Logistic Regression, cont.

Decision Trees

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University
September 26th, 2007

Logistic Regression

- Learn $P(Y|X)$ directly!
  - Assume a particular functional form
  - Sigmoid applied to a linear function of the data:

$$P(Y = 1|X) = \frac{1}{1 + \exp(-z)}$$

Features can be discrete or continuous!
Loss functions: Likelihood v. Conditional Likelihood

- Generative (Naïve Bayes) Loss function:
  Data likelihood
  \[ \ln P(D \mid w) = \sum_{j=1}^{N} \ln P(x_j, y_j \mid w) \]
  \[ = \sum_{j=1}^{N} \ln P(y_j \mid x_j, w) + \sum_{j=1}^{N} \ln P(x_j \mid w) \]

- Discriminative models cannot compute \( P(x \mid w) \)

- But, discriminative (logistic regression) loss function:
  Conditional Data Likelihood
  \[ \ln P(D_Y \mid D_X, w) = \sum_{j=1}^{N} \ln P(y_j \mid x_j, w) \]

  Doesn’t waste effort learning \( P(X) \) – focuses on \( P(Y \mid X) \) all that matters for classification

Optimizing concave function – Gradient ascent

- Conditional likelihood for Logistic Regression is concave → Find optimum with gradient ascent

  Gradient:
  \[ \nabla_w l(w) = \left( \frac{\partial l(w)}{\partial w_0}, \ldots, \frac{\partial l(w)}{\partial w_n} \right)^T \]

  Update rule:
  \[ \Delta w = \eta \nabla_w l(w) \]
  \[ w_i^{(t+1)} = w_i^{(t)} + \eta \frac{\partial l(w)}{\partial w_i} \]

- Gradient ascent is simplest of optimization approaches
  e.g., Conjugate gradient ascent much better (see reading)
Gradient Descent for LR

Gradient ascent algorithm: iterate until change < ε

\[ w_{0(t+1)} \leftarrow w_{0(t)} + \eta \sum_j [y^j - \hat{P}(Y^j = 1 | x^j, w)] \]

For \( i = 1 \ldots n \),

\[ w_{i(t+1)} \leftarrow w_{i(t)} + \eta \sum_j x_{ij}^j [y^j - \hat{P}(Y^j = 1 | x^j, w)] \]

That's all M(C)LE. How about MAP?

\[ p(w | Y, X) \propto P(Y | X, w)p(w) \]

- One common approach is to define priors on \( w \):
  - Normal distribution, zero mean, identity covariance
  - "Pushes" parameters towards zero
- Corresponds to Regularization
  - Helps avoid very large weights and overfitting
  - More on this later in the semester

MAP estimate

\[ w^* = \arg \max_w \ln \left[ P(w) \prod_{j=1}^N P(y^j | x^j, w) \right] \]
M(C)AP as Regularization

\[ \ln \left[p(w) \prod_{j=1}^{N} P(y^j | \mathbf{x}^j, w) \right] \]

\[ = \ln \frac{1}{\sqrt{2\pi J}} e^{-\frac{w^2}{2J}} \prod_{j=1}^{N} P(y^j | \mathbf{x}^j, w) \]

\[ \approx \ln \prod_{j=1}^{N} p(y^j | \mathbf{x}^j, w) - \frac{1}{2} \sum_{i=1}^{J} w_i^2 + \frac{1}{2} \ln \frac{1}{\sqrt{2J}} \]

Penalizes high weights, also applicable in linear regression

Large parameters → Overfitting

- If data is linearly separable, weights go to infinity
- Leads to overfitting:
  - Penalizing high weights can prevent overfitting… again, more on this later in the semester
Gradient of M(C)AP

\[ \frac{\partial}{\partial w_i} \ln \left[ p(w) \prod_{j=1}^{N} P(y_j \mid x^j, w) \right] \]

\[ p(w) = \prod_{i} \frac{1}{\kappa \sqrt{2\pi}} e^{-\frac{w_i^2}{2\kappa}} \]

\[ \frac{\partial}{\partial w_i} \left[ \ln \prod_{j=1}^{N} P(y_j \mid x^j, w) - \frac{\sum_{i} w_i^2}{2\kappa} \right] \]

\[ = \frac{\partial}{\partial w_i} \ln \prod_{j=1}^{N} P(y_j \mid x^j, w) - \frac{w_i^2}{\kappa} \]

we know from before

MLE vs MAP

- Maximum conditional likelihood estimate

\[ w^* = \arg \max_w \ln \left[ \prod_{j=1}^{N} P(y_j \mid x^j, w) \right] \]

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x^j_i [y^j - \hat{P}(Y^j = 1 \mid x^j, w)] \]

- Maximum conditional a posteriori estimate

\[ w^* = \arg \max_w \ln \left[ p(w) \prod_{j=1}^{N} P(y_j \mid x^j, w) \right] \]

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x^j_i [y^j - \hat{P}(Y^j = 1 \mid x^j, w)] \right\} \]
Generative and Discriminative classifiers
- focuses on setting when GNB leads to linear classifier
  - variance \( \frac{1}{n_i} \) (depends on feature \( i \), not on class \( k \))

Asymptotic comparison (# training examples \( \rightarrow \) infinity)
- when GNB model correct
  - GNB, LR produce identical classifiers
- when model incorrect
  - LR is less biased – does not assume conditional independence
    - therefore LR expected to outperform GNB

Non-asymptotic analysis
- convergence rate of parameter estimates, \( n = \# \) of attributes in \( X \)
  - Size of training data to get close to infinite data solution
    - GNB needs \( O(\log n) \) samples
    - LR needs \( O(n) \) samples
  - GNB converges more quickly to its (perhaps less helpful) asymptotic estimates
Some experiments from UCI data sets

What you should know about Logistic Regression (LR)

- Gaussian Naïve Bayes with class-independent variances representationally equivalent to LR
  - Solution differs because of objective (loss) function
- In general, NB and LR make different assumptions
  - NB: Features independent given class → assumption on $P(X|Y)$
  - LR: Functional form of $P(Y|X)$, no assumption on $P(X|Y)$
- LR is a linear classifier
  - decision rule is a hyperplane
- LR optimized by conditional likelihood
  - no closed-form solution
  - concave → global optimum with gradient ascent
  - Maximum conditional a posteriori corresponds to regularization
- Convergence rates
  - GNB (usually) needs less data
  - LR (usually) gets to better solutions in the limit
Linear separability

- A dataset is **linearly separable** iff $\exists$ a separating hyperplane:
  - $\exists \mathbf{w}$, such that:
    - $w_0 + \sum_i w_i x_i > 0$; if $x = \{x_1, \ldots, x_n\}$ is a positive example
    - $w_0 + \sum_i w_i x_i < 0$; if $x = \{x_1, \ldots, x_n\}$ is a negative example

Not linearly separable data

- Some datasets are **not linearly separable**!
Addressing non-linearly separable data – Option 1, non-linear features

- Choose non-linear features, e.g.,
  - Typical linear features: \( w_0 + \sum_i w_i x_i \)
  - 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 semester

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, neural networks, nearest neighbor,…
- 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 semester, 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

mpg values: bad good

root

pchance = 0.001

cylinders = 3
cylinders = 4
cylinders = 5
cylinders = 6
cylinders = 8

0 0
4 17
1 0
8 0
9 1

Predict bad
Predict good
Predict bad
Predict bad
Predict bad

©Carlos Guestrin 2005-2007
Recursion Step

Take the Original Dataset...

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

Build tree from These records..

Build tree from These records..

Build tree from These records..

Build tree from These records..

Records in which cylinders = 4

Records in which cylinders = 5

Records in which cylinders = 6

Records in which cylinders = 8
Second level of tree

Recursively 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

Announcements

- Pittsburgh won the Super Bowl !!
  - Two years ago...

- Recitation this Thursday
  - Logistic regression, discriminative v. generative
Are all decision trees equal?

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

```
   A
  / \  
 t   f  
 B     C
/ \ 
/   \ 
/     
 t     t
f     f
 t   t
 f  f
```

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
Choosing a good attribute

<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>T</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>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

$P(Y=\text{A}) = 1/2$  $P(Y=\text{B}) = 1/4$  $P(Y=\text{C}) = 1/8$  $P(Y=\text{D}) = 1/8$

$P(Y=\text{A}) = 1/4$  $P(Y=\text{B}) = 1/4$  $P(Y=\text{C}) = 1/4$  $P(Y=\text{D}) = 1/4$
Entropy

Entropy $H(X)$ 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

- Low Entropy: the values (locations of soup) sampled entirely from within the soup bowl.
- High Entropy: the values (locations of soup) unpredictable... almost uniformly sampled throughout our dining room.

Information gain

- Advantage of attribute – decrease in uncertainty
  - Entropy of Y before you split: \( H(Y) = -\frac{1}{2} \log 2 + \frac{1}{2} \log 1 \approx -1 \)
  - Entropy after split:
    - Weight by probability of following each branch, i.e., normalized number of records
    - Information gain is difference: \( IG(X) = H(Y) - H(Y \mid X) \)

<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>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

©Carlos Guestrin 2005-2007
Learning decision trees

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

Suppose we want to predict MPG

Look at all the information gains...
A Decision Stump

mpg values: bad good

root
22 18
pvalue = 0.001

cylinders = 3
cylinders = 4
cylinders = 5
cylinders = 6
cylinders = 8
0 0 4 17 1 0 8 0 9 1
Predict bad Predict good Predict bad Predict bad Predict bad

Base Case One

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 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

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

\[y = a \text{ XOR } b\]

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 decision tree:

\[y \text{ values: } 0, 1\]

root

2 2

Predict 0

If we omit Base Case 3:

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

\[y = a \text{ XOR } b\]

The resulting decision tree:
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 $X$.  

MPG Test set error
The test set error is much worse than the training set error…

...why?

Decision trees & Learning Bias
Decision trees will overfit

- Standard decision trees are have no learning biased
  - Training set error is always zero!
    - (If there is no label noise)
  - Lots of variance
  - Will definitely overfit!!!
  - Must bias towards simpler trees
- Many strategies for picking simpler trees:
  - Fixed depth
  - Fixed number of leaves
  - Or something smarter…
A chi-square test

Suppose that mpg was completely uncorrelated with maker.
What is the chance we’d have seen data of at least this apparent level of association anyway?

By using a particular kind of chi-square test, the answer is 7.2%

(Such simple hypothesis tests are very easy to compute, unfortunately, not enough time to cover in the lecture, but in your homework, you’ll have fun! :))
Using Chi-squared to avoid overfitting

- Build the full decision tree as before
- But when you can grow it no more, start to prune:
  - Beginning at the bottom of the tree, delete splits in which $p_{\text{chance}} > \text{MaxPchance}$
  - Continue working your way up until there are no more prunable nodes

$\text{MaxPchance}$ is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise

Pruning example

- With MaxPchance = 0.1, you will see the following MPG decision tree:

Note the improved test set accuracy compared with the unpruned tree

<table>
<thead>
<tr>
<th>Set</th>
<th>Num Errors</th>
<th>Size</th>
<th>Percent Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>5</td>
<td>40</td>
<td>12.50</td>
</tr>
<tr>
<td>Test</td>
<td>56</td>
<td>352</td>
<td>15.91</td>
</tr>
</tbody>
</table>

©Carlos Guestrin 2005-2007
MaxPchance

- Technical note MaxPchance is a regularization parameter that helps us bias towards simpler models

![Expected Test set Error vs MaxPchance](image)

- We'll learn to choose the value of these magic parameters soon!

Real-Valued inputs

- What should we do if some of the inputs are real-valued?

<table>
<thead>
<tr>
<th>mpg</th>
<th>cylinders</th>
<th>displacement</th>
<th>horsepower</th>
<th>weight</th>
<th>acceleration</th>
<th>modelyear</th>
<th>maker</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>4</td>
<td>75</td>
<td>18.2</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>6</td>
<td>196</td>
<td>13</td>
<td>70</td>
<td>america</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>110</td>
<td>12.8</td>
<td>77</td>
<td>europe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>6</td>
<td>198</td>
<td>16.5</td>
<td>74</td>
<td>america</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>4</td>
<td>103</td>
<td>16.5</td>
<td>73</td>
<td>asia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bad</td>
<td>8</td>
<td>302</td>
<td>12.8</td>
<td>78</td>
<td>america</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Infinite number of possible split values!!!

Finite dataset, only finite number of relevant splits!

Idea One: Branch on each possible real value
“One branch for each numeric value” idea:

Hopeless: with such high branching factor will shatter the dataset and overfit

Threshold splits

- Binary tree, split on attribute X
  - One branch: $X < t$
  - Other branch: $X \geq t$
Choosing threshold split

- Binary tree, split on attribute X
  - One branch: X < t
  - Other branch: X ≥ t
- Search through possible values of t
  - Seems hard!!!
- But only finite number of t's are important
  - Sort data according to X into {x₁,…,xₘ}
  - Consider split points of the form xᵢ + (xᵢ₊₁ – xᵢ)/2

A better idea: thresholded splits

- Suppose X is real valued
- Define \( IG(Y|X:t) \) as \( H(Y) - H(Y|X:t) \)
- Define \( H(Y|X:t) = H(Y|X < t) P(X < t) + H(Y|X >= t) P(X >= t) \)
- \( IG(Y|X:t) \) is the information gain for predicting Y if all you know is whether X is greater than or less than t
- Then define \( IG^*(Y|X) = \max_t IG(Y|X:t) \)
- For each real-valued attribute, use \( IG^*(Y|X) \) for assessing its suitability as a split
- Note, may split on an attribute multiple times, with different thresholds
Example with MPG

Information gains using the training set (40 records)

<table>
<thead>
<tr>
<th>Input</th>
<th>Value</th>
<th>Distribution</th>
<th>Info Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>cylinders</td>
<td>&lt; 5</td>
<td>blue</td>
<td>0.0286</td>
</tr>
<tr>
<td></td>
<td>&gt;= 5</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>displacement</td>
<td>&lt; 108</td>
<td>blue</td>
<td>0.428206</td>
</tr>
<tr>
<td></td>
<td>&gt;= 108</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>horsepower</td>
<td>&lt; 94</td>
<td>blue</td>
<td>0.0209</td>
</tr>
<tr>
<td></td>
<td>&gt;= 94</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>&lt; 2709</td>
<td>blue</td>
<td>0.378471</td>
</tr>
<tr>
<td></td>
<td>&gt;= 2709</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>acceleration</td>
<td>&lt; 18.2</td>
<td>blue</td>
<td>0.159882</td>
</tr>
<tr>
<td></td>
<td>&gt;= 18.2</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>modelyear</td>
<td>&lt; 81</td>
<td>blue</td>
<td>0.310193</td>
</tr>
<tr>
<td></td>
<td>&gt;= 81</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>maker</td>
<td>america</td>
<td>blue</td>
<td>0.0437296</td>
</tr>
<tr>
<td></td>
<td>Asia</td>
<td>red</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Europe</td>
<td>blue</td>
<td></td>
</tr>
</tbody>
</table>

Example tree using reals

mpg values: bad good

root
  22 18
  pchance = 0.306

  cylinders < 5
    4 17
    pchance = 0.001

  cylinders >= 5
    18 1
    pchance = 0.003

  horsepower < 94
    1 17
    pchance = 0.274

  horsepower >= 94
    5 3
    pchance = 0.274

  acceleration < 18
    18 0
    pchance = 0.1

  acceleration >= 18
    6 1
    pchance = 0.1

  maker = america
    9 10
    pchance = 0.278

  maker = asia
    0 5
    pchance = 0.278

  maker = europe
    1 3
    pchance = 0.278

  displacement < 116
    6 2
    pchance = 0.278

  displacement >= 116
    1 0
    pchance = 0.278
What you need to know about decision trees

- Decision trees are one of the most popular data mining tools
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap (to solve heuristically)
- Information gain to select attributes (ID3, C4.5, …)
- Presented for classification, can be used for regression and density estimation too
- Decision trees will overfit!!!
  - Zero bias classifier → Lots of variance
  - Must use tricks to find “simple trees”, e.g.,
    - Fixed depth/Early stopping
    - Pruning
    - Hypothesis testing

Acknowledgements

- Some of the material in the decision trees presentation is courtesy of Andrew Moore, from his excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)