Boosting continued

Machine Learning – CSE546
Carlos Guestrin
University of Washington
October 14, 2013

©Carlos Guestrin 2005-2013

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!

\[
H(x) = \text{Sign} \left( \sum_{t=1}^{T} \omega_t h_t(x) \right)
\]

\[
h_t(x) = \begin{cases} 
+1 & \text{if } x_i = 1 \\
-1 & \text{if } x_i = 0 \\
\text{else} & \text{span}
\end{cases}
\]

- But how do you ???
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?
AdaBoost

- Initialize weights to uniform dist: $D_1(j) = 1/N$
- For $t = 1 \ldots T$
  - Train weak learner $h_t$ on distribution $D_t$ over the data
  - Choose weight $\alpha_t$ for hypothesis $h_t$ this way
  - Update weights:
    $$D_{t+1}(j) = \frac{D_t(j) \exp(-\alpha_t y_j h_t(x_j))}{Z_t}$$
    where $Z_t$ is normalizer:
    $$Z_t = \sum_{j=1}^{N} D_t(j) \exp(-\alpha_t y_j h_t(x_j))$$
- Output final classifier:
  $$H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Why choose $\alpha_t$ for hypothesis $h_t$ this way?

Training error of final classifier is bounded by:

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[H(x_j) \neq y_j] \leq \frac{1}{N} \sum_{j=1}^{N} \exp(-y_j f(x_j))$$

Where $f(x) = \sum_t \alpha_t h_t(x)$; $H(x) = \text{sign}(f(x))$
Why choose $\alpha_t$ for hypothesis $h_t$ this way?

We can minimize this bound by choosing $\alpha_t$ on each iteration to minimize $Z_t$:

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

For boolean target function, this is accomplished by [Freund & Schapire '97]:

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

You'll prove this in your homework! 😊

---

Strong, weak classifiers

- If each classifier is (at least slightly) better than random
  - $\epsilon_t < 0.5$
  - $\epsilon_t < 0.5 - \gamma$ for $\gamma > 0$

  AdaBoost will achieve zero training error (exponentially fast):

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[H(x^j) \neq y^j] \leq \prod_{t=1}^{T} Z_t \leq \exp \left( -2 \sum_{t=1}^{T} (1/2 - \epsilon_t)^2 \right)$$

- Is it hard to achieve better than random training error?
  - $\epsilon_t < 1/2$ implies $\epsilon_t < (1/2 - \epsilon_t)$
  - $\epsilon_t > 1/2$ implies $\epsilon_t > (1/2 - \epsilon_t)$

Extra credit:
- no cheating
- no co-instructors
Boosting results – Digit recognition

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

[Schapire, 1989]

Boosting: Experimental Results

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

[Freund & Schapire, 1996]
Boosting and Logistic Regression

Logistic regression assumes:
\[ P(Y = 1 | X) = \frac{1}{1 + \exp(f(x))} \]
And tries to maximize data likelihood:
\[ \ln P(D|H) = \prod_{j=1}^{N} \frac{1}{1 + \exp(-y_j f(x_j))} \]
Equivalent to minimizing log loss
\[ \sum_{j=1}^{N} \ln(1 + \exp(-y_j f(x_j))) \]
Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss
\[
\sum_{j=1}^{N} \ln(1 + \exp(-y^j f(x^j)))
\]
Boosting minimizes similar loss function!!
\[
\frac{1}{N} \sum_{j=1}^{N} \exp(-y^j f(x^j)) = \prod_{t=1}^{T} Z_t
\]
Both smooth approximations of 0/1 loss!

Logistic regression and Boosting

Logistic regression:
- Minimize loss fn
  \[
  \sum_{j=1}^{N} \ln(1 + \exp(-y^j f(x^j)))
  \]
- Define
  \[
  f(x) = w_0 + \sum_i w_i x_i
  \]
  where features \( x_i \) are predefined
- Weights \( w_i \) are learned in joint optimization

Boosting:
- Minimize loss fn
  \[
  \sum_{j=1}^{N} \exp(-y^j f(x^j))
  \]
- 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 incrementally
What you need to know about Boosting

- Combine weak classifiers to obtain very strong classifier
  - Weak classifier – slightly better than random on training data
  - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Boosting v. Logistic Regression
  - 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

Projects

- An opportunity to exercise what you learned and to learn new things
- Individually or groups of two
- Must involve real data
  - Must be data that you have available to you by the time of the project proposals
- Must involve machine learning
- It's encouraged to be related to your research, but must be something new you did this quarter
  - Not a project you worked on during the summer, last year, etc.
- Sample projects on course website
  - Wed., October 23 at 9:00am: Project Proposals
  - Mon., November 11 at 9:00am: Project Milestone
  - Wed., December 4, 3-5pm: Poster Session
  - Mon., December 9 at 9:00am: Project Report
Decision Trees

Machine Learning – CSE546
Carlos Guestrin
University of Washington
October 16, 2013

©Carlos Guestrin 2005-2013

Linear separability

A dataset is **linearly separable** iff there exists a separating hyperplane:

- Exists \( w \), such that:
  - \( w_0 + \sum_i w_i x_i > 0 \); if \( x = (x_1, ..., x_n) \) is a positive example
  - \( w_0 + \sum_i w_i x_i < 0 \); if \( x = (x_1, ..., x_n) \) is a negative example
Not linearly separable data

- Some datasets are not linearly separable!

\[ x_1 \oplus x_2 \]

Addressing non-linearly separable data – Option 1, non-linear features

- Choose non-linear features, e.g.,
  - Typical linear features: \( w_0 + \sum w_i x_i \)
  - Example of non-linear features:
    - Degree 2 polynomials, \( w_0 + \sum_i w_i x_i + \sum_{ij} w_{ij} x_i x_j \)
  - Classifier \( h_w(x) \) still linear in parameters \( w \)
    - As easy to learn
    - Data is linearly separable in higher dimensional spaces
    - More discussion later this quarter
Addressing non-linearly separable data – Option 2, non-linear classifier

- Choose a classifier $h_w(x)$ that is non-linear in parameters $w$, e.g.,
  - Decision trees, boosting, nearest neighbor, neural networks...
- More general than linear classifiers
- But, can often be harder to learn (non-convex/concave optimization required)
- But, but, often very useful
- (BTW. Later this quarter, we'll see that these options are not that different)

A small dataset: Miles Per Gallon

Suppose we want to predict MPG

From the UCI repository (thanks to Ross Quinlan)
A Decision Stump

Recursion Step

Take the Original Dataset...

And partition it according to the value of the attribute we split on.
Recursion Step

Build tree from These examples.

Build tree from These examples.

Build tree from These examples.

Build tree from These examples.

Second level of tree

Recurisively build a tree from the seven records in which there are four cylinders and the maker was based in Asia

(Similar recursion in the other cases)
Classification of a new example

- Classifying a test example – traverse tree and report leaf label
Are all decision trees equal?

- Many trees can represent the same concept
- But, not all trees will have the same size!
  - e.g., $\phi = A \land B \lor \neg A \land C \ (\text{(A and B) or (not A and C)})$

Learning decision trees is hard!!!

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest ’76]
- Resort to a greedy heuristic:
  - Start from empty decision tree
  - Split on **next best attribute (feature)**
  - Recurse on subset of data consistent with each leaf
Choosing a good attribute

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Measuring uncertainty

- Good split if we are more certain about classification after split
  - Deterministic good (all true or all false)
  - Uniform distribution bad

\[
P(Y=A) = 1/2 \quad P(Y=B) = 1/4 \quad P(Y=C) = 1/8 \quad P(Y=D) = 1/8
\]

\[
P(Y=A) = 1/4 \quad P(Y=B) = 1/4 \quad P(Y=C) = 1/4 \quad P(Y=D) = 1/4
\]
Entropy

Entropy $H(Y)$ of a random variable $Y$

$$H(Y) = - \sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

More uncertainty, more entropy!

Information Theory interpretation: $H(Y)$ is the expected number of bits needed to encode a randomly drawn value of $Y$ (under most efficient code).

Andrew Moore’s Entropy in a nutshell

Low Entropy

High Entropy
Andrew Moore’s Entropy in a nutshell

Information gain

**Advantage of attribute – decrease in uncertainty**
- Entropy of $Y$ before you split
  \[ H(Y) = -\sum_{i=1}^{n} P(Y = y_i) \log P(Y) \]
- Entropy after split
  - Weight by probability of following each branch, i.e., normalized number of records
  \[ H(Y | X) = -\sum_{i=1}^{c} P(X = x_i) \sum_{j=1}^{n} P(Y = y_j | X = x_i) \log P(Y = y_j | X = x_i) \]
- Information gain is difference
  \[ IG(X) = H(Y) - H(Y | X) \]

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Learning decision trees

- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute
  - Split on \( \arg \max_i IG(X_i) = \arg \max_i H(Y) - H(Y | X_i) \)
- Recurse

Look at all the information gains...

Suppose we want to predict MPG
A Decision Stump

mpg values: bad good

root
22 18
pchance = 0.001

cylinders = 3
0 0
Predict bad

cylinders = 4
4 17
Predict good

cylinders = 5
1 0
Predict bad

cylinders = 6
8 0
Predict bad

cylinders = 8
9 1
Predict bad

Base Case

Don't split a node if all matching records have the same output value

Don't split a node if all matching records have the same output value
Base Case Two

Don’t split a node if none of the attributes can create multiple non-empty children.

Base Case Two: No attributes can distinguish
Base Cases

- Base Case One: If all records in current data subset have the same output then don’t recurse
- Base Case Two: If all records have exactly the same set of input attributes then don’t recurse

Base Cases: An idea

- Base Case One: If all records in current data subset have the same output then don’t recurse
- Base Case Two: If all records have exactly the same set of input attributes then don’t recurse

Proposed Base Case 3:
If all attributes have zero information gain then don’t recurse

• Is this a good idea?
The problem with Base Case 3

\[ Y = A \text{ XOR } B \]

The information gains:

<table>
<thead>
<tr>
<th>Input</th>
<th>Value</th>
<th>Distribution</th>
<th>Info Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The resulting bad decision tree:

y values: 0 1

root

2 2

Predict 0

If we omit Base Case 3:

y = a XOR b

The resulting decision tree:

low info gain is not a good stopping criterion

Perfect classification

©Carlos Guestrin 2005-2013
Basic Decision Tree Building
Summarized

BuildTree(DataSet, Output)
- If all output values are the same in DataSet, return a leaf node that says "predict this unique output"
- If all input values are the same, return a leaf node that says "predict the majority output"
- Else find attribute X with highest Info Gain
  - Suppose X has \( n_x \) distinct values (i.e. X has arity \( n_x \)).
    - Create and return a non-leaf node with \( n_x \) children.
    - The \( i \)'th child should be built by calling BuildTree(DS\(_i\), Output)
      Where DS\(_i\) built consists of all those records in DataSet for which \( X = i \)th distinct value of X.