331 – Intro to Intelligent Systems
Week 08
Machine Learning
Decision Trees

T.J. Borrelli

Machine Learning

“Natural Selection is the blind watchmaker, blind because it does not see ahead, does not plan consequences, has no purpose in view. Yet the living results of natural selection overwhelmingly impress us with the appearance of design as if by a master watchmaker, impress us with the illusion of design and planning.”

- Richard Dawkins, “The Blind Watchmaker”

Machine Learning

- The goal of machine learning is to develop algorithms that will automate the process of decision-making based on data
  - This will allow the machine to change its *behavior* based on the data it receives by recognizing complex patterns in the data and extrapolating to new situations

Machine Learning

- An agent is **learning** if it improves its performance on future tasks after making observations about the world
- From a collection of input/output pairs, learn a function that predicts output for new inputs
- Why do this?
  - Designers cannot anticipate all possible situations
  - Designers cannot anticipate all changes over time
  - Designers may not know how to solve the problem themselves

Machine Learning

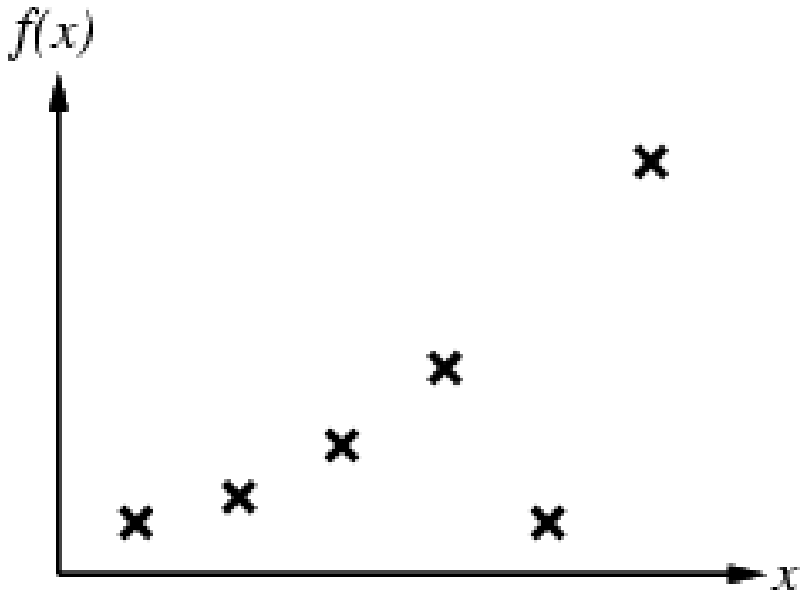
- Machine learning includes:
  - Neural networks
  - Genetic algorithms
  - Bayesian networks
  - Fuzzy Logic
  - Data mining
  - Support vector machines
  - etc.

Learning in General

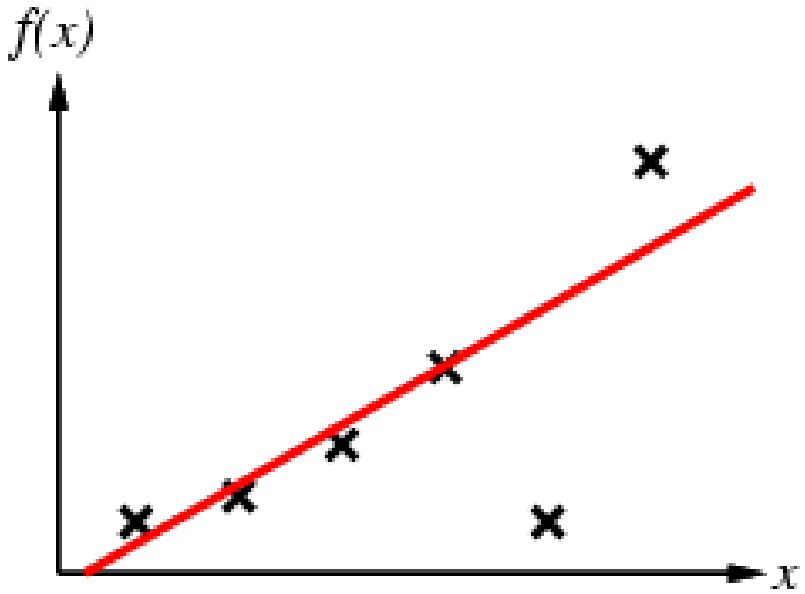
- The goal of learning is to be able to improve performance on future tasks after making observations about the world
- *Deductive learning* attempts to deduce new knowledge from rules and facts using logical inference – going from a known general rule to a new rule that is logically entailed
- *Inductive learning* attempts to learn new knowledge from examples – learning a general function or rule from specific input/output pairs

Inductive Learning Method

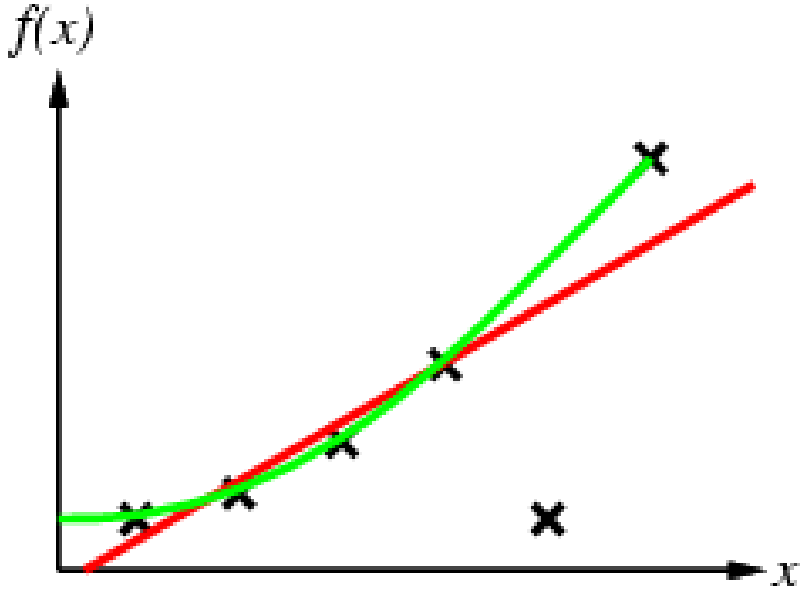
- Construct or adjust an hypothesis *h* to agree with a function *f* given a training set
- *h* is consistent if it agrees with *f* on all examples
- Consider curve fitting:



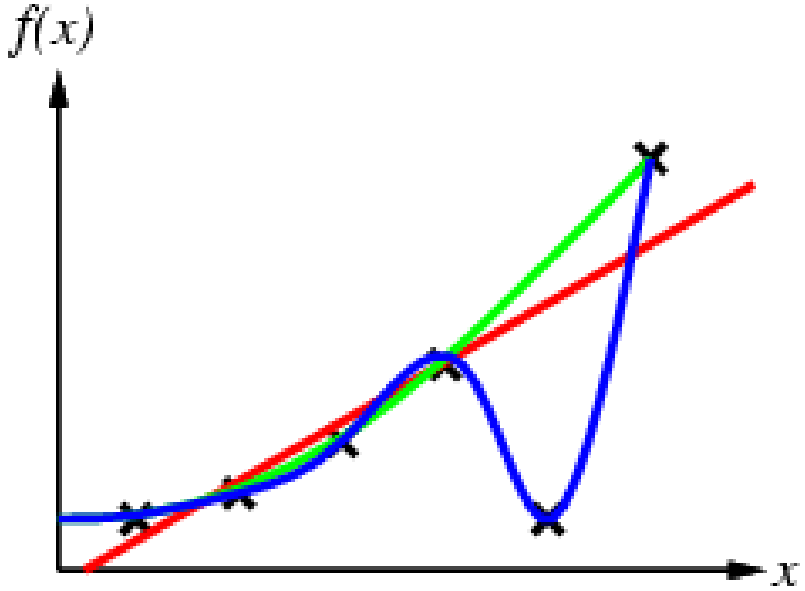
Inductive Learning Method



Inductive Learning Method



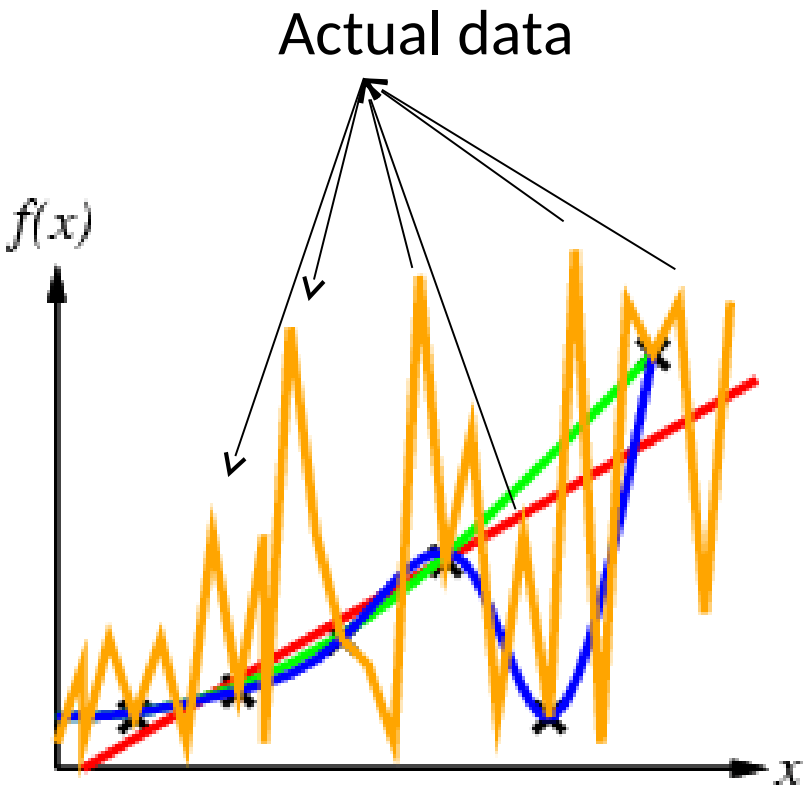
Inductive Learning Method



Inductive Learning Method

Occam's (or Ockham's) razor: prefer the simplest hypothesis that is consistent with *all* of the data. (consistent if it agrees with all data).

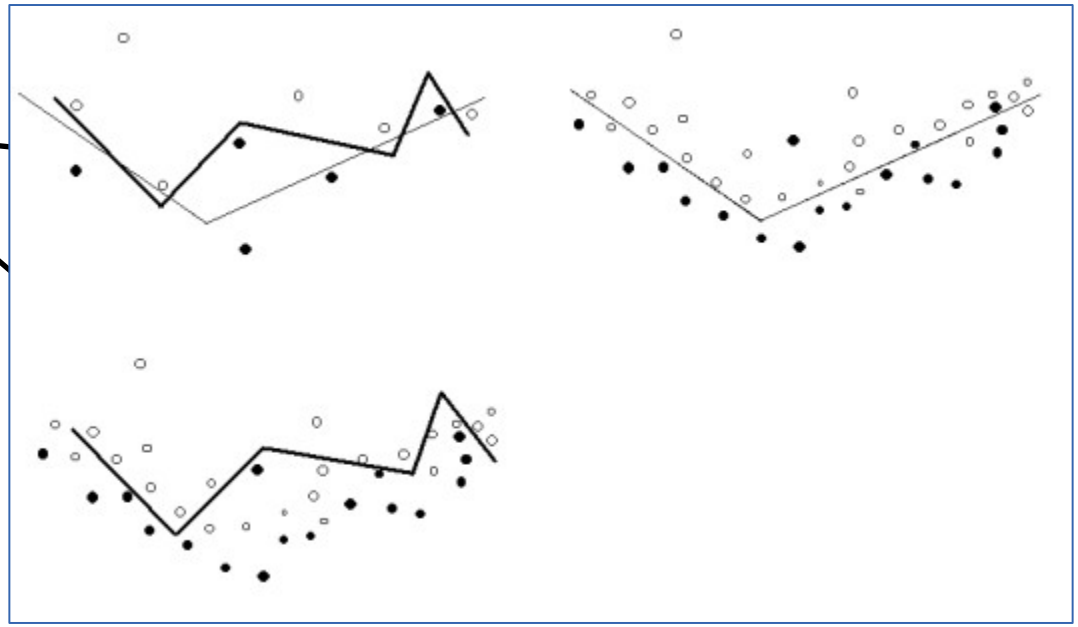
But beware of under-fitting.



The Problem of Over-fitting

The black dots represent positive examples, the gray dots represent negative examples. Goal: find the line that clearly separates the two sets of data. The two lines represent two different hypotheses.

Complex hypothesis

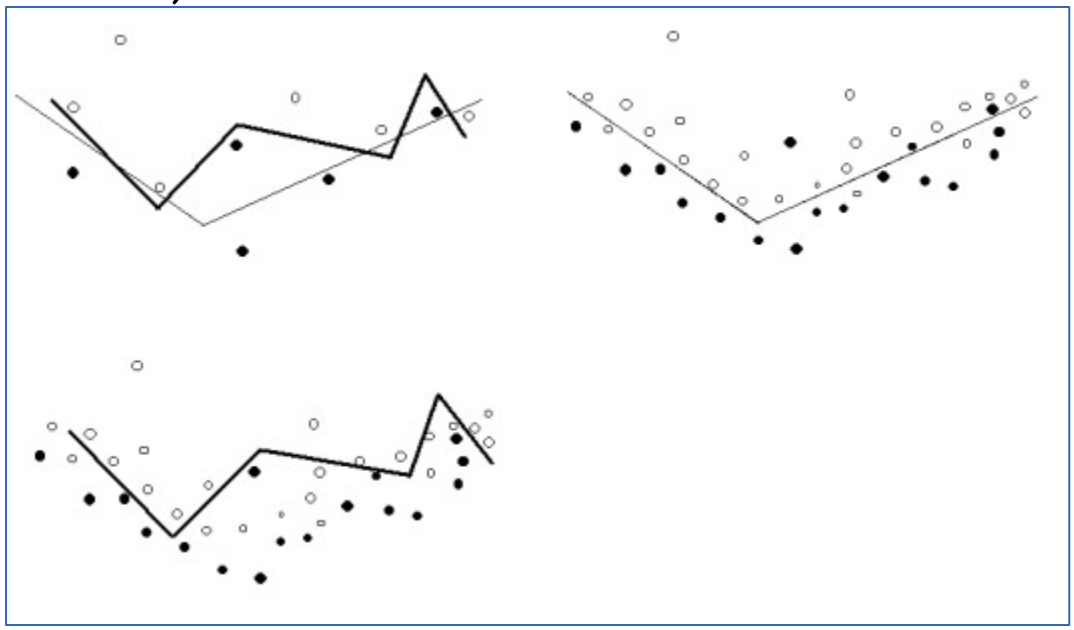


In the upper left diagram, there are just a few items of training data. These are correctly classified by the complex hypothesis. In the upper right diagram we see the complete set of data. The complex hypothesis was obviously incorrect, as shown at the lower left.

In general there is a tradeoff Between a complex hypothesis that fits the training data well and a simpler hypothesis that may generalize better.

The Problem of Over-fitting

- The simpler hypothesis, which matches the training data less well, matches the rest of the data better than the more complex hypothesis, which over-fits



Types of Inductive Learning

- Inductive learning can be *supervised* (a "teacher" is involved) or *unsupervised* (the "student" is on his or her own)
- Supervised learning is called *classification*
- Unsupervised learning is called *clustering*
- Reinforcement learning is another type of learning where there are "rewards" or "punishments" for answers that are correct or incorrect

Supervised Learning

- Also known as classification
- A classifier is designed by using a *training set* of patterns of known class for the purpose of determining the class of future sample patterns of unknown class
- A *test set* of patterns is also provided for which the true class is known, for the purpose of evaluating the effectiveness of the classifier

Non-Parametric Classification

- We do not have enough knowledge or data to be able to assume the general form of the probability model, or to estimate the relevant parameters
  - Look directly at the data instead of a summary of the data (decision trees, etc.)
  - Look at histograms, scatter plots, tables of the data (nearest-neighbor, etc.)

Nearest Neighbor Classification

- Each sample represents a “point” in feature space
- Classify unknown sample as belonging to the same class as the most similar or “nearest” sample point in the training set
- By “nearest” we usually mean the smallest distance in an n-dimensional feature space

Nearest Neighbor Classification

- Euclidean distance formula:

$$d(a,b) = \text{sqrt}\left(\sum (b_i - a_i)^2 \right)$$

- Square root is optional, to save computation time (relative distances remain the same)
- Euclidean distance emphasizes large distances
  - May want to use absolute differences in each feature instead of squared differences

Nearest Neighbor Classification

- Absolute distance formula:

$$d(a,b) = \sum | b_i - a_i |$$

  - Also called “city block distance” or Manhattan distance
- Maximum distance metric

$$d(a,b) = \max | b_i - a_i |$$

  - Finds only the most dissimilar pair of features

Nearest Neighbor Classification

- Minkowski distance formula:

$$d(a,b) = [\sum (b_i - a_i)^r]^{1/r}$$

  - "r" is an adjustable parameter
  - Generalization of the three previously defined distance metrics

Nearest Neighbor Classification

- Problem of scale:
  - An arbitrary change in the unit of measurement of one of the features could easily affect the decision
  - For example, measuring a length in millimeters rather than in meters would increase the relative contribution of this feature by a factor of 1000 compared to the other features if city block distance is used, and by 1,000,000 if Euclidean distance is used!

Nearest Neighbor Classification

- Problem of scale:
  - If one feature has a very wide range of possible values compared to the other features, it will have a very large effect on the total dissimilarity, and the decisions will be based primarily upon this single feature
  - To overcome this, it is necessary to apply scale factors to the features before computing the distances

Nearest Neighbor Classification

- Problem of scale:
  - If we want the potential influence of each of the features to be about equal, the features should be scaled so that each of them has the same standard deviation, range, or other measure of spread for the entire data set

Nearest Neighbor Classification

- Problem of scale:
  - Normalize each feature, x_i, to have a mean of 0 and a standard deviation of 1 for the entire data set:
  - i.e., replace each feature x_i with:

$$z_i = (x_i - \mu_i) / \sigma_i$$

Nearest Neighbor Classification

- Problem of scale:
  - If some prior knowledge of the relative importance of the features is available, the features could be scaled according to their importance or desired contribution to the decision making
  - Let the range, standard deviation, or weight of each normalized feature be proportional to the "accuracy" of that feature (where "accuracy" is defined as the probability of a correct decision when that feature is used alone)

K-Nearest Neighbor (k-NN)

- Base the classification of a sample on the number of “votes” of *k* of its nearest neighbors, rather than on only its single nearest neighbor - denoted *k*-NN
- Choosing *k* too large tends to suppress the fine structure of the underlying density, while choosing *k* too small puts too much emphasis on the chance locations of a few sample points
- In practice, various values of *k* are tried on the training set, and the one that gives the best result is chosen

K-Nearest Neighbor (k-NN)

- k-NN suffers from “the curse of dimensionality”
- In low-dimensional spaces with plenty of data nearest neighbor works well
- As the number of dimensions rises the nearest neighbors are usually not very near
  - Most neighbors are “outliers”!
- In addition, search can be difficult if there are a large number of samples – $O(N^2)$
- We would like sub-linear time complexity

Application: Pandora

- Pandora is an Internet music radio service that allows users to build customized “stations” that play music similar to a song or artist that is specified
- Pandora uses a k-NN style process called *The Music Genome Project* to locate new songs or artists that are similar to the user-specified song or artist

Application: Pandora

The Pandora process in general:

Hundreds of variables are created on which a song can be measured on a scale from 0 to 5 (e.g., acid-rock, accordion playing, etc.)

Pandora pays musicians to analyze songs and rate each one on each variable

The user specifies a song he or she likes (must be in Pandora's database)

The distance between user's choice and every song in the database is calculated

User has option to say "I like this song", "I don't like this song", or nothing

If "like" is chosen, the song and closest song are merged into a cluster

Classification Errors

- Classification errors arise when an observation that is known to belong to a certain class is classified as belonging to a different class
- A very simple rule for classification is to classify the observation as belonging to the most prevalent class (i.e., ignore any predictor results)
- This is known as the *naïve rule*, and is used as a baseline or benchmark for evaluating the performance of more complicated classifiers
    - A classifier that uses predictor results should outperform this

Estimation of Error Rate

- After developing a classification procedure, we must evaluate its performance
  - Model-based error estimation (assumes distributions are known)
  - Counting-based error estimation (assumes true classification or "ground-truth" is known for a sample test set)
  - "Confusion matrix" (also known as a contingency table)

Learning Decision Trees

- Decision tree induction is one of the simplest and most successful forms of machine learning
- A decision tree represents a function that takes as input a vector of attribute values and returns a decision (single output value)
- A decision tree reaches its decision by performing a sequence of tests on each (non-terminal) node in the tree
- Restaurant example

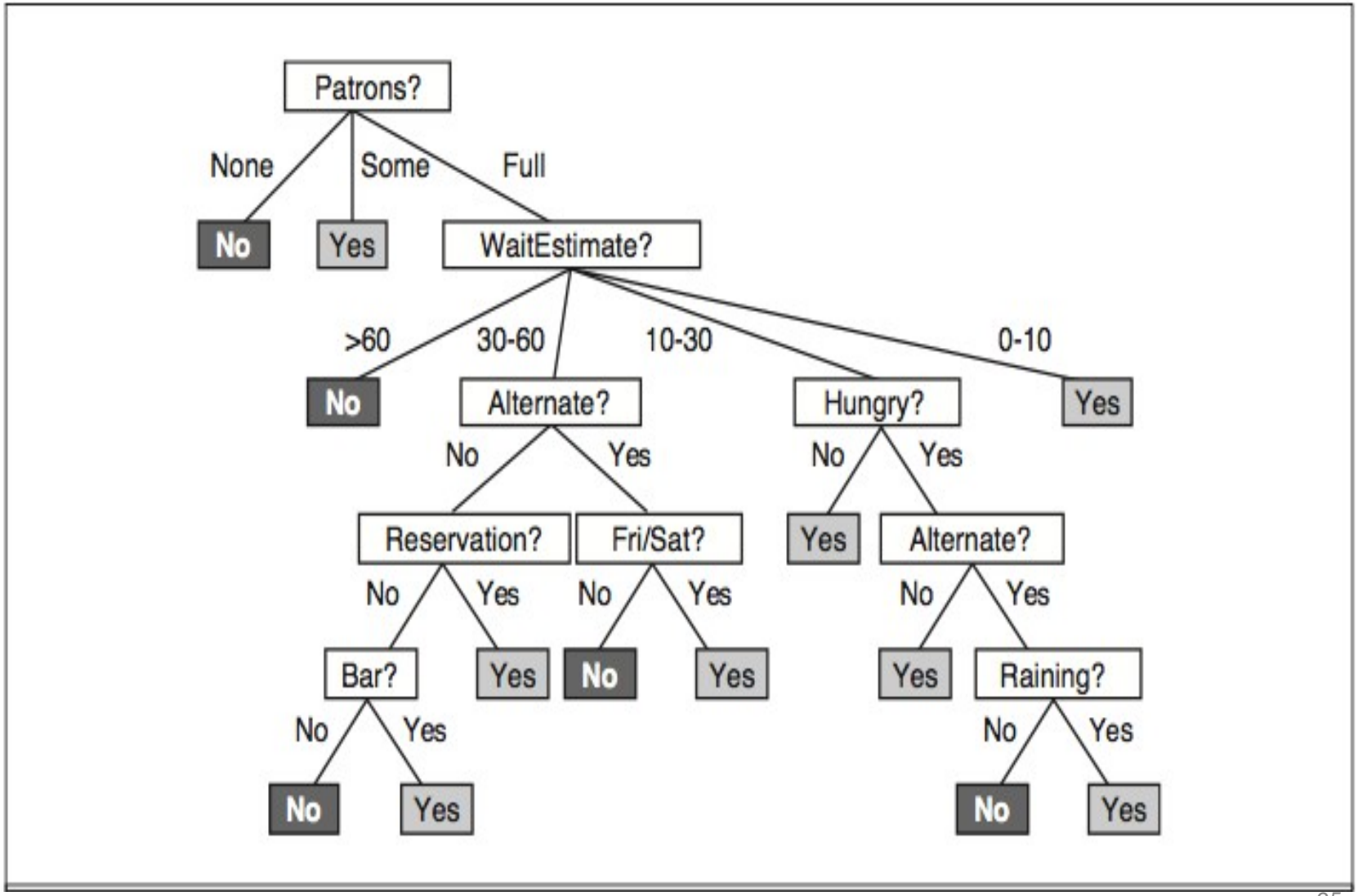
Restaurant Example

- We wish to build a decision tree to decide whether to wait for a table at a restaurant
- Goal predicate: *WillWait*
- List of attributes to consider
  - Alternate, Bar, Fri/Sat, Hungry, Patrons, Price, Raining, Reservation, Type, WaitEstimate

Restaurant Example

- List of attributes to consider
  - Alternate: is there another suitable restaurant nearby
  - Bar: does the restaurant have a comfy bar area
  - Fri/Sat: is it friday or saturday
  - Hungry: are we *really* hungry
  - Patrons: how many people are here (None, Some, Full)
  - Price: 3 categories ($, $$, $$$)
  - Raining: whether it is raining outside
  - Reservation: whether we made a reservation
  - Type: kind of restaurant (French, Italian, Thai, burger)
  - WaitEstimate: wait time estimated by host (<10min, 10-30min, 31-60min, >60min)

Decision Tree – deciding whether to wait for a table



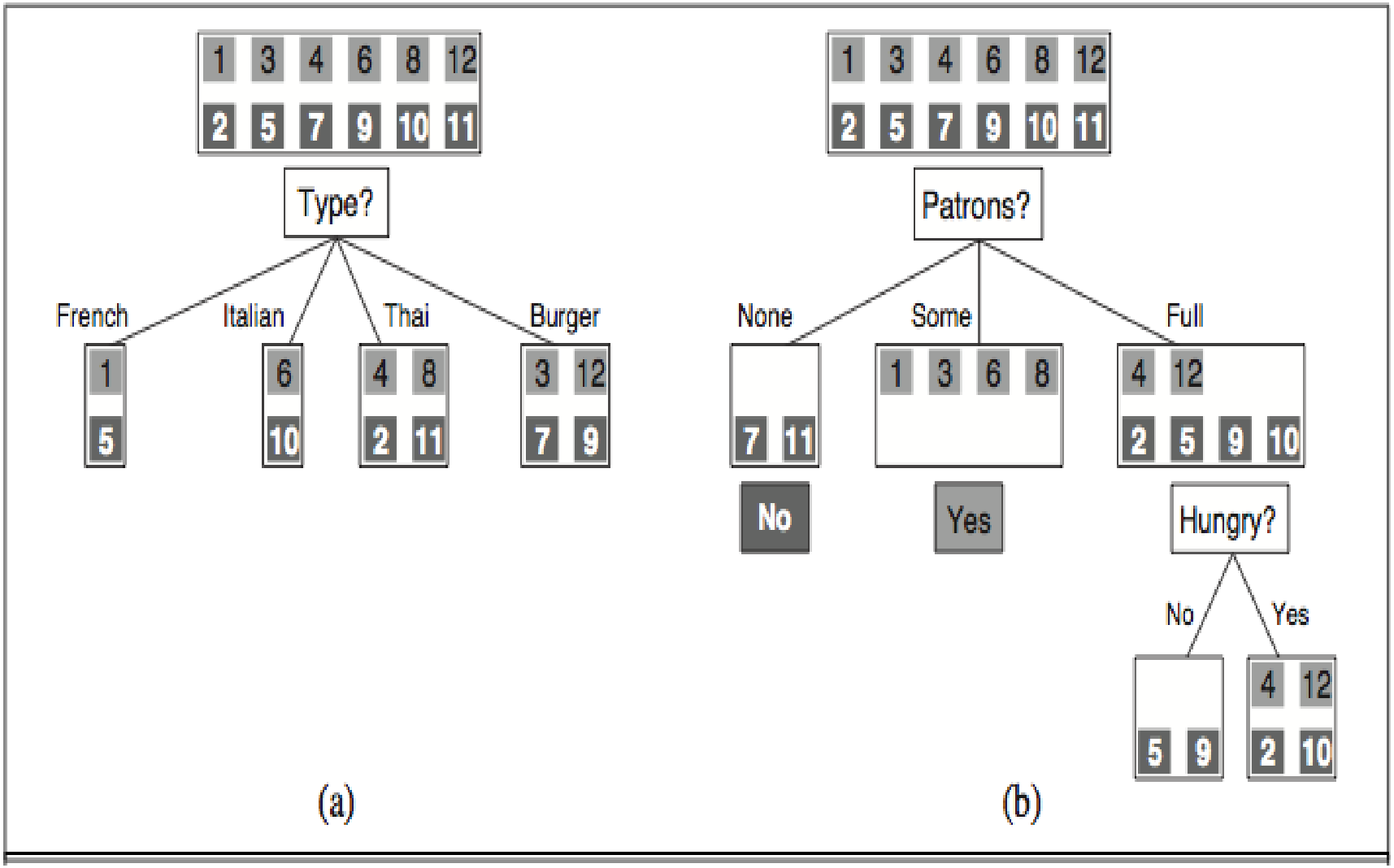
Decision Trees

- For a wide variety of problems, the decision tree format yields a nice, concise result
- Some problems cannot be represented concisely (explodes into exponentially many nodes)
- Our goal with the decision tree is to create one that is consistent with the input/output pairs and is as small as possible
- Often finding the *smallest* tree is also in intractable so we may need to use heuristics

Non-exhaustive list of inputs and outputs

Num	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Wait
x_1	yes	no	no	yes	some	$$$	no	yes	French	0-10	yes
x_2	yes	no	no	yes	full	$	no	no	Thai	30-60	no
x_3	no	yes	no	no	some	$	no	no	Burger	0-10	yes
x_4	yes	no	yes	yes	full	$	yes	no	Thai	10-30	yes
x_5	yes	no	yes	no	full	$$$	no	yes	French	>60	no
x_6	no	yes	no	yes	some	$$	yes	yes	Italian	0-10	yes
x_7	no	yes	no	no	none	$	yes	no	Burger	0-10	no
x_8	no	no	no	yes	some	$$	yes	yes	Thai	0-10	yes
x_9	no	yes	yes	no	full	$	yes	no	Burger	>60	no
x_{10}	yes	yes	yes	yes	full	$$$	no	yes	Italian	10-30	no
x_{11}	no	no	no	no	none	$	no	no	Thai	0-10	no
x_{12}	yes	yes	yes	yes	full	$	no	no	Burger	30-60	yes

Possible Decision Trees



Decision Trees

- Splitting on *Type* (on last slide) brings us no closer to being able to distinguish between “Yes” and “No” answers
- Splitting on *Patrons* however does a good job of separating “Yes” from “No” selections
- We still have some work to do, but we found some leaf nodes early on

Decision Trees

- When creating our decision tree the goal is to approximately minimize its depth
- Therefore we should pick attributes in such a way that we can classify inputs as soon as possible
- A “perfect” attribute is one that divides the examples into sets, each of which are all positive or all negative (thus terminal)
- A really useless attribute such as Type on the previous slide, leaves the example sets with roughly the same proportion of positive and negatives examples as in the original set

Decision Trees and Entropy

- How do we know which attributes to pick so that we pick good ones?
- (Shannon) Entropy – from information theory – a measure of the uncertainty of a random variable
- As information goes up (i.e. we make new observations) – entropy goes down
- Consider a (fair) coin flip – equally likely to come up heads or tails (0 or 1) – this corresponds to "1 bit" of entropy
- Roll of a 4-sided die has 2 bits of entropy – takes 2 bits to describe each of the 4 outcomes