CSE446: Ensemble Learning - Bagging and Boosting
Winter 2015

Luke Zettlemoyer

Slides adapted from Carlos Guestrin, Nick Kushmerick, Padraig Cunningham
Voting (Ensemble Methods)

• Instead of learning a single classifier, learn many weak classifiers that are good at different parts of the data

• **Output class:** (Weighted) vote of each classifier
  – Classifiers that are most “sure” will vote with more conviction
  – Classifiers will be most “sure” about a particular part of the space
  – On average, do better than single classifier!

• **But how??**
  – force classifiers to learn about different parts of the input space? different subsets of the data?
  – weigh the votes of different classifiers?
BAGGing = **Bootstrap AGGregation**  
(Breiman, 1996)

- for i = 1, 2, …, K:
  - $T_i \leftarrow$ randomly select M training instances with replacement
  - $h_i \leftarrow$ learn($T_i$)  
    \[\text{[Decision Tree, Naive Bayes, …]}\]

- Now combine the $h_i$ together with uniform voting ($w_i = 1/K$ for all i)
Bagging Example
decision tree learning algorithm; very similar to version in earlier slides

CART decision boundary
100 bagged trees

shades of blue/red indicate strength of vote for particular classification
Regression results
Squared error loss

CART
Bagged CART

Boston Housing  Ozone  Friedman #1  Friedman #2  Friedman #3
Fighting the bias-variance tradeoff

• **Simple (a.k.a. weak) learners are good**
  – e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  – Low variance, don’t usually overfit

• **Simple (a.k.a. weak) learners are bad**
  – High bias, can’t solve hard learning problems

• **Can we make weak learners always good??**
  – No!!!
  – But often yes...
Boosting [Schapire, 1989]

- **Idea:** given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote.

- **On each iteration** $t$:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis – $h_t$
  - A strength for this hypothesis – $\alpha_t$

$$h(x) = \text{sign} \left( \sum_i \alpha_i h_i(x) \right)$$

- **Final classifier:**
- **Practically useful**
- **Theoretically interesting**
time = 0
blue/red = class
size of dot = weight
weak learner =
Decision stub:
horizontal or vertical

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets.
time = 1

data:

this hypothesis has 15% error
and so does this ensemble, since the ensemble contains just this one hypothesis.
time = 2
time = 3

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets.
time = 13

First, generate a data set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets.
First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets.

Applet adaboost started
time = 300

overfitting!!

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets.

Applet adaboost started
Learning from weighted data

• Consider a weighted dataset
  – \( D(i) \) – weight of \( i \)th training example \((x^i, y^i)\)
  – Interpretations:
    • \( i \)th training example counts as if it occurred \( D(i) \) times
    • If I were to “resample” data, I would get more samples of “heavier” data points

• Now, always do weighted calculations:
  – e.g., MLE for Naïve Bayes, redefine \( \text{Count}(Y=y) \) to be weighted count:

\[
\text{Count}(Y = y) = \sum_{j=1}^{n} D(j) \delta(Y^j = y)
\]

  – setting \( D(j)=1 \) (or any constant value!), for all \( j \), will recreates unweighted case
Given: \((x^1, y^1), \ldots, (x^m, y^m)\) where \(x^i \in \mathbb{R}^n, y^i \in \{-1, +1\}\)

Initialize: \(D_1(i) = 1/m, \) for \(i = 1, \ldots, m\)

For \(t=1\ldots T:\)
- Train base classifier \(h_t(x)\) using \(D_t\)
- Choose \(\alpha_t\)
- Update, for \(i=1..m:\)

\[
D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y^i h_t(x^i))
\]

with normalization constant:
\[
\sum_{i=1}^m D_t(i) \exp(-\alpha_t y^i h_t(x^i))
\]

Output final classifier:
\[
H(x) = \text{sign}\left(\sum_{i=1}^T \alpha_t h_t(x)\right)
\]

How? Many possibilities. Will see one shortly!

Why? Reweight the data: examples \(i\) that are misclassified will have higher weights!

- \(y^i h_t(x^i) > 0 \rightarrow h_i\) correct
- \(y^i h_t(x^i) < 0 \rightarrow h_i\) wrong
- \(h_i\) correct, \(\alpha_t > 0 \rightarrow D_{t+1}(i) < D_t(i)\)
- \(h_i\) wrong, \(\alpha_t > 0 \rightarrow D_{t+1}(i) > D_t(i)\)

Final Result: linear sum of “base” or “weak” classifier outputs.
Given: \((x^1, y^1), \ldots, (x^m, y^m)\) where

Initialize: \(D_1(i) = 1/m,\) for \(i = 1, \ldots, m\)

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

- Train base classifier \(h_t(x)\) using \(D_t\)
- Choose \(\alpha_t\)
- Update, for \(i = 1 \ldots m:\)

\[
D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y^i h_t(x^i))
\]

- \(\varepsilon_t:\) error of \(h_t,\) weighted by \(D_t\)
  - \(0 \leq \varepsilon_t \leq 1\)
- \(\alpha_t:\)
  - No errors: \(\varepsilon_t = 0 \Rightarrow \alpha_t = \infty\)
  - All errors: \(\varepsilon_t = 1 \Rightarrow \alpha_t = -\infty\)
  - Random: \(\varepsilon_t = 0.5 \Rightarrow \alpha_t = 0\)
What $\alpha_t$ to choose for hypothesis $h_t$?

Idea: choose $\alpha_t$ to minimize a bound on training error!

$$\sum_{i=1}^{m} \delta(H(x^i) \neq y^i) \leq \sum_{i=1}^{m} D_t(i) \exp(-y^i f(x^i))$$

Where

$$f(x) = \sum_{t} \alpha_t h_t(x); \quad H(x) = \text{sign}(f(x))$$
What $\alpha_t$ to choose for hypothesis $h_t$?

Idea: choose $\alpha_t$ to minimize a bound on training error!

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x^i) \neq y^i) \leq \frac{1}{m} \sum_{i=1}^{m} D_t(i) \exp(-y^i f(x^i)) = \prod_t Z_t$$

Where

$$f(x) = \sum_t \alpha_t h_t(x); \quad H(x) = \text{sign}(f(x))$$

And

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y^i h_t(x^i))$$

If we minimize $\prod_t Z_t$, we minimize our training error!!!

- We can tighten this bound greedily, by choosing $\alpha_t$ and $h_t$ on each iteration to minimize $Z_t$.
- $h_t$ is estimated as a black box, but can we solve for $\alpha_t$?

[Schapire, 1989] This equality isn’t obvious! Can be shown with algebra (telescoping sums)!
Summary: choose \( \alpha_t \) to minimize error bound

[Schapire, 1989]

We can squeeze this bound by choosing \( \alpha_t \) on each iteration to minimize \( Z_t \).

\[
Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y^i h_t(x^i))
\]

\[
\epsilon_t = \sum_{i=1}^{m} D_t(i) \delta(h_t(x^i) \neq y^i)
\]

For boolean Y: differentiate, set equal to 0, there is a closed form solution! [Freund & Schapire ’97]:

\[
\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
\]
**Initialize:** \( D_1(i) = 1/m, \) for \( i = 1, \ldots, m \)

For \( t=1\ldots T \):
- Train base classifier \( h_t(x) \) using \( D_t \)
- Choose \( \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \)
- Update, for \( i=1..m \):
  \[ D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y^i h_t(x^i)) \]

Output final classifier:
\[
H(x) = \text{sign} \left( \sum_{i=1}^{T} \alpha_t h_t(x^i) \right)
\]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ H(x) = \text{sign}(0.35 \times h_1(x)) \]
- \( h_1(x) = +1 \) if \( x_1 > 0.5 \), -1 otherwise

Use decision stubs as base classifier

Initial:
- \( D_1 = [D_1(1), D_1(2), D_1(3)] = [.33,.33,.33] \)

**t=1:**
- Train stub [work omitted, breaking ties randomly]
  - \( h_1(x) = +1 \) if \( x_1 > 0.5 \), -1 otherwise
  - \( \epsilon_1 = \Sigma D_1(i) \delta(h_1(x^i)\neq y^i) \)
    \[ \epsilon_1 = 0.33 \times 1 + 0.33 \times 0 + 0.33 \times 0 = 0.33 \]
  - \( \alpha_1 = (1/2) \ln((1-\epsilon_1)/\epsilon_1) = 0.5 \times \ln(2) = 0.35 \)
  - \( D_2(1) \propto D_1(1) \times \exp(-\alpha_1 y^1 h_1(x^1)) \)
    \[ = 0.33 \times \exp(-0.35 \times 1 \times -1) = 0.33 \times \exp(0.35) = 0.46 \]
  - \( D_2(2) \propto D_1(2) \times \exp(-\alpha_1 y^2 h_1(x^2)) \)
    \[ = 0.33 \times \exp(-0.35 \times -1 \times -1) = 0.33 \times \exp(-0.35) = 0.23 \]
  - \( D_2(3) \propto D_1(3) \times \exp(-\alpha_1 y^3 h_1(x^3)) \)
    \[ = 0.33 \times \exp(-0.35 \times 1 \times 1) = 0.33 \times \exp(-0.35) = 0.23 \]
- \( D_2 = [D_1(1), D_1(2), D_1(3)] = [0.5, 0.25, 0.25] \)

**t=2**
- Continues on next slide!
Initialize: $D_1(i) = 1/m$, for $i = 1, \ldots, m$

For $t=1\ldots T$:

- Train base classifier $h_t(x)$ using $D_t$
- Choose $\alpha_t \leftarrow \frac{1}{m} \sum_{i=1}^{m} D_t(i) \delta(h_t(x^i) \neq y^i)$
- Update, for $i=1..m$:
  
  $D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y^i h_t(x^i))$

Output final classifier:

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

$t=2$:

- Train stub [work omitted; different stub because of new data weights $D$; breaking ties opportunistically (will discuss at end)]
  
  - $h_2(x)=+1$ if $x_1<1.5$, -1 otherwise
  - $\varepsilon_2=\frac{\sum_i D_2(i) \delta(h_2(x^i)\neq y^i)}{\sum_i D_2(i)} = 0.5 \times 0 + 0.25 \times 1 + 0.25 \times 0 = 0.25$
  - $\alpha_2=(1/2) \ln \left( \frac{1-\varepsilon_2}{\varepsilon_2} \right) = 0.5 \times \ln(0.75) = 0.55$
  - $D_2(1) \alpha D_1(1) \exp(-\alpha_2 y^1 h_2(x^1)) = 0.5 \times \exp(-0.55 \times 1 \times 1) = 0.5 \times \exp(-0.55) = 0.29$
  - $D_2(2) \alpha D_1(2) \exp(-\alpha_2 y^2 h_2(x^2)) = 0.25 \times \exp(-0.55 \times -1 \times 1) = 0.25 \times \exp(0.55) = 0.43$
  - $D_2(3) \alpha D_1(3) \exp(-\alpha_2 y^3 h_2(x^3)) = 0.25 \times \exp(-0.55 \times 1 \times 1) = 0.25 \times \exp(-0.55) = 0.14$

$t=3$:

- $D_3 = [D_3(1), D_3(2), D_3(3)] = [0.33, 0.5, 0.17]$

Continues on next slide!

### Table

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$H(x) = \text{sign}(0.35 \times h_1(x) + 0.55 \times h_2(x))$

- $h_1(x)=+1$ if $x_1>0.5$, -1 otherwise
- $h_2(x)=+1$ if $x_1<1.5$, -1 otherwise
**Initialize:**  
\[ D_1(i) = 1/m, \text{ for } i = 1, \ldots, m \]

For \( t = 1 \ldots T \):
- Train base classifier \( h_t(x) \) using \( D_t \)
- Choose \( \alpha_t = \frac{\epsilon_t}{\epsilon_t} \)
  \[ \epsilon_t = \sum_{i=1}^{m} D_t(i) \delta(h_t(x^i) \neq y^i) \]
  \[ \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \]
- Update, for \( i = 1 \ldots m \):
  \[ D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y^i h_t(x^i)) \]

Output final classifier:
\[ H(x) = \text{sign} \left( \sum_{i=1}^{T} \alpha_t h_t(x) \right) \]

---

**t=3:**
- Train stub [work omitted; different stub because of new data weights \( D \); breaking ties opportunistically (will discuss at end)]
  - \( h_3(x) = +1 \) if \( x_1 < -0.5 \), -1 otherwise
- \( \epsilon_3 = \sum_i D_3(i) \delta(h_3(x^i) \neq y^i) = 0.33 \times 0 + 0.5 \times 0 + 0.17 \times 1 = 0.17 \)
- \( \alpha_3 = \frac{1}{2} \ln((1-\epsilon_3)/\epsilon_3) = 0.5 \times \ln(4.88) = 0.79 \)
- Stop!!! How did we know to stop?

---

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**H(X) = sign(0.35xh_1(x)+0.55xh_2(x)+0.79xh_3(x))**
- \( h_1(x) = +1 \) if \( x_1 > 0.5 \), -1 otherwise
- \( h_2(x) = +1 \) if \( x_1 < 1.5 \), -1 otherwise
- \( h_3(x) = +1 \) if \( x_1 < -0.5 \), -1 otherwise
Strong, weak classifiers

- If each classifier is (at least slightly) better than random: $\varepsilon_t < 0.5$

- Another bound on error:

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x^i) \neq y^i) \leq \prod_{t} Z_t \leq \exp \left( -2 \sum_{t=1}^{T} \left( \frac{1}{2} - \varepsilon_t \right)^2 \right)$$

- What does this imply about the training error?
  - Will reach zero!
  - Will get there exponentially fast!

- Is it hard to achieve better than random training error?
Boosting results – Digit recognition

[Schapire, 1989]

- **Boosting:**
  - Seems to be robust to overfitting
  - Test error can decrease even after training error is zero!!!
Boosting generalization error bound

\[
error_{true}(H) \leq error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)
\]

Constants:

- **T**: number of boosting rounds
  - Higher \( T \) \( \rightarrow \) Looser bound, *what does this imply?*

- **d**: VC dimension of weak learner, measures complexity of classifier
  - Higher \( d \) \( \rightarrow \) bigger hypothesis space \( \rightarrow \) looser bound

- **m**: number of training examples
  - more data \( \rightarrow \) tighter bound
Boosting generalization error bound

\[ error_{true}(H) \leq error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right) \]

Constants:

- **Theory does not match practice:**
  - Robust to overfitting
  - Test set error decreases even after training error is zero

- **Need better analysis tools**
  - we’ll come back to this later in the quarter

- more data \(\rightarrow\) tighter bound

[Freund & Schapire, 1996]
Boosting: Experimental Results

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets

[Freund & Schapire, 1996]
AdaBoost.MH on Train (left) and Test (right) data from the repository. [S. Tong and Singer, ML 1999]
Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss:

\[ \ln(1 + \exp(-y^i f(x^i))) \]

Boosting minimizes similar loss function:

\[ \exp(-y^i f(x^i)) \]

Both smooth approximations of 0/1 loss!
Logistic regression and Boosting

Logistic regression:

- Minimize loss fn
  \[ \sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x^i))) \]
- Define
  \[ f(x) = \sum_j w_j x_j \]
  where each feature \( x_j \) is predefined
- Jointly optimize parameters \( w_0, w_1, \ldots w_n \) via gradient ascent.

Boosting:

- Minimize loss fn
  \[ \sum_{i=1}^{m} \exp(-y_i f(x^i)) \]
- Define
  \[ f(x) = \sum_t \alpha_t h_t(x) \]
  where \( h_t(x) \) learned to fit data
- Weights \( \alpha_j \) learned incrementally (new one for each training pass)
What you need to know about Boosting

• Combine weak classifiers to get very strong classifier
  – Weak classifier – slightly better than random on training data
  – Resulting very strong classifier – can get zero training error

• AdaBoost algorithm

• Boosting v. Logistic Regression
  – Both linear model, boosting “learns” features
  – Similar loss functions
  – Single optimization (LR) v. Incrementally improving classification (B)

• Most popular application of Boosting:
  – Boosted decision stumps!
  – Very simple to implement, very effective classifier