Ensemble methods

Learning theory

- Probability distribution $P$ over $\mathcal{X} \times \{0, 1\}$; let $(X, Y) \sim P$.
- We get $S := \{(x_i, y_i)\}_{i=1}^n$, an iid sample from $P$.
- **Goal:** Fix $\epsilon, \delta \in (0, 1)$. With probability at least $1 - \delta$ (over random choice of $S$), learn a classifier $\hat{f} : \mathcal{X} \to \{-1, +1\}$ with low error rate
  \[\text{err}(\hat{f}) = P(\hat{f}(X) \neq Y) \leq \epsilon.\]
- **Basic question:** When is this possible?
  - Suppose I even promise you that there is a perfect classifier from a particular function class $\mathcal{F}$.
    (E.g., $\mathcal{F}$ = linear classifiers or $\mathcal{F}$ = decision trees.)
  - **Default:** Empirical Risk Minimization (i.e., pick classifier from $\mathcal{F}$ with lowest training error rate), but this might be computationally difficult (e.g., for decision trees).
- **Another question:** Is it easier to learn just non-trivial classifiers in $\mathcal{F}$ (i.e., better than random guessing)?

Boosting

**Boosting:** Using a learning algorithm that provides "rough rules-of-thumb" to construct a **very accurate** predictor.

**Motivation:**

- **Easy** to construct classification rules that are **correct more-often-than-not** (e.g., "If $\geq 5\%$ of the e-mail characters are dollar signs, then it’s spam.").
- but seems **hard** to find a single rule that is **almost always correct**.

**Basic idea:**

- **Input:** training data $S$
  - For $t = 1, 2, \ldots, T$:
    1. Choose subset of examples $S_t \subseteq S$ (or a distribution over $S$).
    2. Use "**weak learning**" algorithm to get classifier: $f_t := \text{WL}(S_t)$.
  - Return an “ensemble classifier” based on $f_1, f_2, \ldots, f_T$.

Boosting: history

- **1984** Valiant and Kearns ask whether “boosting” is theoretically possible (formalized in the PAC learning model).
- **1989** Schapire creates first boosting algorithm, solving the open problem of Valiant and Kearns.
- **1990** Freund creates an optimal boosting algorithm (Boost-by-majority).
- **1995** Freund and Schapire create AdaBoost—a boosting algorithm with practical advantages over early boosting algorithms.

**Winner of 2004 ACM Paris Kanellakis Award:**

*For their "seminal work and distinguished contributions [...] to the development of the theory and practice of boosting, a general and provably effective method of producing arbitrarily accurate prediction rules by combining weak learning rules"; specifically, for AdaBoost, which "can be used to significantly reduce the error of algorithms used in statistical analysis, spam filtering, fraud detection, optical character recognition, and market segmentation, among other applications".*
**AdaBoost**

**Input** Training data \( \{(x_i, y_i)\}_{i=1}^n \) from \( \mathcal{X} \times \{-1, +1\} \).

1. Initialize \( D_1(i) := 1/n \) for each \( i = 1, 2, \ldots, n \) (a probability distribution).
2. For \( t = 1, 2, \ldots, T \) do
   3. Give \( D_t \)-weighted examples to WL; get back \( f_t : \mathcal{X} \to \{-1, +1\} \).
   4. Update weights:
      \[
      z_t := \sum_{i=1}^n D_t(i) \cdot y_i f_t(x_i) \in [-1, +1] \\
      \alpha_t := \frac{1}{2} \ln \frac{1 + z_t}{1 - z_t} \in \mathbb{R} \quad \text{(weight of } f_t) \\
      D_{t+1}(i) := \frac{D_t(i) \exp(-\alpha_t \cdot y_i f_t(x_i))}{Z_t} \quad \text{for each } i = 1, 2, \ldots, n, \\
      \text{where } Z_t > 0 \text{ is normalizer that makes } D_{t+1} \text{ a probability distribution.}
      \]
3. Return Final classifier \( \hat{f}(x) := \text{sign} \left( \sum_{t=1}^T \alpha_t \cdot f_t(x) \right) \).

(Example weights \( D_{t+1}(i) \))

**Example: AdaBoost with decision stumps**

Weak learning algorithm WL: ERM with \( \mathcal{F} = \text{"decision stumps" on } \mathbb{R}^2 \)
(i.e., axis-aligned threshold functions \( x \mapsto \text{sign}(vx_i - t) \)).
Straightforward to handle importance weights in ERM.

(Example from Figures 1.1 and 1.2 of Schapire & Freund text.)

---

**Interpretation**

Interpreting \( z_t \)

Suppose \((X, Y) \sim D_t\). If

\[
P(f(X) = Y) = \frac{1}{2} + \gamma_t,
\]

then

\[
z_t = \sum_{i=1}^n D_t(i) \cdot y_i f(x_i) = 2\gamma_t \in [-1, +1].
\]

- \( z_t = 0 \iff \text{random guessing w.r.t. } D_t. \)
- \( z_t > 0 \iff \text{better than random guessing w.r.t. } D_t. \)
- \( z_t < 0 \iff \text{better off using the opposite of } f\text{'s predictions.} \)

**Classifier weights** \( \alpha_t = \frac{1}{2} \ln \frac{1 + z_t}{1 - z_t} \)

**Example weights** \( D_{t+1}(i) \)

\[
D_{t+1}(i) \propto D_t(i) \cdot \exp(-\alpha_t \cdot y_i f_t(x_i)).
\]
Example: execution of AdaBoost

\[ \begin{align*}
D_1 & \quad D_2 & \quad D_3 \\
+ & - & - \\
+ & - & + \\
+ & - & - \\
\end{align*} \]

\[ f_1 \quad f_2 \quad f_3 \]

\[ z_1 = 0.40, \alpha_1 = 0.42 \]
\[ z_2 = 0.58, \alpha_2 = 0.65 \]
\[ z_3 = 0.72, \alpha_3 = 0.92 \]

Final classifier

\[ \hat{f}(x) = \text{sign}(0.42f_1(x) + 0.65f_2(x) + 0.92f_3(x)) \]
(Zero training error rate!)

Example: final classifier from AdaBoost

\[ \begin{align*}
D_1 & \quad D_2 & \quad D_3 \\
+ & - & - \\
+ & - & + \\
+ & - & - \\
\end{align*} \]

\[ f_1 \quad f_2 \quad f_3 \]

\[ z_1 = 0.40, \alpha_1 = 0.42 \]
\[ z_2 = 0.58, \alpha_2 = 0.65 \]
\[ z_3 = 0.72, \alpha_3 = 0.92 \]

Empirical results

Test error rates of C4.5 and AdaBoost on several classification problems. Each point represents a single classification problem/dataset from UCI repository.

C4.5 = popular algorithm for learning decision trees.
(Figure 1.3 from Schapire & Freund text.)

Training error rate of final classifier from AdaBoost

Recall \( \gamma_t := P(f_t(X) = Y) - 1/2 = z_t/2 \) when \((X,Y) \sim D_t\).

**Training error rate of final classifier from AdaBoost:**

\[ \text{err}(\hat{f}, \{(x_i, y_i)\}_{i=1}^n) \leq \exp \left( -2 \sum_{t=1}^T \gamma_t^2 \right). \]

If average \( \bar{\gamma}^2 := \frac{1}{T} \sum_{t=1}^T \gamma_t^2 > 0 \), then training error rate is \( \leq \exp \left( -2\bar{\gamma}^2T \right) \).

"AdaBoost" = "Adaptive Boosting"

Some \( \gamma_t \) could be small, even negative—only care about overall average \( \bar{\gamma}^2 \).

What about true error rate?
COMBINING CLASSIFIERS

Let \( \mathcal{F} \) be the function class used by the weak learning algorithm WL. The function class used by AdaBoost is

\[
\mathcal{F}_T := \left\{ x \mapsto \text{sign} \left( \sum_{t=1}^T \alpha_t f_t(x) \right) : f_1, f_2, \ldots, f_T \in \mathcal{F}, \alpha_1, \alpha_2, \ldots, \alpha_T \in \mathbb{R} \right\}
\]

i.e., linear combinations of \( T \) functions from \( \mathcal{F} \).

Complexity of \( \mathcal{F}_T \) grows linearly with \( T \).

Theoretical guarantee (e.g., when \( \mathcal{F} = \) decision stumps in \( \mathbb{R}^d \)): With high probability (over random choice of training sample),

\[
\text{err}(\hat{f}) \leq \exp\left( -2\gamma^2 T \right) + O\left( \sqrt{\frac{T \log d}{n}} \right).
\]

Theory suggests danger of overfitting when \( T \) is very large. Indeed, this does happen sometimes ... but often not!

BOOSTING THE MARGIN

Final classifier from AdaBoost:

\[
\hat{f}(x) = \text{sign} \left( \frac{\sum_{t=1}^T \alpha_t f_t(x)}{\sum_{t=1}^T |\alpha_t|} \right).
\]

Call \( y \cdot g(x) \in [-1, +1] \) the margin achieved on example \((x, y)\).

New theory [Schapire, Freund, Bartlett, and Lee, 1998]:

- Larger margins ⇒ better resistance to overfitting, independent of \( T \).
- AdaBoost tends to increase margins on training examples.

(Similar but not the same as SVM margins.)

On “letters” dataset:

<table>
<thead>
<tr>
<th>( T )</th>
<th>Training error rate</th>
<th>Test error rate</th>
<th>% margins ≤ 0.5</th>
<th>min. margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T = 5 )</td>
<td>0.0%</td>
<td>8.4%</td>
<td>7.7%</td>
<td>0.14</td>
</tr>
<tr>
<td>( T = 100 )</td>
<td>0.0%</td>
<td>3.3%</td>
<td>0.0%</td>
<td>0.52</td>
</tr>
<tr>
<td>( T = 1000 )</td>
<td>0.0%</td>
<td>3.1%</td>
<td>0.0%</td>
<td>0.55</td>
</tr>
</tbody>
</table>

LINEAR CLASSIFIERS

Regard function class \( \mathcal{F} \) used by weak learning algorithm as “feature functions”:

\[
x \mapsto \phi(x) := (f(x) : f \in \mathcal{F}) \in \{-1, +1\}^\mathcal{F}
\]

(possibly infinite dimensional!).

AdaBoost’s final classifier is a linear classifier in \( \{-1, +1\}^\mathcal{F} \):

\[
\hat{f}(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t f_t(x) \right) = \text{sign} \left( \sum_{f \in \mathcal{F}} w_f f(x) \right) = \text{sign}(\langle w, \phi(x) \rangle)
\]

where

\[
w_f := \sum_{t=1}^T \alpha_t \cdot \mathbb{1}\{f_t = f\} \quad \forall f \in \mathcal{F}.
\]
Exponential loss

AdaBoost is a particular “coordinate descent” algorithm (similar to but not the same as gradient descent) for

$$\min_{w \in \mathbb{R}^F} \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i \langle w, \phi(x_i) \rangle).$$

More on boosting

Many variants of boosting:
- AdaBoost with different loss functions.
- Boosted decision trees = boosting + decision trees.
- Boosting algorithms for ranking and multi-class.
- Boosting algorithms that are robust to certain kinds of noise.
- Boosting for online learning algorithms (very new!).
- . . .

Many connections between boosting and other subjects:
- Game theory, online learning
- “Geometry” of information (replace $\| \cdot \|_2$ with relative entropy divergence)
- Computational complexity
- . . .

Application: face detection

Face detection
**Problem:** Given an image, locate all of the faces.

**As a classification problem:**
- Divide up images into patches (at varying scales, e.g., $24 \times 24$, $48 \times 48$).
- Learn classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{Y} = \{\text{face, not face}\}$.

Many other things built on top of face detectors (e.g., face tracking, face recognizers); now in every digital camera and iPhoto/Picasa-like software.

**Main problem:** how to make this very fast.
Face detectors via AdaBoost [Viola & Jones, 2001]

Face detector architecture by Viola & Jones (2001): major achievement in computer vision; detector actually usable in real-time.

-Think of each image patch ($d \times d$-pixel gray-scale) as a vector $x \in [0, 1]^{d^2}$.
-Used weak learning algorithm that picks linear classifiers $f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)$, where $w$ has a very particular form:

$$\langle w, x \rangle = \sum_{i} w_i x_i$$

-AdaBoost combines many “rules-of-thumb” of this form.
  -Very simple.
  -Extremely fast to evaluate via pre-computation.

Viola & Jones cascade architecture

Problem: severe class imbalance (most patches don’t contain a face).

Solution: Train several classifiers (each using AdaBoost), and arrange in a special kind of decision list called a cascade:

$$x \rightarrow f^{(1)}(x) \rightarrow f^{(2)}(x) \rightarrow \cdots \rightarrow f^{(n)}(x)$$

-Each $f^{(i)}$ is trained (using AdaBoost), adjust threshold (before passing through sign) to minimize false negative rate.
-Can make $f^{(i)}$ in later stages more complex than in earlier stages, since most examples don’t make it to the end.

⇒ (Cascade) classifier evaluation extremely fast.

Viola & Jones detector: example results

Viola & Jones “integral image” trick

“Integral image” trick:
For every image, pre-compute $s(r,c) =$ sum of pixel values in rectangle from $(0,0)$ to $(r,c)$

(single pass through image).

To compute inner product

$$\langle w, x \rangle = \text{average pixel value in black box} - \text{average pixel value in white box}$$

just need to add and subtract a few $s(r,c)$ values.

⇒ Evaluating “rules-of-thumb” classifiers is extremely fast.

Viola & Jones “integral image” trick

“Integral image” trick:
For every image, pre-compute $s(r,c) =$ sum of pixel values in rectangle from $(0,0)$ to $(r,c)$

(single pass through image).

To compute inner product

$$\langle w, x \rangle = \text{average pixel value in black box} - \text{average pixel value in white box}$$

just need to add and subtract a few $s(r,c)$ values.

⇒ Evaluating “rules-of-thumb” classifiers is extremely fast.
Summary

Two key points:
▶ AdaBoost effectively combines many fast-to-evaluate “weak classifiers”.
▶ Cascade structure optimizes speed for common case.

Bagging

Bagging = Bootstrap aggregating (Leo Breiman, 1994).
Input: training data \{((x_i, y_i))_{i=1}^n\} from \mathcal{X} \times \{-1, +1\}.
For \(t = 1, 2, \ldots, T\):
1. Randomly pick \(n\) examples with replacement from training data \(\{((x_i^{(t)}, y_i^{(t)}))_{i=1}^n\}\) (a bootstrap sample).
2. Run learning algorithm on \(\{((x_i^{(t)}, y_i^{(t)}))_{i=1}^n\}\) classifier \(f_t\).

Return a majority vote classifier over \(f_1, f_2, \ldots, f_T\).

Aside: sampling with replacement

Question: if \(n\) individuals are picked from a population of size \(n\) u.a.r. with replacement, what is the probability that a given individual is not picked?

Answer:

\[ \left(1 - \frac{1}{n}\right)^n \]

For large \(n\):

\[ \lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679. \]

Implications for bagging:
▶ Each bootstrap sample contains about 63% of the data set.
▶ Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
▶ Can average across bootstrap samples to get estimate of bagged classifier’s error rate (sort of).
Random Forests (Leo Breiman, 2001).

Input: training data \( \{(x_i, y_i)\}_{i=1}^{n} \) from \( \mathbb{R}^{d} \times \{-1, +1\} \).

For \( t = 1, 2, \ldots, T \):

1. Randomly pick \( n \) examples with replacement from training data \( \rightarrow \{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^{n} \) (a bootstrap sample).

2. Run variant of decision tree learning algorithm on \( \{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^{n} \), where each split is chosen by only considering a random subset of \( \sqrt{d} \) features (rather than all \( d \) features) \( \rightarrow \) decision tree classifier \( f_t \).

Return a majority vote classifier over \( f_1, f_2, \ldots, f_T \).

Breaking the holdout

1. Randomly split data into three sets: Training, Holdout, and Test data.

2. Loop:
   - Train some classifiers on Training data.
     Evaluate on Holdout data.
   - Exit loop at some point.

3. Use results from previous step to pick final classifier.

Why is this different from the basic holdout method?

Steps after first loop iteration might not be independent of Holdout.
Extreme case of overfitting

(A variant of “Freedman's paradox”.)

- Construct completely random classifiers $f_1, f_2, \ldots, f_K: \mathcal{X} \to \{-1, +1\}$.
- Compute Holdout error rates

$$\text{err}(f_i, \text{Holdout}), \ i = 1, 2, \ldots, K.$$ 

- Select better-than-random classifiers: $G := \{i : \text{err}(f_i, \text{Holdout}) > 1/2\}$.
- Construct $\hat{f} :=$ majority vote classifier over $\{f_i : i \in G\}$.
- Compute Holdout error rate

$$\text{err}(\hat{f}, \text{Holdout}).$$

Is $\text{err}(\hat{f}, \text{Holdout})$ a good estimate of $\text{err}(\hat{f})$?

What can we do?

- Avoid excessive adaptation to Holdout if you can help it.
  (Think carefully about what to evaluate; use Training data when possible.)
- Guard against inadvertent “information leakage” from Holdout.
  *Example of leakage:* using both Training and Holdout to determine standardization or other preprocessing.
- Use “blurred vision” when looking at Holdout: e.g.,

$$\text{err}(f, \text{Holdout}) + \text{noise},$$

where noise is on the order of expected standard deviation.
- Explicit safeguards for adaptive data analysis.

Cautionary tale

nytimes.com/2015/06/04/technology/computer-scientists-are-astir-after-baidu-team-is-barred-from-ai-competition.html

It requires that computer systems created by the teams classify the objects in a set of digital images into 1,000 different categories. The rules of the contest permit each team to run test versions of their programs twice weekly ahead of a final submission as they train their programs to “learn” what they are seeing.

(Holdout)

However, on Tuesday, the contest organizers posted a public statement noting that between November and May 30, different accounts had been used by the Baidu team to submit more than 200 times to the contest server, “far exceeding the specified limit of two submissions per week.”