Machine Learning

10-701, Fall 2015

Ensemble methods
Boosting from Weak Learners

Eric Xing

Lecture 11, October 15, 2015

Reading: Chap. 14.3 C.B book
Weak Learners:
Fighting the bias-variance tradeoff

- Simple (a.k.a. weak) learners e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)

Are good 😊 - Low variance, don’t usually overfit
Are bad 😞 - High bias, can’t solve hard learning problems

- Can we make weak learners always good???
  - No!!! But often yes…
Why boost weak learners?

Goal: Automatically categorize type of call requested
(Collect, Calling card, Person-to-person, etc.)

- Easy to find “rules of thumb” that are “often” correct.
  E.g. If ‘card’ occurs in utterance, then predict ‘calling card’

- Hard to find single highly accurate prediction rule.

- yes I’d like to place a collect call long distance please (Collect)
- operator I need to make a call but I need to bill it to my office (ThirdNumber)
- yes I’d like to place a call on my master card please (CallingCard)
Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn many weak classifiers that are good at different parts of the input space.

- **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!

\[ H(X) = \text{sign}(\sum_{t} \alpha_t h_t(X)) \]

\[ H(X) = h_1(X) + h_2(X) \]
Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn many weak classifiers that are good at different parts of the input space.

- **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 do you ???**
  - force classifiers $h_t$ to learn about different parts of the input space?
  - weigh the votes of different classifiers? $\alpha_t$
Bagging

- Recall decision trees (lecture 3)
  - Pros: interpretable, can handle discrete and continuous features, robust to outliers, low bias, etc.
  - Cons: high variance

- Trees are perfect candidates for ensembles
  - Consider averaging many (nearly) unbiased tree estimators
  - Bias remains similar, but variance is reduced

- This is called **bagging** (bootstrap aggregating) (Breiman, 1996)
  - Train many trees on bootstrapped data, then take average

\[ f(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x) \]

- Bootstrap: statistical term for “roll n-face dice n times”
Random Forest

- Reduce correlation between trees, by introducing randomness

1. For $b = 1, \ldots, B$,
   1. Draw a bootstrap dataset $Z^*$
   2. Learn a tree $f_b(\cdot)$ on $Z^*$, in particular select $m$ features randomly out of $p$ features as candidates before splitting

2. Output:
   - Regression: $f(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x)$
   - Classification: majority vote

- Typically take $m \leq \sqrt{p}$
Rationale: Combination of methods

- There is no algorithm that is always the most accurate

- We can select simple “weak” classification or regression methods and combine them into a single “strong” method

- Different learners use different
  - Algorithms
  - Parameters
  - Representations (Modalities)
  - Training sets
  - Subproblems

- The problem: how to combine them
Boosting [Schapire’89]

- **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 weak hypothesis – $h_t$
  - A strength for this hypothesis – $\alpha_t$

- Final classifier:
  \[ H(X) = \text{sign}(\sum \alpha_t h_t(X)) \]

- **Practically useful, and theoretically interesting**

- **Important issues:**
  - what is the criterion that we are optimizing? (measure of loss)
  - we would like to estimate each new component classifier in the same manner (modularity)
Combination of classifiers

- Suppose we have a family of component classifiers (generating ±1 labels) such as decision stumps:

\[ h(x; \theta) = \text{sign}(wx_k + b) \]

where \( \theta = \{k, w, b\} \)

- Each decision stump pays attention to only a single component of the input vector
Combination of classifiers con’d

- We’d like to combine the simple classifiers additively so that the final classifier is the sign of

\[ \hat{h}(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m) \]

where the “votes” \( \{\alpha_i\} \) emphasize component classifiers that make more reliable predictions than others.

- Important issues:
  - what is the criterion that we are optimizing? (measure of loss)
  - we would like to estimate each new component classifier in the same manner (modularity)
AdaBoost

- **Input:**
  - $N$ examples $S_N = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - a weak base learner $h = h(x, \theta)$

- **Initialize:** equal example weights $w_i = 1/N$ for all $i = 1..N$

- **Iterate for $t = 1 \ldots T$:**
  1. train base learner according to weighted example set $(w_t, x)$ and obtain hypothesis $h_t = h(x, \theta)$
  2. compute hypothesis error $\varepsilon_t$
  3. compute hypothesis weight $\alpha_t$
  4. update example weights for next iteration $w_{t+1}$

- **Output:** final hypothesis as a linear combination of $h_t$
AdaBoost

- At the $k$th iteration we find (any) classifier $h(x; \theta_k^*)$ for which the weighted classification error:

$$
\varepsilon_k = \frac{\sum_{i=1}^{n} W_i^{k-1} I(y_i \neq h(x_i; \theta_k^*))}{\sum_{i=1}^{n} W_i^{k-1}}
$$

is better than chance.
- This is meant to be "easy" --- weak classifier

- Determine how many “votes” to assign to the new component classifier:

$$
\alpha_k = 0.5 \log \left( \frac{(1 - \varepsilon_k)}{\varepsilon_k} \right)
$$

- stronger classifier gets more votes

- Update the weights on the training examples:

$$
W_i^k = W_i^{k-1} \exp \left\{ - y_i a_k h(x_i; \theta_k) \right\}
$$
Boosting Example (Decision Stumps)

$D_1$

$D_2$

$D_3$

$h_1$

$h_2$

$h_3$

$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$
Boosting Example (Decision Stumps)

\[ H_{\text{final}} = \text{sign} \left( 0.42 + 0.65 + 0.92 \right) \]
What is the criterion that we are optimizing? (measure of loss)
Measurement of error

- Loss function:
  \[ \lambda(y, h(x)) \]  
  (e.g. \( I(y \neq h(x)) \))

- Generalization error:
  \[ L(h) = E[\lambda(y, h(x))] \]

- Objective: find \( h \) with minimum generalization error

- Main boosting idea: minimize the empirical error:
  \[ \hat{L}(h) = \frac{1}{N} \sum_{i=1}^{N} \lambda(y_i, h(x_i)) \]
Exponential Loss

- Empirical loss:

\[ \hat{L}(h) = \frac{1}{N} \sum_{i=1}^{N} \lambda(y_i, \hat{h}_m(x_i)) \]

- Another possible measure of empirical loss is

\[ \hat{L}(h) = \sum_{i=1}^{n} \exp\left\{ -y_i \hat{h}_m(x_i) \right\} \]
Exponential Loss

- One possible measure of empirical loss is

\[
\hat{L}(h) = \sum_{i=1}^{n} \exp\{-y_i \hat{h}_m(x_i)\}
\]

\[
= \sum_{i=1}^{n} \exp\{-y_i \hat{h}_{m-1}(x_i) - y_i a_m h(x_i; \theta_m)\}
\]

\[
= \sum_{i=1}^{n} \exp\{-y_i \hat{h}_{m-1}(x_i)\} \exp\{-y_i a_m h(x_i; \theta_m)\}
\]

\[
= \sum_{i=1}^{n} W_i^{m-1} \exp\{-y_i a_m h(x_i; \theta_m)\}
\]

Recall that:

\[
\hat{h}_m(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)
\]

- The combined classifier based on \(m - 1\) iterations defines a weighted loss criterion for the next simple classifier to add.

- Each training sample is weighted by its "classifiability" (or difficulty) seen by the classifier we have built so far.
Linearization of loss function

- We can simplify a bit the estimation criterion for the new component classifiers (assuming $\alpha$ is small)

$$\exp\{- y_i a_m h(x_i; \theta_m)\} \approx 1 - y_i a_m h(x_i; \theta_m)$$

- Now our empirical loss criterion reduces to

$$\sum_{i=1}^{n} \exp\{- y_i \hat{h}_m(x_i)\}$$

$$\approx \sum_{i=1}^{n} W_i^{m-1} (1 - y_i a_m h(x_i; \theta_m))$$

$$= \sum_{i=1}^{n} W_i^{m-1} - a_m \sum_{i=1}^{n} W_i^{m-1} y_i h(x_i; \theta_m)$$

- We could choose a new component classifier to optimize this weighted agreement
A possible algorithm

- At stage $m$ we find $\theta^*$ that maximize (or at least give a sufficiently high) weighted agreement:

$$\sum_{i=1}^{n} W_{i}^{m-1} y_i h(x_i; \theta^*_m)$$

  - each sample is weighted by its "difficulty" under the previously combined $m - 1$ classifiers,
  - more "difficult" samples received heavier attention as they dominates the total loss

- Then we go back and find the “votes” $\alpha^*_m$ associated with the new classifier by minimizing the original weighted (exponential) loss $\hat{L}(h) = \sum_{i=1}^{n} W_{i}^{m-1} \exp\{-y_i a_m h(x_i; \theta_m)\}$

$$\Rightarrow \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

© Eric Xing @ CMU, 2006-2015
The AdaBoost algorithm

- At the $k$th iteration we find (any) classifier $h(x; \theta_k^*)$ for which the weighted classification error:

$$
\varepsilon_k = \frac{\sum_{i=1}^{n} W_i^{k-1} I(y_i \neq h(x_i; \theta_k^*))}{\sum_{i=1}^{n} W_i^{k-1}}
$$

is better than change.

- This is meant to be "easy" --- weak classifier

- Determine how many “votes” to assign to the new component classifier:

$$
\alpha_k = 0.5 \log \left( \frac{(1 - \varepsilon_k)}{\varepsilon_k} \right)
$$

- stronger classifier gets more votes

- Update the weights on the training examples:

$$
W_i^k = W_i^{k-1} \exp \left\{ - y_i a_k h(x_i; \theta_k) \right\}
$$
The AdaBoost algorithm cont’d

- The final classifier after $m$ boosting iterations is given by the sign of

$$\hat{h}(x) = \frac{\alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)}{\alpha_1 + \ldots + \alpha_m}$$

- the votes here are normalized for convenience
Boosting

- We have basically derived a Boosting algorithm that sequentially adds new component classifiers, each trained on reweighted training examples
  - each component classifier is presented with a slightly different problem

- AdaBoost preliminaries:
  - we work with normalized weights $W_i$ on the training examples, initially uniform ($W_i = 1/n$)
  - the weight reflect the "degree of difficulty" of each datum on the latest classifier
AdaBoost: summary

- **Input:**
  - \( N \) examples \( S_N = \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
  - a weak base learner \( h = h(x, \theta) \)

- **Initialize:** equal example weights \( w_i = 1/N \) for all \( i = 1..N \)

- **Iterate for** \( t = 1 \ldots T \):
  1. train base learner according to weighted example set \( (w, x) \) and obtain hypothesis \( h_t = h(x, \theta) \)
  2. compute hypothesis error \( \varepsilon_t \)
  3. compute hypothesis weight \( \alpha_t \)
  4. update example weights for next iteration \( w_{t+1} \)

- **Output:** final hypothesis as a linear combination of \( h_t \)
**Base Learners**

- Weak learners used in practice:
  - Decision stumps (axis parallel splits)
  - Decision trees (e.g. C4.5 by Quinlan 1996)
  - Multi-layer neural networks
  - Radial basis function networks

- Can base learners operate on weighted examples?
  - In many cases they can be modified to accept weights along with the examples
  - In general, we can sample the examples (with replacement) according to the distribution defined by the weights
Boosting results – Digit recognition

- Boosting often, but not always
  - Robust to overfitting
  - Test set error decreases even after training error is zero

[Schapire, 1989]
Generalization Error Bounds

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

- **T** – number of boosting rounds
- **d** – VC dimension of weak learner, measures complexity of classifier
- **m** – number of training examples

<table>
<thead>
<tr>
<th>bias</th>
<th>variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td>small</td>
<td>large</td>
</tr>
</tbody>
</table>

T small
T large

贸易off

[Freund & Schapire’95]

© Eric Xing @ CMU, 2006-2015
Generalization Error Bounds

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

Boosting can overfit if $T$ is large

Boosting often, \textbf{Contradicts experimental results}

- Robust to overfitting
- Test set error decreases even after training error is zero

Need better analysis tools – margin based bounds
Why it is working?

- You will need some learning theory (to be covered in the next two lectures) to understand this fully, but for now let's just go over some high level ideas

- Generalization Error:

  With high probability, Generalization error is less than:

  \[
  \Pr[H(x) \neq y] + \tilde{O} \left( \sqrt{\frac{T d}{m}} \right)
  \]

  As \( T \) goes up, our bound becomes worse,
  Boosting should overfit!
Experiments

The Boosting Approach to Machine Learning, by Robert E. Schapire

© Eric Xing @ CMU, 2006-2015
Training Margins

- When a vote is taken, the more predictors agreeing, the more confident you are in your prediction.

- Margin for example:

\[
\text{margin}_h(x_i, y_i) = y_i \left[ \frac{\alpha_1 h(x_i; \theta_1) + \ldots + \alpha_m h(x_i; \theta_m)}{\alpha_1 + \ldots + \alpha_m} \right]
\]

The margin lies in \([-1, 1]\) and is negative for all misclassified examples.

- Successive boosting iterations improve the majority vote or margin for the training examples.
A Margin Bound

- For any $\gamma$, the generalization error is less than:

$$\Pr(\text{margin}_h(x,y) \leq \gamma) + O\left(\sqrt{\frac{d}{m\gamma^2}}\right)$$


- It does not depend on $T$!!!
Summary

- Boosting takes a weak learner and converts it to a strong one.
- Works by asymptotically minimizing the empirical error.
- Effectively maximizes the margin of the combined hypothesis.
Some additional points for fun
Boosting and Logistic Regression

Logistic regression assumes:

\[ P(Y = 1|X) = \frac{1}{1 + \exp(f(x))} \]

And tries to maximize data likelihood:

\[
P(\mathcal{D}|f) = \prod_{i=1}^{m} \frac{1}{1 + \exp(-y_if(x_i))}
\]

Equivalent to minimizing log loss

\[-\log P(\mathcal{D}|f) = \sum_{i=1}^{m} \ln(1 + \exp(-y_if(x_i)))\]
Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss:

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

$$f(x) = w_0 + \sum_j w_j x_j$$

Boosting minimizes similar loss function!!

$$\frac{1}{m} \sum_{i=1}^{m} \exp(-y_if(x_i)) = \prod_t Z_t$$

$$f(x) = \sum_t \alpha_t h_t(x)$$

Weighted average of weak learners

Both smooth approximations of 0/1 loss!

0/1 loss

log loss

exp loss

© Eric Xing @ CMU, 2006-2015
**Boosting and Logistic Regression**

**Logistic regression:**
- Minimize log loss
  \[ \sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i))) \]
- Define
  \[ f(x) = \sum_{j} w_j x_j \]
  where \( x_j \) predefined features (linear classifier)
- Jointly optimize over all weights \( w_0, w_1, w_2 \ldots \)

**Boosting:**
- Minimize exp loss
  \[ \sum_{i=1}^{m} \exp(-y_i f(x_i)) \]
- Define
  \[ f(x) = \sum_{t} \alpha_t h_t(x) \]
  where \( h_t(x) \) defined dynamically to fit data (not a linear classifier)
- Weights \( \alpha_t \) learned per iteration \( t \) incrementally
Hard & Soft Decision

Weighted average of weak learners

\[ f(x) = \sum_{t} \alpha_t h_t(x) \]

Hard Decision/Predicted label:

\[ H(x) = \text{sign}(f(x)) \]

Soft Decision:
(based on analogy with logistic regression)

\[ P(Y = 1|X) = \frac{1}{1 + \exp(f(x))} \]
Effect of Outliers

**Good ☺**: Can identify outliers since focuses on examples that are hard to categorize

**Bad ☹**: Too many outliers can degrade classification performance dramatically increase time to convergence
Goal: Find nonlinear predictor $\hat{h}(x) \in \mathcal{H}$ such that

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h(X), Y)$$

Gradient boosting generalizes Adaboost (exponential loss) to any smooth loss functions $\mathcal{L}(\cdot, \cdot)$.

**Square loss (regression)**

$$\mathcal{L}(h(X), Y) = \sum_{i=1}^{n} (h(x_i) - y_i)^2$$

**Logistic loss (classification)**

$$\mathcal{L}(h(X), Y) = \sum_{i=1}^{n} \ln(1 + e^{-h(x_i)y_i})$$

**Margin loss (ranking)**

$$\mathcal{L}(h(X), Y) = \sum_{(i, i') : y(i, i') = 1} \max(0, 1 - (h(x_i) - h(x_{i'})))^2$$ (prefer item i over j)

Others…
Let’s use decision tree to approximate \( g_{k-1} \).

A J-leaf node decision tree can be viewed as a partition of the input space

\[
q : \mathbb{R}^d \rightarrow \{1, 2, \ldots, J\}
\]

and a prediction value (weight) associated with each partition

\[
\omega \in \mathbb{R}^J
\]

Will learn \( q \) (tree structure) first, then \( \omega \).