Ensemble methods

COMS 4771

1. Weak versus strong learning

PAC learning

Learning protocol

- Get objects $x_1, \ldots, x_n \sim_{iid} P$, each paired with label $c(x_i)$.
- Using a procedure that takes input $(x_1, c(x_1)), \ldots, (x_n, c(x_n))$ and runs in time $t$, you pick a hypothesis $h: X \rightarrow \{0, 1\}$.
- Get quiz object $x \sim P$ (independent of all previous $x_i$).
- Evaluate: is $h(x) = c(x)$?

Probably approximately correct (PAC) learning (Valiant, 1984):

For any $\epsilon, \delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\epsilon, 1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x(1), \ldots, x(n) \sim_{iid} P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon \right] \geq 1 - \delta.$$
Boosting

Boosting: using an algorithm for weak learning (i.e., a weak learner) to achieve strong learning.

Basic template for boosting algorithms:

Input: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{-1, +1\}\)

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

1. Choose distribution \(D_t\) over training examples.
2. Use weak learner with \(D_t\) to get weak hypothesis \(h_t\).
3. Given \(D_t\)-weighted examples to WL; get back \(h_t: \mathcal{X} \to \{-1, +1\}\).

Return: single “ensemble” hypothesis based on \(h_1, \ldots, h_T\).

AdaBoost

input Training examples \((x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{-1, +1\}\)

1. initialize \(D_1(i) := 1/n\) for each \(i = 1, \ldots, n\) (a probability distribution).
2. for \(t = 1, \ldots, T\) do
3. 3. Give \(D_t\)-weighted examples to WL; get back \(h_t: \mathcal{X} \to \{-1, +1\}\).
4. 4. Update weights:
\[
\gamma_t := \frac{\sum_{i=1}^{n} D_t(i) \cdot 1\{h_t(x_i) = y_i\}}{\sum_{i=1}^{n} D_t(i)} - \frac{1}{2}
\]
\[
\alpha_t := \frac{1}{2} \ln \frac{1 + 2\gamma_t}{1 - 2\gamma_t} \quad \text{(weight of } h_t\text{)}
\]
\[
D_{t+1}(i) := \frac{D_t(i) \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)}{Z_t}
\]
where \(Z_t > 0\) is normalizer that makes \(D_{t+1}\) a probability distribution.
5. end for
6. return Final hypothesis \(\hat{h}(x) := \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right)\).

(Let \(\text{sign}(z) := 1\) if \(z > 0\) and \(\text{sign}(z) := -1\) if \(z \leq 0\).)

Interpretation

Regarding \(D_t\) as a distribution over training examples \((x_1, y_1), \ldots, (x_n, y_n),\)

\[
\gamma_t = \Pr_{(x, y) \sim D_t} (h_t(x) = y) - \frac{1}{2} \quad \text{(i.e., accuracy minus 1/2)}.
\]

Hypothesis weights \(\alpha_t = \frac{1}{2} \ln \frac{1 + 2\gamma_t}{1 - 2\gamma_t}\).

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

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

Which data get higher weight in \(D_{t+1}\)?

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”.

1988 Kearns and Valiant ask whether boosting is theoretically possible.
1989 Schapire creates boosting algorithm; solves Kearns and Valiant’s problem.
1990 Freund creates optimal boosting algorithm (Boost-by-majority).
1995 Freund and Schapire create AdaBoost—a boosting algorithm with practical advantages over previous boosting algorithms.
Example: AdaBoost with decision stumps

Weak learner: ERM with \( H = \text{"decision stumps"} \) on \( \mathbb{R}^2 \) (decision trees with one axis-aligned split).
Straightforward to handle importance weights in ERM.

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

Example: execution of AdaBoost

\[
\begin{align*}
\gamma_1 &= 0.20, \alpha_1 = 0.42 \\
\gamma_2 &= 0.29, \alpha_2 = 0.65 \\
\gamma_3 &= 0.36, \alpha_3 = 0.92
\end{align*}
\]

Example: final hypothesis from AdaBoost

\[
\hat{h}(x) = \text{sign}(0.42h_1(x) + 0.65h_2(x) + 0.92h_3(x))
\]
(Zero training error rate!)

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 hypothesis

Recall \( \gamma_t := \Pr_{(x,y) \sim D_t}(h_t(x) = y) - 1/2. \)

Training error rate of final hypothesis from AdaBoost:

\[
\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\hat{h}(x_i) \neq y_i\} \leq \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2\right).
\]

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

"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 hypotheses

Let \( \mathcal{H} \) be the hypothesis class used by the weak learner WL.

The hypothesis class used by AdaBoost is

\[
\mathcal{H}_T := \{x \mapsto \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right): h_1, \ldots, h_T \in \mathcal{H}, \alpha_1, \ldots, \alpha_T \in \mathbb{R}\}
\]

i.e., thresholding linear combinations of \( T \) hypotheses from \( \mathcal{H}. \)

Fact: VC dimension of \( \mathcal{H}_T \) is \( \approx T \cdot \text{VC}(\mathcal{H}). \)

Theoretical guarantee:

If WL uses hypothesis class \( \mathcal{H}, \) then with high probability over choice of \( (x_1, y_1), \ldots, (x_n, y_n) \sim_{iid} P, \) AdaBoost returns hypothesis \( \hat{h} \in \mathcal{H}_T \) satisfying

\[
\Pr_{(x,y) \sim P}(\hat{h}(x) \neq y) \leq \exp\left(-2\bar{\gamma}^2 T\right) + O\left(\sqrt{\frac{T \cdot \text{VC}(\mathcal{H})}{n}}\right).
\]

Theory suggests danger of overfitting when \( T \) is large relative to \( n. \)

Indeed, this does happen sometimes . . . but often not!

A typical run of boosting

AdaBoost+C4.5 on “letters” dataset.

![Graph showing error rate over rounds](Figure 1.7 from Schapire & Freund text)

Final hypothesis from AdaBoost:

\[
\hat{f}(x) = \text{sign}\left(\frac{\sum_{t=1}^{T} \alpha_t \cdot h_t(x)}{\sum_{t=1}^{T} |\alpha_t|}\right), \quad x \in \mathcal{X}.
\]

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

(Similar to, but not the same as, SVM margins.)

Margins theory (Schapire, Freund, Bartlett, and Lee, 1998):

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

On “letters” dataset:

<table>
<thead>
<tr>
<th>( T )</th>
<th>training error rate</th>
<th>test error rate</th>
<th>% margins ( \leq 0.5 )</th>
<th>min. margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.0%</td>
<td>8.4%</td>
<td>7.7%</td>
<td>0.14</td>
</tr>
<tr>
<td>100</td>
<td>0.0%</td>
<td>3.3%</td>
<td>0.0%</td>
<td>0.52</td>
</tr>
<tr>
<td>1000</td>
<td>0.0%</td>
<td>3.1%</td>
<td>0.0%</td>
<td>0.55</td>
</tr>
</tbody>
</table>
Linear classifiers

Implicit feature mapping based on hypothesis class $\mathcal{H}$:

$$x \mapsto \phi(x) := (h(x) : h \in \mathcal{H}) \in \{-1,+1\}^\mathcal{H}$$

(possibly infinite dimensional!).

AdaBoost’s final hypothesis is a linear classifier in $\{-1,+1\}^\mathcal{H}$:

$$\hat{h}(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right) = \text{sign} \left( \sum_{h \in \mathcal{H}} w_h h(x) \right) = \text{sign} \left( w^T \phi(x) \right)$$

where

$$w_f := \sum_{t=1}^T \alpha_t \cdot 1\{h_t = h\}, \ h \in \mathcal{H}.$$
Application: 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 $\mathcal{Y} = \{\text{face}, \text{not face}\}$.

Many other things built on top of face detectors (e.g., face tracking, face recognizers).

Main problem: how to make this very fast.

Viola & Jones “integral image” trick

"Integral image" trick:
- For every image, pre-compute $s(r,c) = \text{sum of pixel values in rectangle from } (0,0) \text{ to } (r,c)$ (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 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:

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

⇒ (Cascade) classifier evaluation extremely fast.
3. Bagging

**Bagging** = **Bootstrap aggregating** (Leo Breiman, 1994).

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

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

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

2. Run learning algorithm on \( S^{(t)} \) \( \rightarrow \) classifier \( f_t \).

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

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

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

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

2. Run variant of decision tree learning algorithm on \( S^{(t)} \), 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 \( f_1, \ldots, f_T \).

Why do Random Forests work well?

- Usually, decision trees are grown to perfectly fit the bootstrap sample. (Overfitting???)
- The hope is that sampling with replacement from training data is similar to sampling new training data from \( P \).

<table>
<thead>
<tr>
<th>Distribution ( P )</th>
<th>multiple IID samples from ( P ): ( S_1, S_2, \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empirical distribution ( P_n )</td>
<td>multiple IID samples from ( P_n ): ( S_n^{(1)}, S_n^{(2)}, \ldots )</td>
</tr>
</tbody>
</table>

- Combining trees in majority-vote “reduces variance” (helps to prevent overfitting).

Key takeaways

1. Weak and strong learning.
2. AdaBoost algorithm; concept of margins in boosting.
3. Interpreting AdaBoost’s final classifier as a linear classifier, and interpreting AdaBoost as a coordinate descent algorithm.
4. Structure of decision lists / cascades.
5. Concept of bootstrap samples; bagging and random forests.