CSE546: Ensemble Learning - Bagging and Boosting

Winter 2012

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 $i = 1, 2, \ldots, K$:
  – $T_i \leftarrow$ randomly select $M$ training instances with replacement
  – $h_i \leftarrow$ learn($T_i$) \[ID3, NB, kNN, neural net, \ldots\]

• Now combine the $T_i$ together with uniform voting ($w_i=1/K$ for all $i$)
Bagging Example
CART decision boundary
100 bagged trees

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

- 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 (rewighted) 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 – $\alpha_t$

• **Final classifier:**

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

• **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

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

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 = 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.
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.

Overfitting!!

time = 300
Learning from weighted data

- **Consider a weighted dataset**
  - \( D(i) \) – weight of \( i \)th training example \((x^i, y^i)\)
  - Interpreta\( \)tions:
    - \( i \)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 \(x_i \in X, y_i \in Y = \{-1, +1\}\) 

Initialize \(D_1(i) = 1/m\). 

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

- Train base learner using distribution \(D_t\). 
- Get base classifier \(h_t : X \rightarrow \mathbb{R}\). 
- Choose \(\alpha_t \in \mathbb{R}\). 
- Update: 

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

where \(Z_t\) is a normalization factor 

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

Output the final classifier: 

\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).
\]

How? Many possibilities. Will see one shortly! 

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

\[
\begin{align*}
&y_i h_t(x_i) > 0 \rightarrow h_i \text{ correct} \\
&y_i h_t(x_i) < 0 \rightarrow h_i \text{ wrong} \\
&h_i \text{ correct, } \alpha_t > 0 \rightarrow D_{t+1}(i) < D_t(i) \\
&h_i \text{ wrong, } \alpha_t > 0 \rightarrow D_{t+1}(i) > D_t(i)
\end{align*}
\]

Final Result: linear sum of “base” or “weak” classifier outputs. 

Figure 1: The boosting algorithm AdaBoost.
Given: \((x_1, y_1), \ldots, (x_m, y_m)\)
Initialize \(D_1(i) = 1/m.\)
For \(t = 1, \ldots, T:\)

- Train base learner using distribution \(D_t.\)
- Get base classifier \(h_t : X \to \mathbb{R}.\)
- Choose \(\alpha_t \in \mathbb{R}.\)
- Update:

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

\(\epsilon_t\): error of \(h_t\), weighted by \(D_t\)
- \(0 \leq \epsilon_t \leq 1\)

\(\alpha_t\):
- No errors: \(\epsilon_t = 0 \Rightarrow \alpha_t = \infty\)
- All errors: \(\epsilon_t = 1 \Rightarrow \alpha_t = -\infty\)
- Random: \(\epsilon_t = 0.5 \Rightarrow \alpha_t = 0\)
What $\alpha_t$ to choose for hypothesis $h_t$?

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

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i f(x_i))$$

Where

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

[Schapire, 1989]
What $\alpha_t$ to choose for hypothesis $h_t$?

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

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_i \exp(-y_i f(x_i)) = \prod_t Z_t$$

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 $\alpha_t$?

This equality isn’t obvious! Can be shown with algebra (telescoping sums)!

[Schapire, 1989]
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]:

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$
Strong, weak classifiers

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

• Another bound on error:

\[
\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \prod_t Z_t \leq \exp \left( -2 \sum_{t=1}^{T} (1/2 - \varepsilon_t)^2 \right)
\]

• 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

• **Boosting:**
  – Seems to be robust to overfitting
  – Test error can decrease even after training error is zero!!!

[Schapire, 1989]
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, \textit{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

[Freund & Schapire, 1996]
Boosting generalization error bound

[Freund & Schapire, 1996]

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

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
Boosting: Experimental Results

[Freund & Schapire, 1996]

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets
Logistic Regression as Minimizing Loss

Logistic regression assumes:

\[ P(Y = 1|X) = \frac{1}{1 + \exp(f(x))} \quad f(x) = w_0 + \sum_i w_i h_i(x) \]

And tries to maximize data likelihood, for \( Y = \{-1, +1\} \):

\[ P(y_i|x_i) = \frac{1}{1 + e^{-y_i f(x_i)}} \quad \ln P(D_Y | D_X, w) = \sum_{j=1}^{N} \ln P(y^j | x^j, w) \]

\[ = - \sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i))) \]

Equivalent to minimizing log loss:

\[ \sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i))) \]
Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss:

\[ \sum_{i=1}^{m} \ln(1 + \exp(-y_if(x_i))) \]

Boosting minimizes similar loss function:

\[ \frac{1}{m} \sum_{i} \exp(-y_if(x_i)) = \prod_{t} Z_t \]

\[ \delta(H(x_i) \neq y_i) \]

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 \( x_j \) 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 \( h_t(x_i) \) defined dynamically 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