CPSC 340: Machine Learning and Data Mining

Decision Trees
Fall 2016
Admin

• **Assignment 1** is out, due September 23\(^{rd}\).
  – You’ll need a CS undergrad account to use Handin:
    • [https://www.cs.ubc.ca/getacct](https://www.cs.ubc.ca/getacct)
    – It’s due after the add/drop deadline but **START THIS WEEK**.
    • It will give you an idea of the workload/background expected.

• You can get help on **Piazza**.

• Tutorials start today:
  – Monday 4-5 and 5-6, Tuesday 4:30-5:30, Wednesday 9-10.

• Office hours:
  – Tuesdays at 2-3 (ICICS 146) and 3:30-4:30 (DLC Table 4), and Wednesdays 4-5 (ICICS X337).
Last Time: Data Representation and Exploration

• We discussed **object-feature representation**:
  - **Examples**: another name we’ll use for objects.

• We discussed **summary statistics** and **visualizing data**.

<table>
<thead>
<tr>
<th>Age</th>
<th>Job?</th>
<th>City</th>
<th>Rating</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Yes</td>
<td>Van</td>
<td>A</td>
<td>22,000.00</td>
</tr>
<tr>
<td>23</td>
<td>Yes</td>
<td>Bur</td>
<td>BBB</td>
<td>21,000.00</td>
</tr>
<tr>
<td>22</td>
<td>No</td>
<td>Van</td>
<td>CC</td>
<td>0.00</td>
</tr>
<tr>
<td>25</td>
<td>Yes</td>
<td>Sur</td>
<td>AAA</td>
<td>57,000.00</td>
</tr>
</tbody>
</table>

http://www.statcrunch.com/5.0/viewresult.php?resid=1024581
http://cdn.okcdn.com/blog/humanexperiments/looks-v-personality.png
Motivating Example: Food Allergies

• You frequently start getting an upset stomach

• You suspect an adult-onset food allergy.

http://www.cliparthut.com/upset-stomach-clipart-cn48e5.html
Motivating Example: Food Allergies

• To solve the mystery, you start a food journal:

<table>
<thead>
<tr>
<th></th>
<th>Egg</th>
<th>Milk</th>
<th>Fish</th>
<th>Wheat</th>
<th>Shellfish</th>
<th>Peanuts</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td>0</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>1.2</td>
<td>0</td>
<td>0.10</td>
<td>0.01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0</td>
<td>1.2</td>
<td>0.3</td>
<td>0.10</td>
<td>0.01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• But it’s hard to find the pattern:
  – You can’t isolate and only eat one food at a time.
  – You may be allergic to more than one food.
  – The quantity matters: a small amount may be ok.
  – You may be allergic to specific interactions.
Supervised Learning

- We can formulate this as **supervised learning**:

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>Fish</th>
<th>Wheat</th>
<th>Shellfish</th>
<th>Peanuts</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td>0</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>0.01</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>1.2</td>
<td>0</td>
<td>0.10</td>
<td>0.01</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.3</td>
<td>0</td>
<td>1.2</td>
<td>0.3</td>
<td>0.10</td>
<td>0.01</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- Input for an **object** (day of the week) is a set of **features** (quantities of food).
- Output is a desired **class label** (whether or not we got sick).
- Goal of **supervised learning**:
  - Use data to write a program mapping from features to labels.
  - Program predicts whether foods will make you sick (even with new combinations).
Supervised Learning

• With discrete labels, supervised learning is called classification.
  – But we’re not particularly interested in food allergies.
• Instead we’re interested in studying the concept of supervised learning:
  – Take features of objects and corresponding labels as inputs.
  – Output a program that can predict the label of a generic object.
• This is the most successful machine learning technique:
  – Spam filtering, optical character recognition, Microsoft Kinect, speech recognition, classifying tumours, etc.
• Most useful when:
  – You don’t know how to write a program to do the task.
  – But you have input/output examples.
• Today we will learn about one approach:
  – Decision trees.
But first....

• What types of **preprocessing** might we do?
  – **Data cleaning**: check for and fix missing/unreasonable values.
  – **Summary statistics**:
    • Can help identify “unclean” data.
    • Correlation might reveal an obvious dependence (“sick” ⇔ “peanuts”).
  – **Data transformations**:
    • Convert everything to same scale? (e.g., grams)
    • Add foods from day before? (maybe “sick” depends on multiple days)
    • Add date? (maybe what makes you “sick” changes over time).
  – **Data visualization**: look at a scatterplot of each feature and the label.
    • Maybe the visualization will show something weird in the features.
    • Maybe the pattern is really obvious!

• What you do might depend on how much data you have:
  – Very little data:
    • Represent food by common allergic ingredients (lactose, gluten, etc.)?
  – Lots of data:
    • Use more fine-grained features (bread from bakery vs. hamburger bun)?
Decision Trees

- **Decision trees** are simple programs consisting of:
  - A nested sequence of “if-else” decisions based on the features (splitting rules).
  - A class label as a return value at the end of each sequence.

- **Example decision tree:**

  ```java
  if (milk > 0.5)
      { return 'sick' }
  else
      { if (egg > 1)
          return 'sick'
        else
            return 'not sick'
      }
  ```

  Can draw sequences of decisions as a tree:
Decision Tree Learning

• It might be hard to find a good decision tree by hand.
  – There could be a huge number of variables.
  – Sequences of rules might be hard to find.

• Decision tree learning:
  – Use the data to automatically write the decision tree program.

• Basic idea: search over trees for the “best” tree.
Learning A Decision Stump

• We’ll start **decision stumps**:
  – Simple decision tree with 1 splitting rule based on thresholding 1 feature.

![Decision Stump Diagram]

• How do we find the best “rule” (i.e., the feature and threshold)?
  1. Define a ‘score’ for the rule.
  2. Search for the rule with the best score.
Decision Stump: Accuracy Score

• Most intuitive score: classification accuracy.
  – “If we use this rule, how many objects do we label correctly?”

• Computing classification accuracy for (egg > 1):
  – Find most common labels if we use this rule:
    • When (egg > 1), we were “sick” both times.
    • When (egg <= 1), we were “not sick” three out of four times.
  – Compute accuracy:
    • Rule (egg > 1) is correct on 5/6 objects.

• Scores of other rules:
  – (milk > 0.5) obtains lower accuracy of 4/6.
  – (egg > 0) obtains optimal accuracy of 6/6.
  – () obtains “baseline” accuracy of 3/6, as does (egg > 2).
Decision Stump: Rule Search (Attempt 1)

• Accuracy “score” evaluates quality of a rule.
  – Find the best rule by maximizing score.

• Attempt 1 (exhaustive search):

<table>
<thead>
<tr>
<th>Compute score of (egg &gt; 0)</th>
<th>Compute score of (milk &gt; 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute score of (egg &gt; 0.01)</td>
<td>Compute score of (milk &gt; 0.01)</td>
</tr>
<tr>
<td>Compute score of (egg &gt; 0.02)</td>
<td>Compute score of (milk &gt; 0.02)</td>
</tr>
<tr>
<td>Compute score of (egg &gt; 0.03)</td>
<td>Compute score of (milk &gt; 0.03)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Compute score of (egg &gt; 99.99)</td>
<td>Compute score of (milk &gt; 0.99)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

• As you go, keep track of the highest score.
• Return highest-scoring rule.
Cost of Decision Stumps (Attempt 1)

• How much does this cost?

• Assume we have:
  – ‘n’ objects (days that we measured).
  – ‘d’ features (foods that we measured).
  – ‘t’ thresholds (>0, >0.01, >0.02,...)

• Computing the score of one rule costs $O(n)$:
  – We need to go through all ‘n’ examples.
  – If you are not familiar with “$O(n)$” see notes on webpage.

• To compute scores for $d*t$ rules, total cost is $O(ndt)$.

• Can we do better?
Speeding up Rule Search

• We can ignore rules outside feature ranges:
  – E.g., we never have (egg > 50) in our data.
  – These rules can never improve accuracy.
  – Restrict thresholds to range of features.

• Most of the thresholds give the same score.
  – If we never have (0.5 < egg < 1) in the data,
    • then (egg < 0.6) and (egg < 0.9) have the same score.
  – Restrict thresholds to values in data.
Decision Stump: Rule Search (Attempt 2)

• Attempt 2 (search only over features in data):
  
  Compute score of (eggs > 0)  Compute score of (milk > 0.5)  …
  Compute score of (eggs > 1)  Compute score of (milk > 0.7)  …
  Compute score of (eggs > 2)  Compute score of (milk > 1)   …
  Compute score of (eggs > 3)  Compute score of (milk > 1.25) …
  Compute score of (eggs > 4)  …

• Now at most ‘n’ thresholds for each feature.
• We only consider $O(nd)$ rules instead of $O(dt)$ rules:
  – Total cost changes from $O(ndt)$ to $O(n^2d)$. 
Supervised Learning Notation

• Standard **supervised learning notation**:
  
  - Feature matrix ‘$X$’ has rows as objects, columns as features.
    - $X_{ij}$ is feature ‘$j$’ for object ‘$i$’.
    - E.g., $X_{ij}$ is quantity of food ‘$j$’ on day ‘$i$’.
  
  - Label vector ‘$y$’ contains the labels of the objects.
    - $y_i$ is the label of object ‘$i$’.

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>Fish</th>
<th>Wheat</th>
<th>Shellfish</th>
<th>Peanuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td>0</td>
<td>0.3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>1.2</td>
<td>0</td>
<td>0.10</td>
<td>0.01</td>
</tr>
<tr>
<td>0.3</td>
<td>0</td>
<td>1.2</td>
<td>0.3</td>
<td>0.10</td>
<td>0.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Decision Stump Learning Pseudo-Code

Input: \textbf{feature matrix} $X$ and \textbf{label vector} $y$

for each feature '$j$'

for each example '$i$'

set \textbf{threshold} to feature '$j$' in example '$i$'!

find \textbf{mode} of label vector when feature '$j$' is \textbf{above} threshold

find \textbf{mode} of label vector when feature '$j$' is \textbf{below} threshold.

\textbf{classify} all examples based on threshold

\textbf{count} the \textbf{number of errors}.

\textbf{store} this rule if it has the lowest error so far.

Output: an \textbf{optimal} decision stump rule
Input: \textbf{feature matrix} $X$ and \textbf{label vector} $y$

$[n, d] = \text{size}(X)$

$\minError = \sum(y \overset{\wedge}{=} \text{mode}(y))$

$\minRule = []$

for $j = 1 : d$
  for $i = 1 : n$
    $t = X(i, j)$
    $y_{above} = \text{mode}(y(X(:, j) > t))$
    $y_{below} = \text{mode}(y(X(:, j) \leq t))$
    $\hat{y} = y_{above} * \text{ones}(n, 1)$
    $\text{error} = \sum(\hat{y} \overset{\wedge}{=} y)$
    if $\text{error} < \minError$
      $\minError = \text{error}$
      $\minRule = [j, t]$
  end
end

\text{Compute error if you don't split.}

For each feature 'j':

for each example 'i' set threshold to feature 'j' in

find mode of label vector when $X(:, j) > t$
find mode of label vector when $X(:, j) \leq t$

Classify all examples based on threshold count the number of errors.

Store this rule if it has the lowest
Decision Stump: Rule Search (Attempt 3)

• Do we have to compute score from scratch?
  – Rule (egg > 1) and (egg > 2) have same decisions, except when (egg == 2).
  – Sort the examples based on ‘egg’.
  – Go through the rules in order, updating the score.

• Sorting costs $O(n \log n)$ per feature.

• You do at most $O(n)$ score updates per feature.

• Total cost is reduced from $O(n^2d)$ to $O(nd \log n)$.

• This is a good runtime:
  – $O(nd)$ is the size of data, same as runtime up to a log factor.
  – We can apply this algorithm to huge datasets.
Decision Tree Learning

- **Decision stumps** have only 1 rule based on only 1 feature.
  - Very limited class of models: usually not very accurate for most tasks.

- **Decision trees** allow **sequences of splits** based on multiple features.
  - Very general class of models: can get very high accuracy.
  - However, it’s **computationally infeasible to find the best decision tree**.

- Most common decision tree learning algorithm in practice:
  - **Greedy recursive splitting**.
Example of Greedy Recursive Splitting

• Start with the full dataset:

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0.3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Find the decision stump with the best score:

milk > 0.5

no

yes

Sick?

not sick

Split into two smaller datasets based on stump:

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0.3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Greedy Recursive Splitting

We now have a decision stump and two datasets:

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0.3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Egg</th>
<th>Milk</th>
<th>...</th>
<th>Sick?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Split the leaves by fitting a decision stump to each dataset:
Greedy Recursive Splitting

Splitting the leaves gives a “depth 2” decision tree:

We can then split the training examples into 4 datasets, and recurse on these...
Greedy Recursive Splitting

A “depth 3” decision tree:

Typically we continue splitting until:
- The leaves only has one label.
- We reach a user-defined maximum depth.
Discussion of Decision Tree Learning

• Advantages:
  – Interpretable.
  – Fast to learn.
  – Very fast to classify

• Disadvantages:
  – Hard to find optimal set of rules.
  – Greedy splitting uses very simple rules.
  – Unless very deep, greedy splitting often not accurate.

• Issues:
  – Can you revisit a feature?
    • Yes, knowing other information could make feature relevant again.
  – More complicated rules?
    • Yes, but searching for the best rule gets much more expensive.
  – Is accuracy the best score?
    • No, there may no split that increase accuracy. Alternative: information gain.
  – What depth?
Summary

• **Supervised learning:**
  – using data to write a program based on input/output examples.

• **Decision trees:** predicting a label using a sequence of simple rules.

• **Decision stumps:** simple decision tree that is very fast to fit.

• **Greedy recursive splitting:** uses a sequence of stumps to fit a tree.
  – Very fast and interpretable, but not always the most accurate.

• Next time: the most important ideas in machine learning.
Bonus Slide: Can you re-visit a feature?

- Yes.

Knowing (ice cream > 0.3) makes small milk quantities relevant.
Bonus Slide: Can you have more complicated rules?

- Yes:

  \[ \text{milk} + \text{ice cream} > 0.5 \]

  - no
  - yes
    - not sick
    - sick

- But searching for the best rule can get expensive.
Bonuseslide: Which Score Function?

• Shouldn’t we just use accuracy score?
  – For leafs: yes, just maximize accuracy.
  – For internal nodes: maybe not.
    • There may be no simple rule like (egg > 0.5) that improves accuracy.

• Most common score in practice: information gain.
  – Choose split that decreases entropy (“randomness”) of labels the most.
  – Basic idea: easier to find good rules on “less random” labels.
• Entropy is measure of “randomness” of a set of variables.

Low entropy means “very predictable”  High entropy means “very random”

• For discrete data, the uniform distribution has the highest entropy.
• So **information gain** tries to make labels “more predictable”.
Bonus Slide: Probabilistic Predictions

• Often, we’ll have multiple ‘y’ values at each leaf node.
• In these cases, we might return probabilities instead of a label.

• E.g., if in the leaf node we have 5 “sick” objects and 1 “not sick”:
  – Return $p(y = \text{“sick”} \mid x_i) = 5/6$ and $p(y = \text{“not sick”} \mid x_i) = 1/6$.

• In general, a natural estimate of the probabilities at the leaf nodes:
  – Let ‘$n_k$’ be the number of objects that arrive to leaf node ‘k’.
  – Let ‘$n_{kc}$’ be the number of times (y == c) in the objects at leaf node ‘k’.
  – Maximum likelihood estimate for this leaf is $p(y = c \mid x_i) = n_{kc}/n_k$. 