ML4 B&I: Introduction to Machine Learning

Lecture 2- Decision Trees & Ensembles

Murat A. Erdogdu

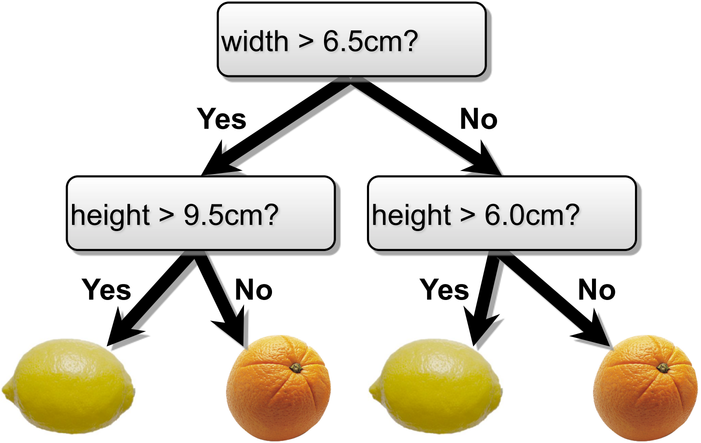
Vector Institute, Fall 2022

Today

- Announcement: A2 to be released this F, due on next F 5pm on D2L.
- Decision Trees
  - Simple but powerful learning algorithm
  - Used widely in Kaggle competitions
  - Lets us motivate concepts from information theory (entropy, mutual information, etc.)
- Bias-variance decomposition
  - Concept to motivate combining different classifiers.
- Ensemble methods
  - A commonly used technique to combine various methods.

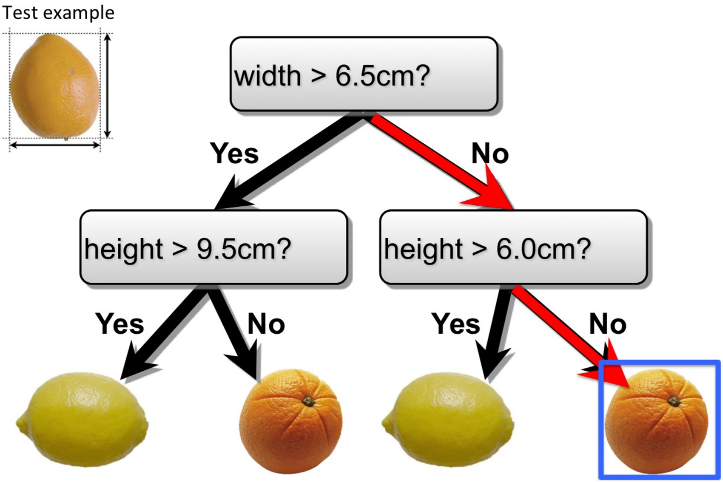
Decision Trees

- Measure attributes: width, heigth
- Make predictions by splitting on features according to a tree structure.



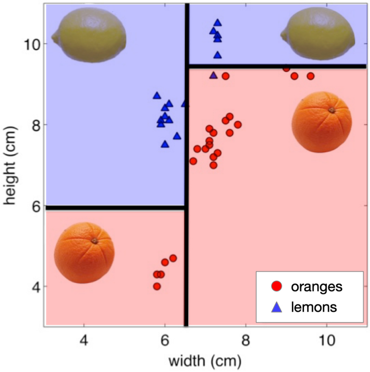
Decision Trees

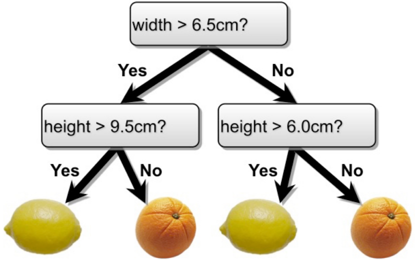
- Make predictions by splitting on features according to a tree structure.



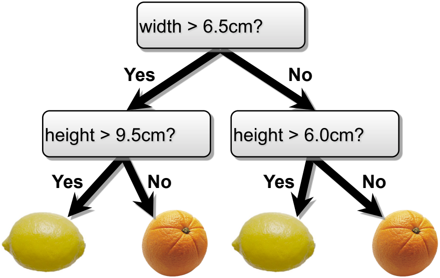
Decision Trees—Continuous Features

- Split *continuous features* by checking whether that feature is greater than or less than some threshold.
- Decision boundary is made up of axis-aligned planes.





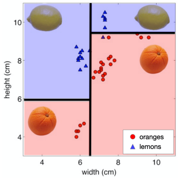
Decision Trees



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

Question: What are the hyperparameters of this model?

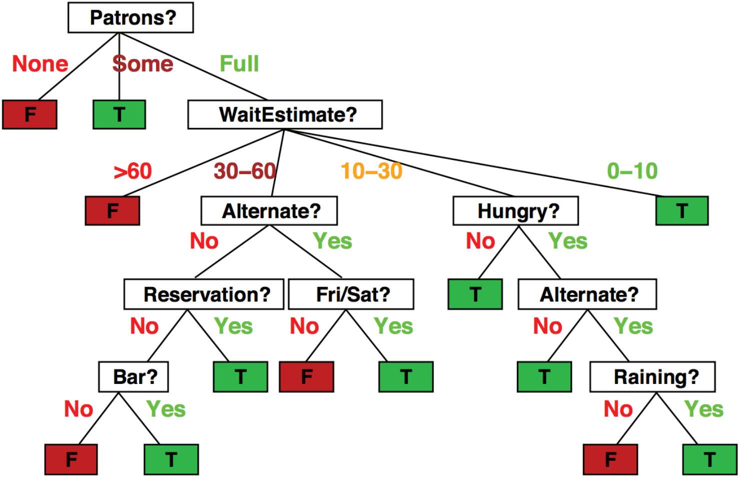
Decision Trees—Classification and Regression



- 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
- $m = 4$ on the right
- Regression tree:
  - continuous output
  - leaf value y^m typically set to the mean value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$
- Classification tree (we will focus on this):
  - discrete output
  - leaf value y^m typically set to the most common value in $\{t^{(m_1)}, \ldots, t^{(m_k)}\}$

Decision Trees—Discrete Features

- Will I eat at this restaurant?



Decision Trees—Discrete Features

- Split *discrete features* into a partition of possible values.

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*

Features:

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).

Learning Decision Trees

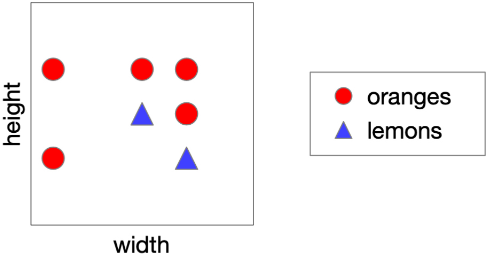
- Decision trees are universal function approximators.
  - For any training set we can construct a decision tree that has exactly the one leaf for every training point, but it probably won't generalize.
  - Example - If all D features were binary, and we had $N = 2^D$ unique training examples, a **Full Binary Tree** would have one leaf per example.
- So, how do we construct a useful decision tree?

Learning Decision Trees

- Resort to a greedy heuristic:
  - Start with the whole training set and an empty decision tree.
  - Pick a feature and candidate split that would most reduce a loss
  - Split on that feature and recurse on subpartitions.
- What is a loss?
  - When learning a model, we use a scalar number to assess whether we're on track
  - Scalar value: low is good, high is bad
- Which loss should we use?

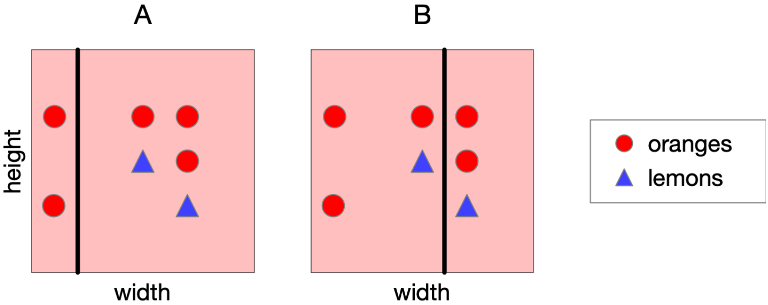
Choosing a Good Split

- Consider the following data. Let's split on width.
- Classify by majority.



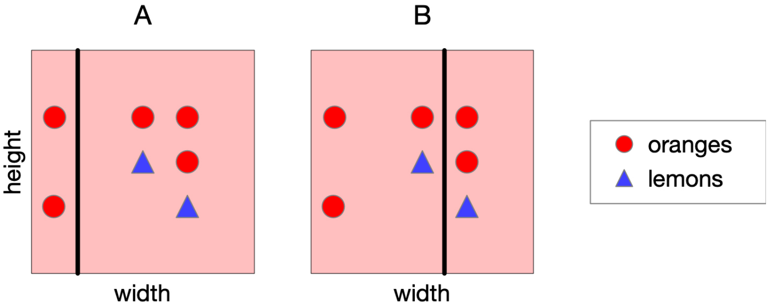
Choosing a Good Split

- Which is the best split? Vote!



Choosing a Good Split

- A feels like a better split, because the left-hand region is very certain about whether the fruit is an orange.
- Can we quantify this?



Choosing a Good Split

- How can we quantify uncertainty in prediction for a given leaf node?
  - If all examples in leaf have same class: good, low uncertainty
  - If each class has same amount of examples in leaf: bad, high uncertainty
- **Idea:** Use counts at leaves to define probability distributions; use a probabilistic notion of uncertainty to decide splits.
- A brief detour through information theory...

Entropy - Quantifying uncertainty

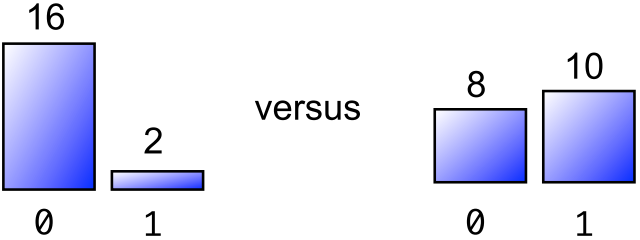
- You may have encountered the term entropy quantifying the state of chaos in chemical and physical systems,
- The entropy of a random variable quantifies the uncertainty inherent.
- To explain entropy, consider flipping two different coins...

We Flip Two Different Coins

Each coin is a binary random variable with outcomes 1 or 0:

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

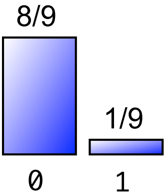
Sequence 2:
0 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 ... ?

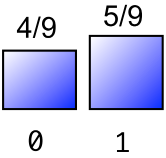


Quantifying Uncertainty

- The entropy of a loaded coin with probability p of heads is given by

$$-p\log_2(p) - (1-p)\log_2(1-p)$$





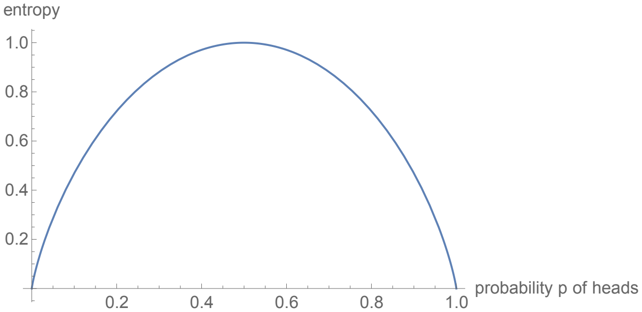
$$-\frac{8}{9}\log_2\frac{8}{9} - \frac{1}{9}\log_2\frac{1}{9} \approx \frac{1}{2}$$

$$-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 1$$

- Notice: the coin whose outcomes are more certain has a lower entropy.
- In the extreme case $p = 0$ or $p = 1$, we were certain of the outcome before observing. So, we gained no certainty by observing it, i.e., entropy is 0.

Quantifying Uncertainty

- Can also think of entropy as the expected information content of a random draw from a probability distribution.



- Claude Shannon showed: you cannot store the outcome of a random draw using fewer expected bits than the entropy without losing information.
- So units of entropy are bits; a fair coin flip has 1 bit of entropy.

Entropy

- More generally, the entropy of a discrete random variable Y is given by

$$H(Y) = -\sum_{y \in Y} p(y) \log_2 p(y)$$

- **"High Entropy"**:
  - Variable has a uniform like distribution over many outcomes
  - Flat histogram
  - Values sampled from it are less predictable
- **"Low Entropy"**
  - Distribution is concentrated on only a few outcomes
  - Histogram is concentrated in a few areas
  - Values sampled from it are more predictable

[Slide credit: Vibhav Gogate]

Entropy

- Suppose we observe partial information X about a random variable Y
  - For example, $X = \text{sign}(Y)$.
- We want to work towards a definition of the expected amount of information that will be conveyed about Y by observing X.
  - Or equivalently, the expected reduction in our uncertainty about Y after observing X.

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

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

Conditional Entropy

- 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

- 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:

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

Conditional Entropy

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

	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 does not affect our uncertainty about Y: $H(Y|X) = H(Y)$
  - But knowing Y makes our knowledge of Y certain: $H(Y|Y) = 0$
  - 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 more certain am I about whether it's cloudy if I'm told whether it is raining? My uncertainty in Y minus my expected uncertainty that would remain in Y after seeing X.
- This is called the information gain $IG(Y|X)$ in Y due to X, or the mutual information of Y and X

$$IG(Y|X) = H(Y) - H(Y|X) \tag{1}$$

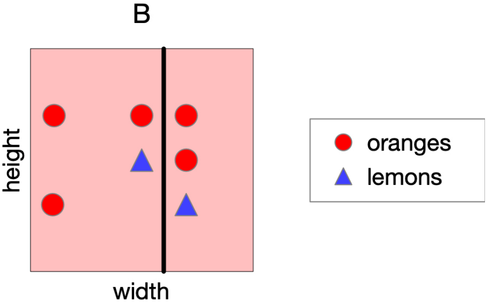
- If X is completely uninformative about Y: $IG(Y|X) = 0$
- If X is completely informative about Y: $IG(Y|X) = H(Y)$

Revisiting Our Original Example

- Information gain measures the informativeness of a variable, which is exactly what we desire in a decision tree split!
- The information gain of a split: how much information (over the training set) about the class label Y is gained by knowing which side of a split you're on.

Information Gain of Split B

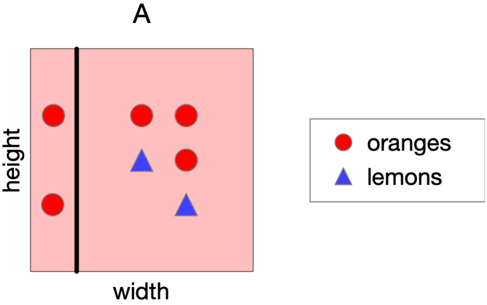
- What is the information gain of split B? Not terribly informative...



- Entropy of class outcome before split:
$H(Y) = -\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}) \approx 0.86$
- Conditional entropy of class outcome after split:
$H(Y|left) \approx 0.81$, $H(Y|right) \approx 0.92$
- $IG(split) \approx 0.86 - (\frac{4}{7} \cdot 0.81 + \frac{3}{7} \cdot 0.92) \approx 0.006$

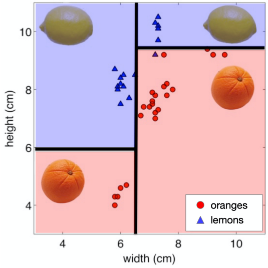
Information Gain of Split A

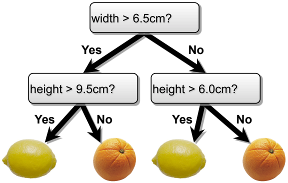
- What is the information gain of split A? Very informative!



- Entropy of class outcome before split:
  $H(Y) = -\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}) \approx 0.86$
- Conditional entropy of class outcome after split:
  $H(Y|left) = 0$, $H(Y|right) \approx 0.97$
- $IG(split) \approx 0.86 - (\frac{2}{7} \cdot 0 + \frac{5}{7} \cdot 0.97) \approx 0.17!!$

Constructing Decision Trees





- At each level, one must choose:
  1. Which feature to split.
  2. Possibly where to split it.
- Choose them based on how much information we would gain from the decision! (choose feature that gives the highest gain)

Decision Tree Construction Algorithm

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

  Loop:

  1. pick a feature to split at a non-terminal node
  2. split examples into groups based on feature value

- Terminates when all leaves contain only examples in the same class or are empty.
- Choose
  - the feature to split on and the split (threshold)
  - that gives the highest information gain.

Back to Our Example

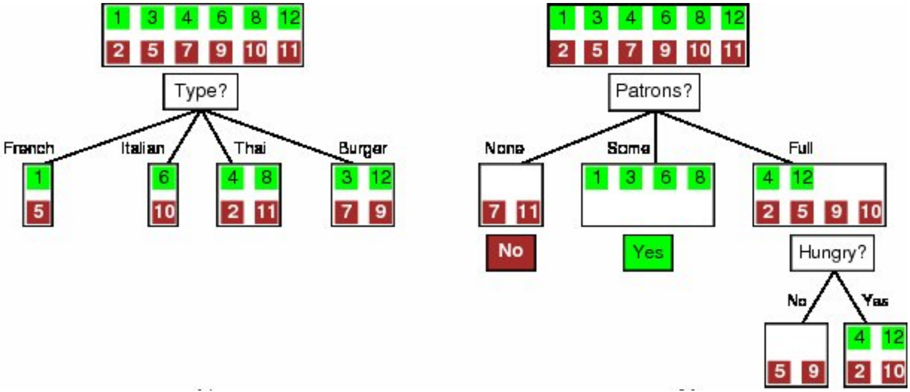
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	Yes	No	No	Yes	Some	$$$	No	Yes	French	0–10	$y_1 =$ Yes
$\mathbf{x}_2$	Yes	No	No	Yes	Full	$	No	No	Thai	30–60	$y_2 =$ No
$\mathbf{x}_3$	No	Yes	No	No	Some	$	No	No	Burger	0–10	$y_3 =$ Yes
$\mathbf{x}_4$	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10–30	$y_4 =$ Yes
$\mathbf{x}_5$	Yes	No	Yes	No	Full	$$$	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

Features:

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]

Feature Selection

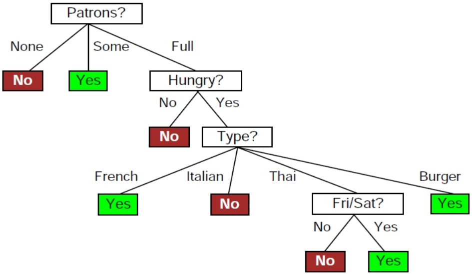


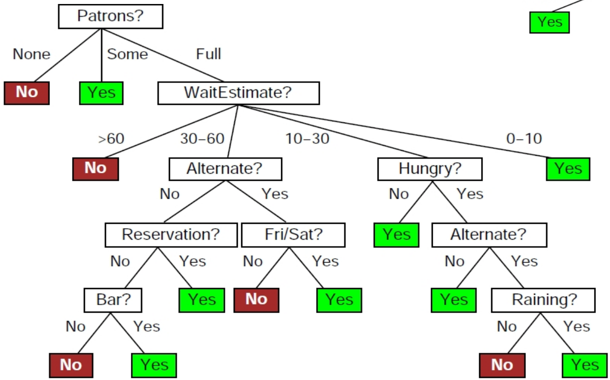
$$IG(Y) = H(Y) - H(Y|X)$$

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

$$IG(Patrons) = 1 - \left[\frac{2}{12}H(Y|\text{Non}) + \frac{4}{12}H(Y|\text{Some}) + \frac{6}{12}H(Y|\text{Full})\right] \approx 0.541$$

Which Tree is Better? Vote!





What Makes a Good Tree?

- Not too small: need to handle important but possibly subtle distinctions in data
- Not too big:
  - Avoid over-fitting training examples
  - Computational efficiency (avoid redundant, spurious attributes)
  - Human interpretability
- We desire small trees with informative nodes near the root

KNN versus Decision Trees

Advantages of decision trees over KNNs

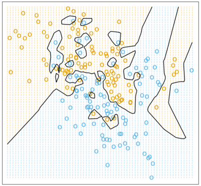
- Simple to deal with discrete features, missing values, and poorly scaled data
- Fast at test time
- More interpretable

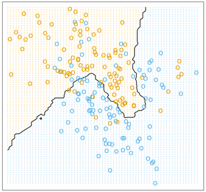
Advantages of KNNs over decision trees

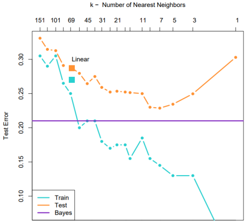
- Few hyperparameters
- Can incorporate interesting distance measures (e.g. shape contexts)

Bias-Variance Decomposition

- Overly simple models underfit the data, and overly complex models overfit.
- We can quantify underfitting and overfitting in terms of the bias/variance decomposition.





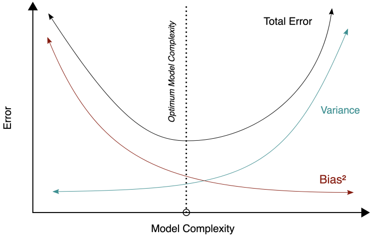


Bias variance tradeoff

$$\text{test error} = \text{bias}^2 + \text{variance}$$

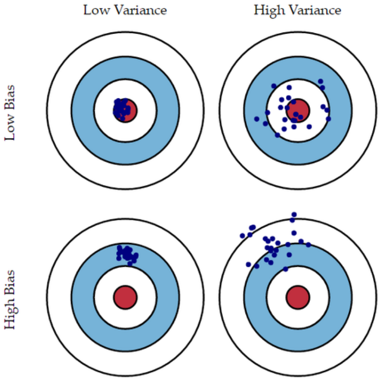
We split the expected loss into three terms:

- bias: how wrong the expected prediction is (corresponds to underfitting – small decision tree)
- variance: the amount of variability in the predictions (corresponds to overfitting – huge decision tree)



Bias and Variance

- Throwing darts = predictions for each draw of a dataset



Ensemble methods: Bagging

- Suppose we could somehow sample m independent training sets $\{\mathcal{D}_i\}_{i=1}^m$ from p_{dataset}.
- We could then learn a predictor $h_i := h_{\mathcal{D}_i}$ based on each one, and take the average $h = \frac{1}{m}\sum_{i=1}^m h_i$.
- How does this affect the performance?
  - **Bias: unchanged**, since the averaged prediction has the same expectation

$$\mathbb{E}_{\mathcal{D}_1,\ldots,\mathcal{D}_m \overset{iid}{\sim} p_{\text{dataset}}}[h(\mathbf{x})] = \frac{1}{m}\sum_{i=1}^m \mathbb{E}_{\mathcal{D}_i \sim p_{\text{dataset}}}[h_i(\mathbf{x})] = \mathbb{E}_{\mathcal{D}\sim p_{\text{dataset}}}[h_{\mathcal{D}}(\mathbf{x})]$$

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

$$\underset{\mathcal{D}_1,\ldots,\mathcal{D}_m}{\mathrm{Var}}[h(\mathbf{x})] = \frac{1}{m^2}\sum_{i=1}^m \underset{\mathcal{D}_i}{\mathrm{Var}}[h_i(\mathbf{x})] = \frac{1}{m}\underset{\mathcal{D}}{\mathrm{Var}}[h_{\mathcal{D}}(\mathbf{x})].$$

What if $m \to \infty$?

Bagging: The Idea

- In practice, we don't have access to the underlying data generating distribution p_{sample}.
- It is expensive to collect many i.i.d. datasets from p_{dataset}.
- Solution: **bootstrap aggregation**, or **bagging**.
  - Take a single dataset $\mathcal{D}$ with n examples.
  - Generate m new datasets, each by sampling n training examples from $\mathcal{D}$, with replacement.
  - Average the predictions of models trained on each of these datasets.

Bagging: The Idea

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  - Still helps reduce the variance.
- Ironically, it can be advantageous to introduce *additional* variability into your algorithm, as long as it reduces the correlation between samples.
  - 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
- The main idea in random forests is to improve the variance reduction of bagging by reducing the correlation between the trees.
- 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

Conclusion

- Decision trees are simple and interpretable models.
- Complexity of the model impacts the test error through bias-variance decomposition
- Ensemble methods can be used to "trick" bias-variance tradeoff.
- Next lecture, we focus on linear regression.