Classification and decision trees

- Reading: Russell and Norvig, ch. 18.1-18.3
- Learning
- Classification
- Decision trees

Learning agents

- Cannot anticipate all situations
  - Known missing cases – designer of agent aware of something that agent will need to learn, like other vehicles for self-driving car
  - Unknown future – ex: something that depends on the news
  - Problem without a known solution – ex: hiring good employees

Forms of learning

- To consider:
  - Which component of the agent should be improved
  - What knowledge the agent already has
  - How data and component are represented
  - What feedback is available

Supervised learning

- Data is available with correct answer given
- Agent is trying to learn a function
- *Unsupervised learning* tries to find patterns without guidance as to correct answer
- *Reinforcement learning* learns from rewards and punishments
  - Ex: move that results in losing a game should be remembered as a mistake

Modeling function to learn

- Model tries to fit examples
- Frequently make assumptions about form of correct model
  - Ex: linear regression assumes solution is a linear equation
- General issue: avoid *overfitting* the training data
  - Model matches training data, but does not work well for other inputs
- General principle: among options, choose simplest model
  - Ex: if you don't get a birthday card from a friend, what are some possible interpretations? Which is the most likely?

Classification

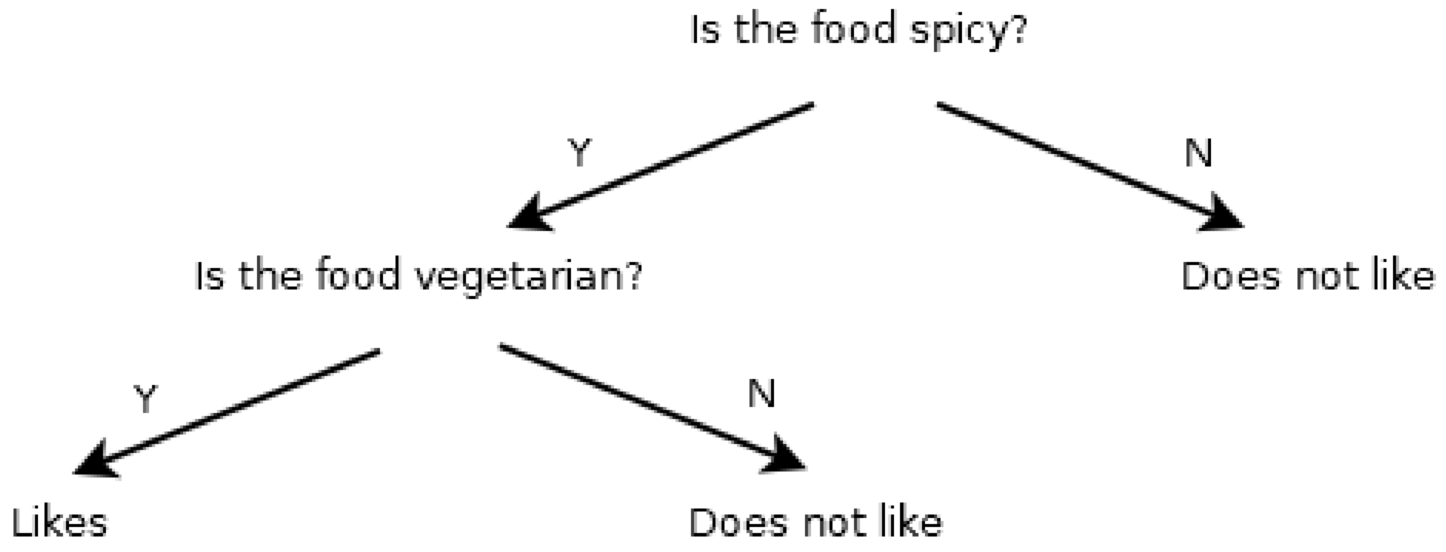
- Given: finite set of classes, items to label as belonging to exactly one of these classes
  - [Regression is a similar problem where you assume there is a relationship from input to output that you can model
    - Linear regression assumes the output is a linear combination of the input variables (ex: $2x + 3y - 1$)]
- Approach:
  - Create model from *training data*, for which correct class is known
  - Validate model against test data, for which correct class is known
  - If successful, use model for unclassified data (correct class is unknown)

Decision trees

- Represent function as a tree
- Each node corresponds to testing one input
  - Children correspond to one value (sometimes a subset of the values)
- Each leaf is labeled with the class to assign to the item
- Organization is like an automated answering system or help desk

Example

- Simple decision tree to classify whether someone will like a restaurant



Notes on decision trees

- Can represent any function of the inputs
- Naïve approach: create tree to match training data exactly
    - May not be able to do this if training data includes cases where instances producing same inputs belong to different classes
    - What else would be wrong with this?

Expressiveness

- Limiting trees to binary decision trees with *n* boolean variables, 2^(2^n) possible trees
  - Vs. 3^n options for conjunctive normal form in logic
  - More expressive than logic, but also more models will fit training data

Implementing decision trees

- Key choice is how to pick attribute to use at each node
  - Training data is split over each child node based on value of chosen attribute
  - Recursively choose attribute for each child node
- If only one class represented node, it’s a leaf where that class is picked
- If more than one class represented at leaf node, pick most common class

Choosing attributes at each node

- Ideal: if attribute A is used, all training data for this node is divided so that each child node only contains data from one class
- One way to interpret this is that finding such an attribute maximizes the *information gain*
    - This is also considered (equivalently) a reduction in the *entropy*
    - If ideal distribution reached, entropy = 0
        - Prefer attributes that get closer to the ideal
        - Worst case: resulting nodes are split evenly among the classes

Choosing attributes at each node

- The specific formula for entropy/information = $\Sigma -P_i \log P_i$ over all *i*, where P_i = probability of *i*th value of attribute's domain
  - Approximate probability by frequency in training data
  - Units = bits: idea is that if 2 values A and B are equally likely, it takes 1 bit to split data into one group with only A's and one group with only B's
  - When considering an attribute, compare entropy after split
    - Add up entropies of all children, weighted by how many data points are distributed to each child node

Example

- See Russell's example at: https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec08.pdf
  - (explains attributes better than in Russell's slides)
  - Classes: to wait or not wait at a restaurant
  - At root, consider all attributes:
    - Using type of food, all child nodes have equal wait/don't wait cases, so entropy is still 1 = zero information gain
    - For whether there are Patrons in the restaurant (none/some/full), entropy (H)

    = -2/12*H([0,1]) – 4/12 * H([1,0]) – 6/12 * H([2/6, 4/6])

    ≈ .459

Issues, improvements

- Attributes with a large domain appear to have a large information gain even when not useful for classification
  - Ex: credit card #, social security #
  - When appropriate, can *bin* the data (in restaurant example, wait time is binned into: 0-10 min., 10-30, 30-60, >60)
    - How this is decided is not well defined

Issues, improvements

- *Pruning* decision tree can simplify model, improve accuracy
  - Ex: can test impact of removing node on accuracy
    - Rudin (MIT notes) mentions C4.5 algorithm (Quinlan):
      - Replace node by most common class (turn node into a leaf)
      - Replace node by one of the subtrees
- Best division at a node may not be using a single attribute
  - Support vector machines support more complicated division