldots, d_L\), where \(d_j\)'s weight depends on its error on \(X\) w.r.t. \(p_j\).

---

**Boosting**

**Algorithm Idea** \([p_j \leftrightarrow D_j; d_j \leftrightarrow h]\)

Repeat for \(j = 1, \ldots, L\):

- Run learning algorithm on examples randomly drawn from training set \(X\) according to distribution \(p_j\) (\(p_1 = \text{uniform}\))
  - Can sample \(X\) according to \(p_j\) and train normally, or directly minimize error on \(X\) w.r.t. \(p_j\)
  - Output of learner is binary hypothesis \(d_j\)
  - Compute error \(p_j(d_j) = \text{error of } d_j \text{ on examples from } X \text{ drawn according to } p_j\) (can compute exactly)
  - Create \(p_{j+1}\) from \(p_j\) by decreasing weight of instances that \(d_j\) predicts correctly

---

**Boosting**

**Algorithm Pseudocode** (Fig 17.2)

**Training:**

For all \(|x', r'|_{i=1}^N \in X\), initialize \(p^0_j = \frac{1}{N}\).

For all base-learners \(j = 1, \ldots, L\):

- Randomly draw \(X_j\) from \(X\) with probabilities \(p^0_j\).
- Train \(d_j\) using \(X_j\).

For each \((x', r')\), calculate \(y'_j = d_j(x')\).

Calculate error rate: \(e_j = \sum p^0_j \cdot 1(y'_j \neq r')\).

- If \(e_j \leq \frac{1}{2}\), stop.
- Else, decide \(\beta_j = e_j / (1 - e_j)\).

For each \((x', r')\), decrease probabilities if correct:

If \(y'_j = r'\), then \(p^1_{j+1} = p^0_j \beta_j p^0_j\).

Else \(p^1_{j+1} = p^0_j / \beta_j\).

Normalize probabilities:

\[Z_j = \sum p^1_{j+1} = \frac{p^1_j}{Z_j} \text{ for all } j\]

**Testing:**

Given \(x\), calculate \(d_j(x)\), \(j = 1, \ldots, L\):

\[y(x) = \sum_{j=1}^L \left( \log \frac{p^1_j}{Z_j} \right) d_j(x)\]
**Boosting**

Schapire & Freund Example: $D_1 = \{ \mathbf{p}_1, \mathbf{h}_1, \mathbf{c}_1 = \frac{1}{2} \ln (1/3) = \frac{1}{2} \ln \left( \frac{1}{3} \right) \}$

---

**Boosting**

Schapire & Freund Example: $D_2 = \{ \mathbf{p}_2, \mathbf{h}_2, \mathbf{c}_2 = \frac{1}{2} \ln (1/3) = \frac{1}{2} \ln \left( \frac{1}{3} \right) \}$

---

**Boosting**

Example (cont'd)

$$H_{\text{final}} = \text{sign}(\alpha_1 \mathbf{p}_1 + \alpha_2 \mathbf{p}_2)$$

---

**Boosting**

Experimental Results

*Scatter plot: Percent classification error of non-boosted vs boosted on 27 learning tasks*

---

**Boosting**

Experimental Results (cont'd)
If $\epsilon_j < 1/2 - \gamma_j$, error of ensemble on $\lambda'$ drops exponentially in $\sum_{j=1}^{J} \gamma_j$

- Can also bound generalization error of ensemble
- Very successful empirically
  - Generalization sometimes improves if training continues after ensemble's error on $\lambda'$ drops to 0
  - Contrary to intuition: would expect overfitting
  - Related to increasing the combined classifier's margin
- Useful even with very simple base learners, e.g., decision stumps