CSC 2515: Lecture 03: Non-parametric Methods

Richard Zemel & Raquel Urtasun

University of Toronto

Oct 1, 2015
Non-parametric models

- distance
- non-linear decision boundaries
Classification: Oranges and Lemons

![scatter plot of orange and lemon heights and widths](image)
Can construct simple linear decision boundary:

$$y = \text{sign}(w_0 + w_1x_1 + w_2x_2)$$
What is the meaning of "linear" classification

- Classification is intrinsically non-linear
  - It puts non-identical things in the same class, so a difference in the input vector sometimes causes zero change in the answer
- **Linear classification** means that the part that adapts is linear (just like linear regression)
  \[ z(x) = \mathbf{w}^T \mathbf{x} + w_0 \]
  with adaptive \( \mathbf{w}, w_0 \)
- The adaptive part is followed by a non-linearity to make the decision
  \[ y(x) = f(z(x)) \]
- What functions \( f() \) have we seen so far in class?
Classification as Induction

The diagram shows a scatter plot with data points representing heights and widths of objects. The points are categorized into two groups: oranges (circles) and lemons (triangles). The plot aims to illustrate the classification process where non-parametric methods are used to distinguish between the two categories based on their dimensions.
Instance-based Learning

- Alternative to parametric model is non-parametric
- Simple methods for approximating discrete-valued or real-valued target functions (classification or regression problems)
- Learning amounts to simply storing training data
- Test instances classified using similar training instances
- Embodies often sensible underlying assumptions:
  - Output varies smoothly with input
  - Data occupies sub-space of high-dimensional input space
Nearest Neighbors

- Assume training examples correspond to points in d-dimensional Euclidean space
- Target function value for new query estimated from known value of nearest training example(s)
- Distance typically defined to be Euclidean:

\[
\| \mathbf{x}^{(a)} - \mathbf{x}^{(b)} \|^2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}
\]

- Algorithm
  1. find example \((\mathbf{x}^*, t^*)\) closest to the test instance \(\mathbf{x}^{(q)}\)
  2. output \(y^{(q)} = t^*\)

- Note: we don’t need to compute the square root. Why?
Nearest Neighbors Decision Boundaries

- Nearest neighbor algorithm does not explicitly compute decision boundaries, but these can be inferred.

- Decision boundaries: Voronoi diagram visualization
  - show how input space divided into classes
  - each line segment is equidistant between two points of opposite classes
k Nearest Neighbors

- Nearest neighbors sensitive to mis-labeled data ("class noise") → smooth by having k nearest neighbors vote
- Algorithm:
  1. find k examples \( \{x^{(i)}, t^{(i)}\} \) closest to the test instance \( x \)
  2. classification output is majority class

\[
y = \arg \max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})
\]
Some attributes have larger ranges, so are treated as more important
  ▶ normalize scale

Irrelevant, correlated attributes add noise to distance measure
  ▶ eliminate some attributes
  ▶ or vary and possibly adapt weight of attributes

Non-metric attributes (symbols)
  ▶ Hamming distance

Brute-force approach: calculate Euclidean distance to test point from each stored point, keep closest: \( O(dn^2) \). We need to reduce computational burden:
  1. Use subset of dimensions
  2. Use subset of examples
    ▶ Remove examples that lie within Voronoi region
    ▶ Form efficient search tree (kd-tree), use Hashing (LSH), etc
Decision Boundary K-NN

![Graph showing decision boundaries in a scatter plot with height (cm) and width (cm) axes.](image)
**K-NN Summary**

- Single parameter \((k)\) → how do we set it?
- Naturally forms complex decision boundaries; adapts to data density
- Problems:
  - Sensitive to class noise.
  - Sensitive to dimensional scales.
  - Distances are less meaningful in high dimensions
  - Scales with number of examples
- Inductive Bias: What kind of decision boundaries do we expect to find?
Today (Part 2)

- Decision Trees
  - entropy
  - information gain
Another Classification Idea

- We could view the decision boundary as being the composition of several simple boundaries.
Decision Tree: Example

- **width > 6.5cm?**
  - Yes
  - **height > 9.5cm?**
    - Yes (lemon)
    - No (orange)
  - No
    - **height > 6.0cm?**
      - Yes (lemon)
      - No (orange)
Decision Trees

- Internal nodes test attributes
- Branching is determined by attribute value
- Leaf nodes are outputs (class assignments)
- In general, a decision tree can represent any binary function
Choose an attribute on which to descend at each level.
Condition on earlier (higher) choices.
Generally, restrict only one dimension at a time.
Declare an output value when you get to the bottom.
In the orange/lemon example, we only split each dimension once, but that is not required.

**How do you construct a useful decision tree?**

We use **information theory** to guide us.
Two Binary Sequences

Sequence 1:
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ... ?

Sequence 2:
0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?

versus

16

0 1

versus

8 10

0 1
Quantifying Uncertainty

\[ H(X) = - \sum_{x \in X} p(x) \log_2 p(x) \]

- \frac{8}{9} \log_2 \frac{8}{9} - \frac{1}{9} \log_2 \frac{1}{9} \approx \frac{1}{2}

- \frac{4}{9} \log_2 \frac{4}{9} - \frac{5}{9} \log_2 \frac{5}{9} \approx 0.99

- How surprised are we by a new value in the sequence?
- How much information does it convey?
Quantifying Uncertainty: Shannon Entropy

\[ H(X) = - \sum_{x \in X} p(x) \log_2 p(x) \]

- Shannon Entropy is an extremely powerful concept.
- It tells you how much you can compress your data!
Choose an attribute on which to descend at each level.

Condition on earlier (higher) choices

Generally, restrict only one dimension at a time.

How do you construct a useful decision tree?

We use information theory to guide us.
Entropy of a Joint Distribution

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

\[
H(X, Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x, y)
\]

\[
= \frac{24}{100} \log_2 \frac{24}{100} - \frac{1}{100} \log_2 \frac{1}{100} - \frac{25}{100} \log_2 \frac{25}{100} - \frac{50}{100} \log_2 \frac{50}{100}
\]

\[
\approx 1.56 \text{bits}
\]
Specific Conditional Entropy

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

What is the entropy of cloudiness, given that it is raining?

\[
H(X | Y = y) = - \sum_{x \in X} p(x | y) \log_2 p(x | y)
\]

\[
= - \frac{24}{25} \log_2 \frac{24}{25} - \frac{1}{25} \log_2 \frac{1}{25}
\]

\[
\approx 0.24 \text{bits}
\]
The expected conditional entropy:

\[ H(X|Y) = \sum_{y \in Y} p(y) H(X|Y = y) \]

\[ = - \sum_{y \in Y} \sum_{x \in X} p(x, y) \log_2 p(x|y) \]
(Non-Specific) Conditional Entropy

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

\[
H(X|Y) = \sum_{y \in Y} p(y)H(X|Y = y)
\]

\[
= \frac{1}{4}H(\text{clouds|is raining}) + \frac{3}{4}H(\text{clouds|not raining})
\]

\[
\approx 0.75 \text{ bits}
\]
### Information Gain

<table>
<thead>
<tr>
<th></th>
<th>Cloudy</th>
<th>Not Cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>24/100</td>
<td>1/100</td>
</tr>
<tr>
<td>Not Raining</td>
<td>25/100</td>
<td>50/100</td>
</tr>
</tbody>
</table>

How much information about cloudiness do we get by discovering whether it is raining?

\[
IG(X|Y) = H(X) - H(X|Y) 
\]

\[
\approx 0.25 \text{ bits}
\]

- Also called **information gain** in $X$ due to $Y$
- For decision trees, $X$ is the class/label and $Y$ is an attribute
I made the fruit data partitioning just by eyeballing it.

We can use the information gain to automate the process.

At each level, one must choose:

1. Which variable to split.
2. Possibly where to split it.

Choose them based on how much information we would gain from the decision!
Decision Tree Algorithm

- Simple, greedy, recursive approach, builds up tree node-by-node

1. pick an attribute to split at a non-terminal node
2. split examples into groups based on attribute value
3. for each group:
   - if no examples – return majority from parent
   - else if all examples in same class – return class
   - else loop to step 1
### Decision Tree Example: Data

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Full</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
</tr>
<tr>
<td>X₁¹</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
</tr>
<tr>
<td>X₁²</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
</tr>
</tbody>
</table>

Russell & Norvig example
Attribute Selection

\[ IG(Y) = H(X) - H(X|Y) \]

\[ IG(\text{type}) = 1 - \left[ \frac{2}{12} H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} H\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} H\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \]

\[ IG(\text{Patrons}) = 1 - \left[ \frac{2}{12} H(0, 1) + \frac{4}{12} H(1, 0) + \frac{6}{12} H\left(\frac{2}{6}, \frac{4}{6}\right) \right] \approx 0.541 \]
Which Tree is Better?

- **Patrons?**
  - None: No
  - Some: Yes
  - Full: Wait Estimate?
    - >60: No
    - 30-60: Yes
    - 10-30: Alternate?
      - No: Reservation?
        - No: Bar?
          - No: No
          - Yes: Yes
        - Yes: Yes
      - Yes: Fri/Sat?
        - Yes: Yes
        - No: Yes
    - 0-10: Hungry?
      - Yes: Yes
      - No: Alternate?
        - No: Reservation?
          - Yes: Yes
          - No: Yes
        - Yes: Fri/Sat?
          - No: No
          - Yes: Yes
      - Yes: Yes

- **Hungry?**
  - No: Yes
  - Yes: Type?
    - French: Yes
    - Italian: No
    - Thai: No
    - Burger: Fri/Sat?
      - Yes: Yes
      - No: No

- **Alternate?**
  - Yes: Yes
  - No: Yes
What Makes a Good Tree?

- Not too small: need to handle important but possibly subtle distinctions in data
- Not too big:
  - Computational efficiency (avoid redundant, spurious attributes)
  - Avoid over-fitting training examples
- Occam’s Razor: find the simplest hypothesis (smallest tree) that fits the observations
- Inductive bias: small trees with informative nodes near the root
Problems:

- You have exponentially less data at lower levels.
- Too big of a tree can overfit the data.
- Greedy algorithms don’t necessarily yield the global optimum.

In practice, one often regularizes the construction process to try to get small but highly-informative trees.

Decision trees can also be used for regression on real-valued outputs, but it requires a different formalism.
Comparison to k-NN

K-Nearest Neighbors
- Decision boundaries: piece-wise
- Test complexity: non-parametric, few parameters besides (all?) training examples

Decision Trees
- Decision boundaries: axis-aligned, tree structured
- Test complexity: attributes and splits
Applications of Decision Trees

- Can express any Boolean function, but most useful when function depends critically on few attributes
- Bad on: parity, majority functions; also not well-suited to continuous attributes
- Practical Applications:
  - Flight simulator: 20 state variables; 90K examples based on expert pilot’s actions; auto-pilot tree
  - Yahoo Ranking Challenge
  - Random Forests