Lec09: Decision Trees and Ensembles

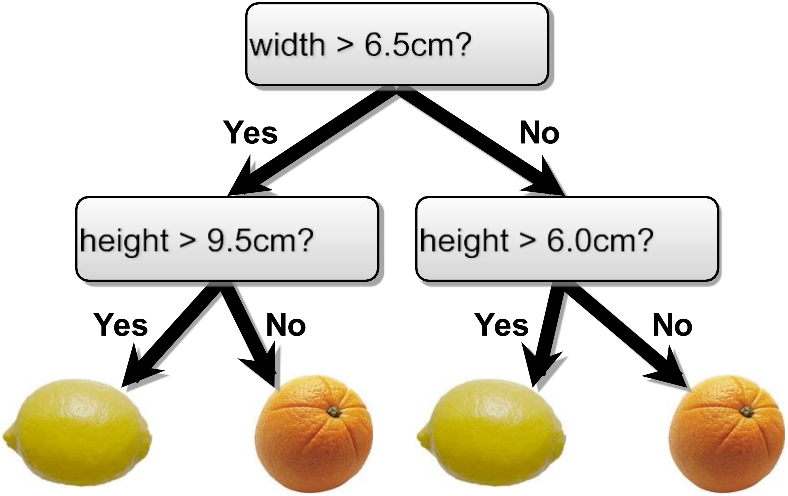
Xiaojun Chang

University of Science and Technology of China

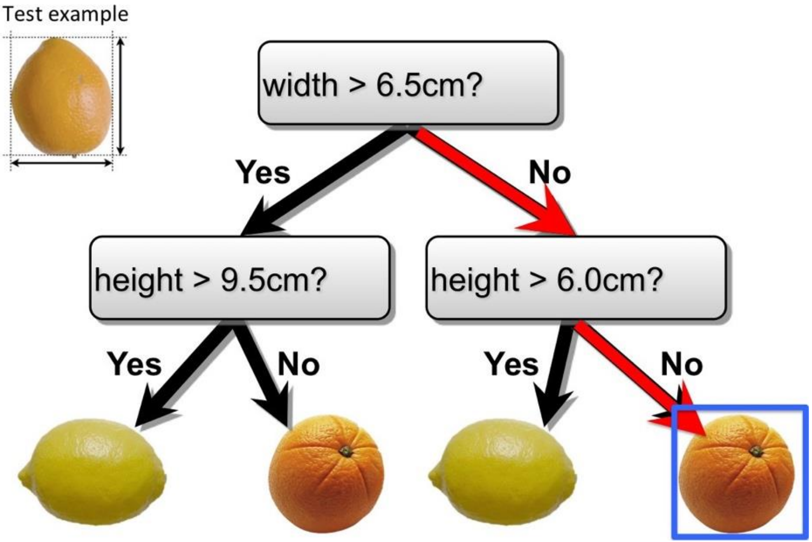
Overview

- Decision Trees
  - Simple but powerful learning algorithm
  - One of the most widely used learning algorithms in Kaggle competitions
- Lets us introduce ensembles, a key idea in ML more broadly
- Useful information theoretic concepts (entropy, mutual information, etc.)

Decision Trees

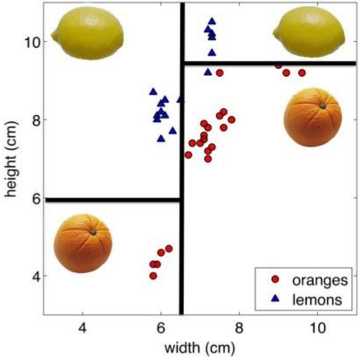


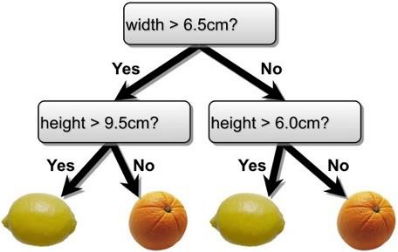
Decision Trees



Decision Trees

- Decision trees make predictions by recursively splitting on different attributes according to a tree structure.





Example with Discrete Inputs

- What if the attributes are discrete?

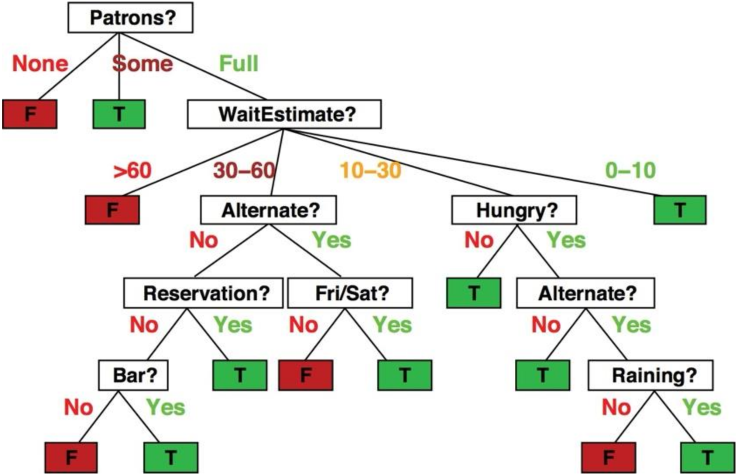
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	Yes	No	No	Yes	Some	$$$	No	Yes	French	0–10	$y_1 =$ Yes
$\mathbf{x}_2$	Yes	No	No	Yes	Full	$	No	No	Thai	30–60	$y_2 =$ No
$\mathbf{x}_3$	No	Yes	No	No	Some	$	No	No	Burger	0–10	$y_3 =$ Yes
$\mathbf{x}_4$	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10–30	$y_4 =$ Yes
$\mathbf{x}_5$	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	$y_5 =$ No
$\mathbf{x}_6$	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0–10	$y_6 =$ Yes
$\mathbf{x}_7$	No	Yes	No	No	None	$	Yes	No	Burger	0–10	$y_7 =$ No
$\mathbf{x}_8$	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0–10	$y_8 =$ Yes
$\mathbf{x}_9$	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	$y_9 =$ No
$\mathbf{x}_{10}$	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10–30	$y_{10} =$ No
$\mathbf{x}_{11}$	No	No	No	No	None	$	No	No	Thai	0–10	$y_{11} =$ No
$\mathbf{x}_{12}$	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30–60	$y_{12} =$ Yes

Attributes:

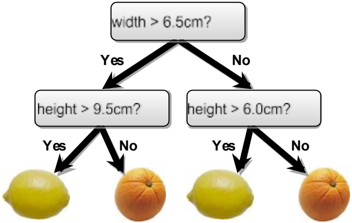
1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
6.	Price: the restaurant's price range ($, $$, $$$).
7.	Raining: whether it is raining outside.
8.	Reservation: whether we made a reservation.
9.	Type: the kind of restaurant (French, Italian, Thai or Burger).
10.	WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

Decision Tree: Example with Discrete Inputs

- The tree to decide whether to wait (T) or not (F)

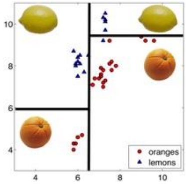


Decision Trees



- Internal nodes test attributes
- Branching is determined by attribute value
- Leaf nodes are outputs (predictions)

Decision Tree: Classification and Regression



- Each path from root to a leaf defines a region R_m of input space
- Let $\{(x^{(m_1)}, t^{(m_1)}), \ldots, (x^{(m_k)}, t^{(m_k)})\}$ be the training examples that fall into R_m
- **Classification tree**:
  - discrete output
  - leaf value y^m typically set to the most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$
- **Regression tree**:
  - continuous output
  - leaf value y^m typically set to the mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

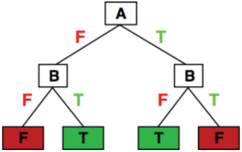
Note: We will focus on classification

[Slide credit: S. Russell]

Expressiveness

- **Discrete-input, discrete-output case**:
  - Decision trees can express any function of the input attributes
  - E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- **Continuous-input, continuous-output case**:
  - Can approximate any function arbitrarily closely
- Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples

[Slide credit: S. Russell]

How do we Learn a Decision Tree?

- How do we construct a useful decision tree?

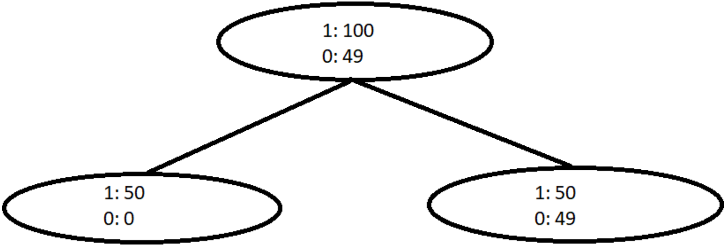
Learning Decision Trees

Learning the simplest (smallest) decision tree is an NP complete problem [if you are interested, check: Hyafil & Rivest'76]

- Resort to a greedy heuristic:
  - Start from an empty decision tree
  - Split on the "best" attribute
  - Recurse
- Which attribute is the "best"?
  - Choose based on accuracy?

Choosing a Good Split

- Why isn't accuracy a good measure?



- Is this split good? Zero accuracy gain.
- Instead, we will use techniques from information theory

Idea: Use counts at leaves to define probability distributions, so we can measure uncertainty

Choosing a Good Split

- Which attribute is better to split on, X_1 or X_2?
  - Deterministic: good (all are true or false; just one class in the leaf)
  - Uniform distribution: bad (all classes in leaf equally probable)
  - What about distributons in between?

Note: Let's take a slight detour and remember concepts from information theory

[Slide credit: D. Sontag]

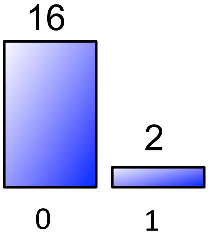
We Flip Two Different Coins

Sequence 1:

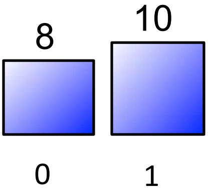
0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 ... ?

Sequence 2:

0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?



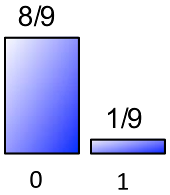
versus



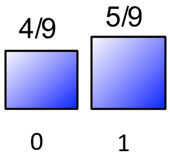
Quantifying Uncertainty

Entropy is a measure of expected “surprise”:

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



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

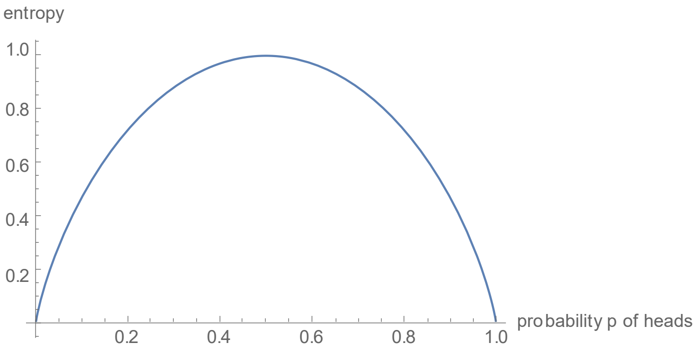


$$-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

- Measures the information content of each observation
- Unit = bits
- A fair coin flip has 1 bit of entropy

Quantifying Uncertainty

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



Entropy

- **“High Entropy”**:
  - Variable has a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable
- **“Low Entropy”**
  - Distribution of variable has many peaks and valleys
  - Histogram has many lows and highs
  - Values sampled from it are more predictable

[Slide credit: Vibhav Gogate]

Entropy of a Joint Distribution

- Example: $X = \{\text{Raining, Not raining}\}$, $Y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

$$
\begin{aligned}
H(X,Y) &= -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 p(x,y) \\
&= -\frac{24}{100} \log_2 \frac{24}{100} - \frac{1}{100} \log_2 \frac{1}{100} - \frac{25}{100} \log_2 \frac{25}{100} - \frac{50}{100} \log_2 \frac{50}{100} \\
&\approx 1.56\text{bits}
\end{aligned}
$$

Specific Conditional Entropy

- Example: X = {Raining, Not raining}, Y = {Cloudy, Not cloudy}

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

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

$$\begin{aligned} H(Y|X=x) &= -\sum_{y \in Y} p(y|x) \log_2 p(y|x) \\ &= -\frac{24}{25} \log_2 \frac{24}{25} - \frac{1}{25} \log_2 \frac{1}{25} \\ &\approx 0.24\text{bits} \end{aligned}$$

- We used: $p(y|x) = \frac{p(x,y)}{p(x)}$, and $p(x) = \sum_y p(x,y)$ (sum in a row)

Conditional Entropy

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

- The expected conditional entropy:

$$\begin{aligned} H(Y|X) &= \sum_{x \in X} p(x) H(Y|X = x) \\ &= -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x) \end{aligned}$$

Conditional Entropy

- Example: X = {Raining, Not raining}, Y = {Cloudy, Not cloudy}

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

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

$$\begin{aligned} H(Y|X) &= \sum_{x \in X} p(x) H(Y|X=x) \\ &= \frac{1}{4} H(\text{cloudy}|\text{is raining}) + \frac{3}{4} H(\text{cloudy}|\text{not raining}) \\ &\approx 0.75 \text{ bits} \end{aligned}$$

Conditional Entropy

- Some useful properties:
  - H is always non-negative
  - Chain rule: $H(X, Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
  - If X and Y independent, then X doesn't tell us anything about Y: $H(Y|X) = H(Y)$
  - But Y tells us everything about Y: $H(Y|Y) = 0$
  - By knowing X, we can only decrease uncertainty about Y: $H(Y|X) \leq H(Y)$

Information Gain

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

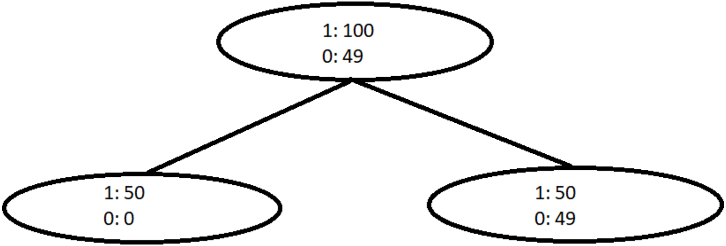
- How much information about cloudiness do we get by discovering whether it is raining?

$$
\begin{aligned}
IG(Y|X) &= H(Y) - H(Y|X) \\
&\approx 0.25 \text{ bits}
\end{aligned}
$$

- This is called the information gain in Y due to X, or the mutual information of Y and X
- If X is completely uninformative about Y: $IG(Y|X) = 0$
- If X is completely informative about Y: $IG(Y|X) = H(Y)$

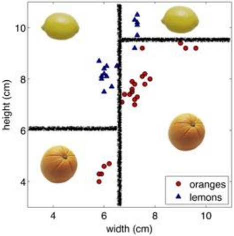
Revisiting Our Original Example

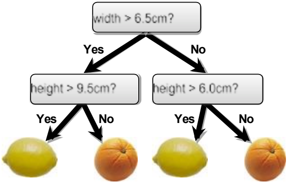
- Information gain measures the informativeness of a variable, which is exactly what we desire in a decision tree attribute!
- What is the information gain of this split?



- Root entropy: $H(Y) = -\frac{49}{149}\log_2(\frac{49}{149}) - \frac{100}{149}\log_2(\frac{100}{149}) \approx 0.91$
- Leafs entropy: $H(Y\,|\,left) = 0$, $H(Y\,|\,right) \approx 1$.
- $IG(split) \approx 0.91 - (\frac{1}{3}\cdot 0 + \frac{2}{3}\cdot 1) \approx 0.24 > 0$

Constructing Decision Trees





- 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! (choose attribute that gives the highest gain)

Decision Tree Construction 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

Back to Our Example

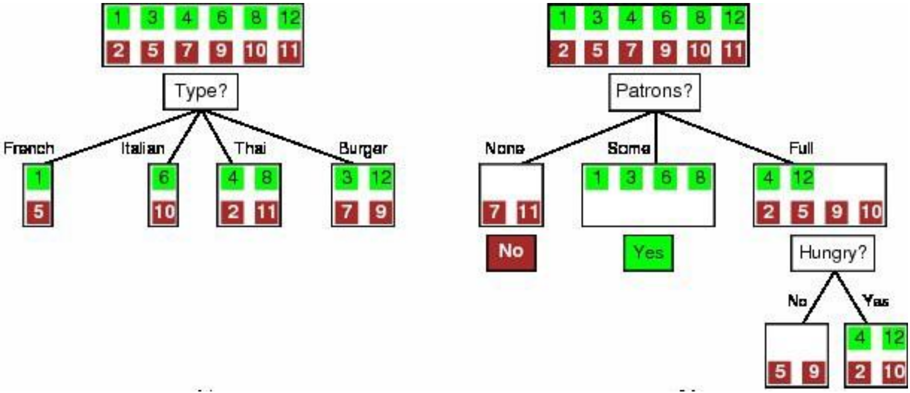
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	Yes	No	No	Yes	Some	$$$	No	Yes	French	0–10	$y_1 = Yes$
$\mathbf{x}_2$	Yes	No	No	Yes	Full	$	No	No	Thai	30–60	$y_2 = No$
$\mathbf{x}_3$	No	Yes	No	No	Some	$	No	No	Burger	0–10	$y_3 = Yes$
$\mathbf{x}_4$	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10–30	$y_4 = Yes$
$\mathbf{x}_5$	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	$y_5 = No$
$\mathbf{x}_6$	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0–10	$y_6 = Yes$
$\mathbf{x}_7$	No	Yes	No	No	None	$	Yes	No	Burger	0–10	$y_7 = No$
$\mathbf{x}_8$	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0–10	$y_8 = Yes$
$\mathbf{x}_9$	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	$y_9 = No$
$\mathbf{x}_{10}$	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10–30	$y_{10} = No$
$\mathbf{x}_{11}$	No	No	No	No	None	$	No	No	Thai	0–10	$y_{11} = No$
$\mathbf{x}_{12}$	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30–60	$y_{12} = Yes$

Attributes:

1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
6.	Price: the restaurant's price range ($, $$, $$$).
7.	Raining: whether it is raining outside.
8.	Reservation: whether we made a reservation.
9.	Type: the kind of restaurant (French, Italian, Thai or Burger).
10.	WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

[from: Russell & Norvig]

Attribute Selection

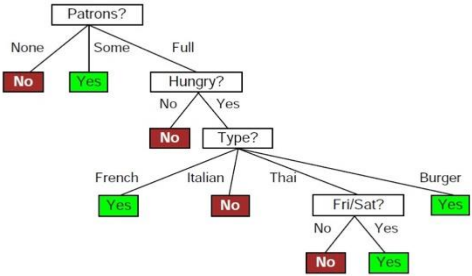


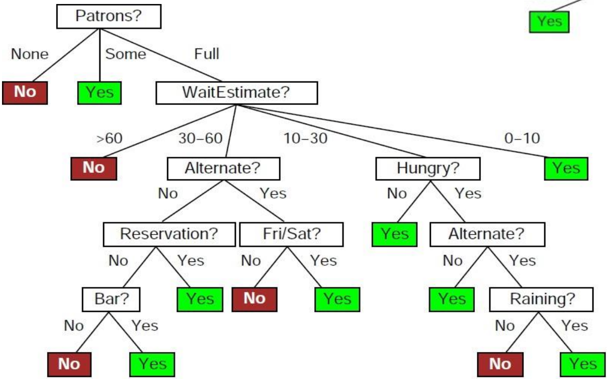
$$IG(Y) = H(Y) - H(Y|X)$$

$$IG(type) = 1 - \left[\frac{2}{12}H(Y|\text{Fr.}) + \frac{2}{12}H(Y|\text{It.}) + \frac{4}{12}H(Y|\text{Thai}) + \frac{4}{12}H(Y|\text{Bur.})\right] = 0$$

$$IG(Patrons) = 1 - \left[\frac{2}{12}H(0,1) + \frac{4}{12}H(1,0) + \frac{6}{12}H(\frac{2}{6},\frac{4}{6})\right] \approx 0.541$$

Which Tree is Better?





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
  - Human interpretability
- "Occam's Razor": find the simplest hypothesis that fits the observations
  - Useful principle, but hard to formalize (how to define simplicity?)
  - See Domingos, 1999, "The role of Occam's razor in knowledge discovery"
- We desire small trees with informative nodes near the root

Decision Tree Miscellany

- 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
- Handling continuous attributes
  - Split based on a threshold, chosen to maximize information gain
- Decision trees can also be used for regression on real-valued outputs. Choose splits to minimize squared error, rather than maximize information gain.

Comparison to k-NN

Advantages of decision trees over KNN

- Good when there are lots of attributes, but only a few are important
- Good with discrete attributes
- Easily deals with missing values (just treat as another value)
- Robust to scale of inputs
- Fast at test time
- More interpretable

Advantages of KNN over decision trees

- Few hyperparameters
- Able to handle attributes/features that interact in complex ways (e.g. pixels)
- Can incorporate interesting distance measures (e.g. shape contexts)
- Typically make better predictions in practice
  - As we'll see next lecture, ensembles of decision trees are much stronger. But they lose many of the advantages listed above.

Ensembles and Bagging

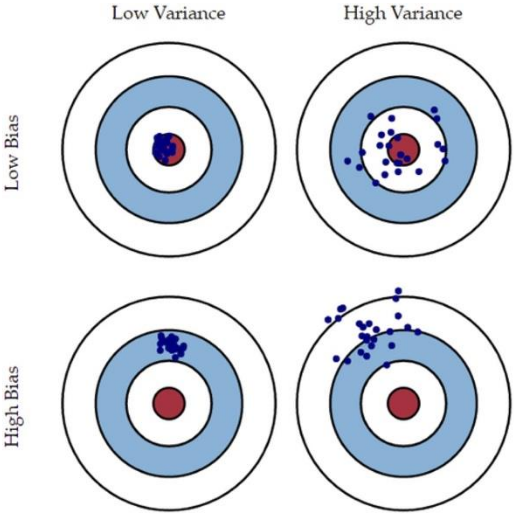
Ensemble methods: Overview

- An ensemble of predictors is a set of predictors whose individual decisions are combined in some way to classify new examples
  - E.g., (possibly weighted) majority vote
- For this to be nontrivial, the classifiers must differ somehow, e.g.
  - Different algorithm
  - Different choice of hyperparameters
  - Trained on different data
  - Trained with different weighting of the training examples
- Ensembles are usually trivial to implement. The hard part is deciding what kind of ensemble you want, based on your goals.

Ensemble methods: Overview

- This lecture: bagging
  - Train classifiers independently on random subsets of the training data.
- Later lecture: boosting
  - Train classifiers sequentially, each time focusing on training examples that the previous ones got wrong.
- Bagging and boosting serve very different purposes. To understand this, we need to take a detour to understand the bias and variance of a learning algorithm.

Bias and Variance



Loss Functions

- A loss function $L(y, t)$ defines how bad it is if the algorithm predicts y, but the target is actually t.
- Example: 0-1 loss for classification

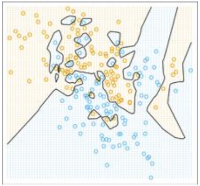
$$L_{0-1}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{if } y \neq t \end{cases}$$

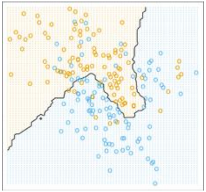
  - Averaging the 0-1 loss over the training set gives the training error rate, and averaging over the test set gives the test error rate.
- Example: squared error loss for regression

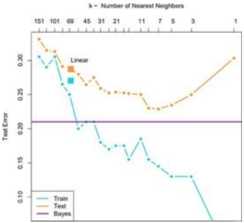
$$L_{\mathrm{SE}}(y, t) = \frac{1}{2}(y - t)^2$$

  - The average squared error loss is called mean squared error (MSE).

- Recall that overly simple models underfit the data, and overly complex models overfit.



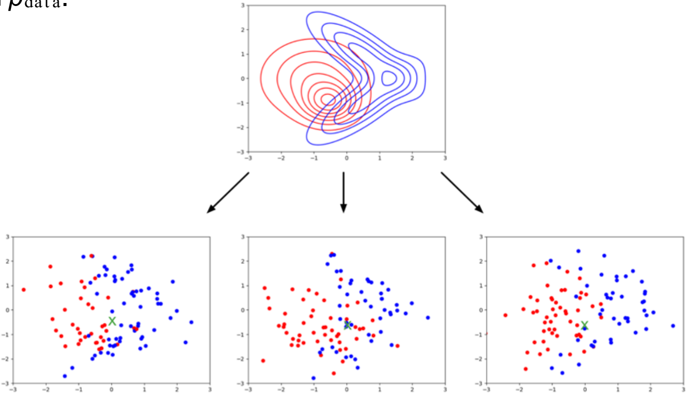




- We can quantify this effect in terms of the bias/variance decomposition.
  - Bias and variance of what?

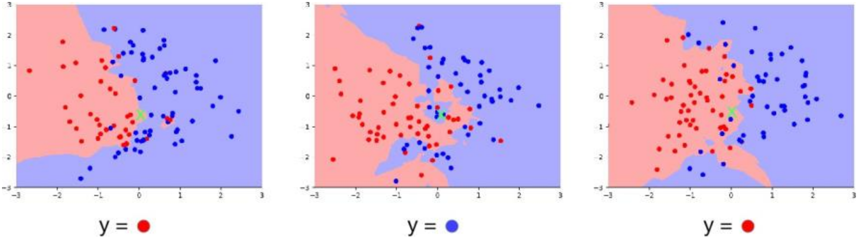
Bias-Variance Decomposition: Basic Setup

- Suppose the training set $\mathcal{D}$ consists of pairs $(\mathbf{x}_i, t_i)$ sampled independent and identically distributed (i.i.d.) from a single data generating distribution p_{data}.
- Pick a fixed query point $\mathbf{x}$ (denoted with a green x).
- Consider an experiment where we sample lots of training sets independently from p_{data}.



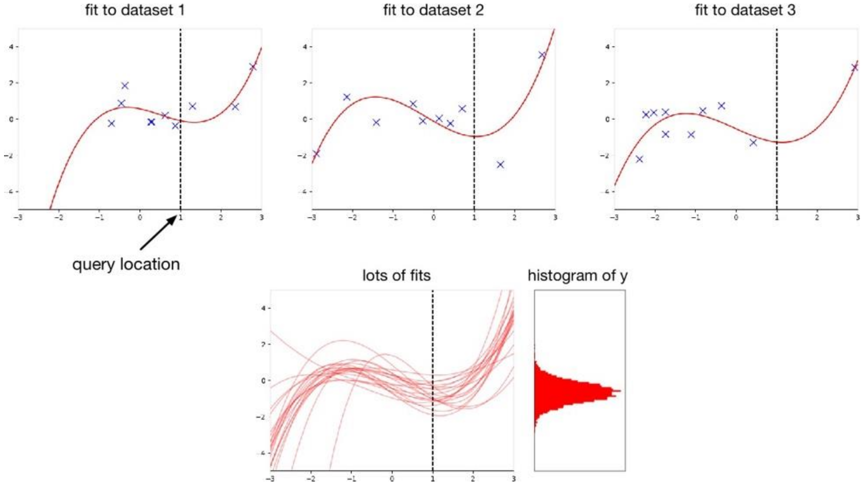
Bias-Variance Decomposition: Basic Setup

- Let's run our learning algorithm on each training set, and compute its prediction y at the query point $\mathbf{x}$.
- We can view y as a random variable, where the randomness comes from the choice of training set.
- The classification accuracy is determined by the distribution of y.



Bias-Variance Decomposition: Basic Setup

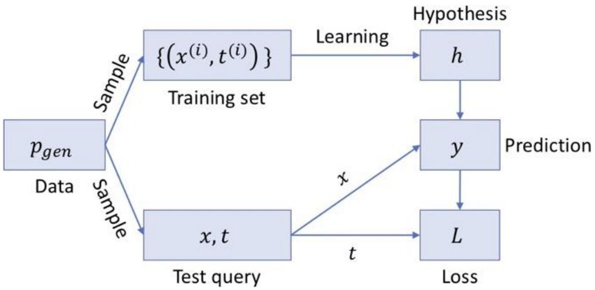
Here is the analogous setup for regression:



Since y is a random variable, we can talk about its expectation, variance, etc.

Bias-Variance Decomposition: Basic Setup

- Recap of basic setup:



- Notice: y is independent of t. (Why?)
- This gives a distribution over the loss at $\mathbf{x}$, with expectation $E[L(y, t) \mid \mathbf{x}]$.
- For each query point $\mathbf{x}$, the expected loss is different. We are interested in minimizing the expectation of this with respect to $\mathbf{x} \sim p_{\mathrm{data}}$.

Bayes Optimality

- For now, focus on squared error loss, $L(y, t) = \frac{1}{2}(y - t)^2$.
- A first step: suppose we knew the conditional distribution $p(t \mid \mathbf{x})$. What value y should we predict?
  - Here, we are treating t as a random variable and choosing y.
- **Claim:** $y_* = \mathbb{E}[t \mid \mathbf{x}]$ is the best possible prediction.
- **Proof:**

$$\begin{aligned}
\mathbb{E}[(y - t)^2 \mid \mathbf{x}] &= \mathbb{E}[y^2 - 2yt + t^2 \mid \mathbf{x}] \\
&= y^2 - 2y\mathbb{E}[t \mid \mathbf{x}] + \mathbb{E}[t^2 \mid \mathbf{x}] \\
&= y^2 - 2y\mathbb{E}[t \mid \mathbf{x}] + \mathbb{E}[t \mid \mathbf{x}]^2 + \mathrm{Var}[t \mid \mathbf{x}] \\
&= y^2 - 2yy_* + y_*^2 + \mathrm{Var}[t \mid \mathbf{x}] \\
&= (y - y_*)^2 + \mathrm{Var}[t \mid \mathbf{x}]
\end{aligned}$$

Bayes Optimality

$$\mathbb{E}[(y - t)^2 \mid \mathbf{x}] = (y - y_*)^2 + \mathrm{Var}[t \mid \mathbf{x}]$$

- The first term is nonnegative, and can be made 0 by setting $y = y_*$.
- The second term corresponds to the inherent unpredictability, or noise, of the targets, and is called the Bayes error.
  - This is the best we can ever hope to do with any learning algorithm. An algorithm that achieves it is Bayes optimal.
  - Notice that this term doesn't depend on y.
- This process of choosing a single value y_* based on $p(t \mid \mathbf{x})$ is an example of decision theory.

Bayes Optimality

- Now return to treating y as a random variable (where the randomness comes from the choice of dataset).
- We can decompose out the expected loss (suppressing the conditioning on $\mathbf{x}$ for clarity):

$$
\begin{aligned}
\mathbb{E}[(y-t)^2] &= \mathbb{E}[(y-y_*)^2] + \mathrm{Var}(t) \\
&= \mathbb{E}[y_*^2 - 2y_* y + y^2] + \mathrm{Var}(t) \\
&= y_*^2 - 2y_*\mathbb{E}[y] + \mathbb{E}[y^2] + \mathrm{Var}(t) \\
&= y_*^2 - 2y_*\mathbb{E}[y] + \mathbb{E}[y]^2 + \mathrm{Var}(y) + \mathrm{Var}(t) \\
&= \underbrace{(y_* - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}
\end{aligned}
$$

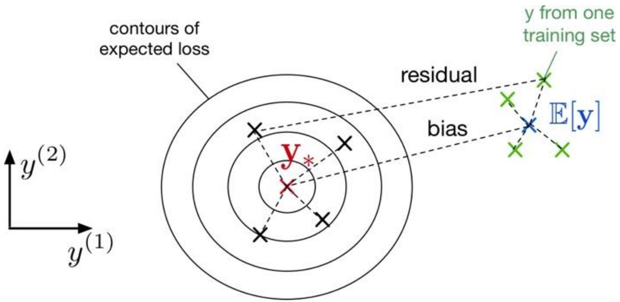
Bayes Optimality

$$\mathbb{E}[(y-t)^2] = \underbrace{(y_* - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}$$

- We just split the expected loss into three terms:
  - bias: how wrong the expected prediction is (corresponds to underfitting)
  - variance: the amount of variability in the predictions (corresponds to overfitting)
  - Bayes error: the inherent unpredictability of the targets
- Even though this analysis only applies to squared error, we often loosely use "bias" and "variance" as synonyms for "underfitting" and "overfitting".

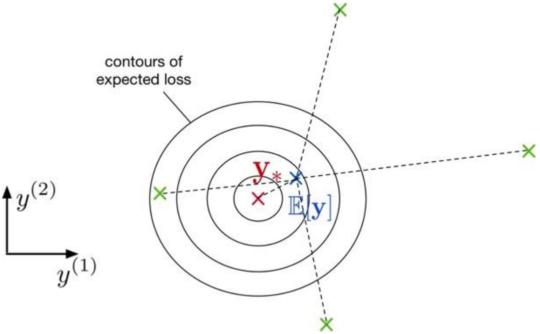
Bias/Variance Decomposition: Another Visualization

- We can visualize this decomposition in output space, where the axes correspond to predictions on the test examples.
- If we have an overly simple model (e.g. KNN with large k), it might have
  - high bias (because it's too simplistic to capture the structure in the data)
  - low variance (because there's enough data to get a stable estimate of the decision boundary)



Bias/Variance Decomposition: Another Visualization

- If you have an overly complex model (e.g. KNN with $k = 1$), it might have
  - low bias (since it learns all the relevant structure)
  - high variance (it fits the quirks of the data you happened to sample)



Bagging

Now, back to bagging!

Bagging: Motivation

- Suppose we could somehow sample m independent training sets from p_{data}.
- We could then compute the prediction y_i based on each one, and take the average $y = \frac{1}{m}\sum_{i=1}^{m} y_i$.
- How does this affect the three terms of the expected loss?
  - **Bayes error: unchanged**, since we have no control over it
  - **Bias: unchanged**, since the averaged prediction has the same expectation

$$\mathbb{E}[y] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \mathbb{E}[y_i]$$

  - **Variance: reduced**, since we're averaging over independent samples

$$\mathrm{Var}[y] = \mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} y_i\right] = \frac{1}{m^2}\sum_{i=1}^{m} \mathrm{Var}[y_i] = \frac{1}{m}\mathrm{Var}[y_i].$$

Bagging: The Idea

- In practice, running an algorithm separately on independently sampled datasets is very wasteful!
- Solution: bootstrap aggregation, or bagging.
  - Take a single dataset D with n examples.
  - Generate m new datasets, each by sampling n training examples from D, with replacement.
  - Average the predictions of models trained on each of these datasets.
- The bootstrap is one of the most important ideas in all of statistics!

Bagging: The Idea

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  - Possible to show that if the sampled predictions have variance σ^2 and correlation ρ, then

$$\mathrm{Var}\left(\frac{1}{m}\sum_{i=1}^{m} y_i\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2.$$

Ironically, it can be advantageous to introduce *additional* variability into your algorithm, as long as it reduces the correlation between samples.

  - Intuition: you want to invest in a diversified portfolio, not just one stock.
  - Can help to use average over multiple algorithms, or multiple configurations of the same algorithm.

Random Forests

- Random forests = bagged decision trees, with one extra trick to decorrelate the predictions
- When choosing each node of the decision tree, choose a random set of d input features, and only consider splits on those features
- Random forests are probably the best black-box machine learning algorithm — they often work well with no tuning whatsoever.
    - one of the most widely used algorithms in Kaggle competitions

Summary

- Bagging reduces overfitting by averaging predictions.
- Used in most competition winners
  - Even if a single model is great, a small ensemble usually helps.
- Limitations:
  - Does not reduce bias.
  - There is still correlation between classifiers.
- Random forest solution: Add more randomness.