Decision and Regression Trees

Applied Machine Learning
Derek Hoiem

Dall-E: A dirt road splits around a large gnarly tree, fractal art

Logistics

More course staff introductions

- Josh Levine (TA)
- Kshitij Phulare (TA)
- Pedro Cisneros (Post-doc Course Assistant)

All course materials: https://courses.engr.illinois.edu/cs441/sp2023/

- See CampusWire for Office Hours

Survey by email

- Will spend some time going over key concepts Thursday

Recap of previous lectures

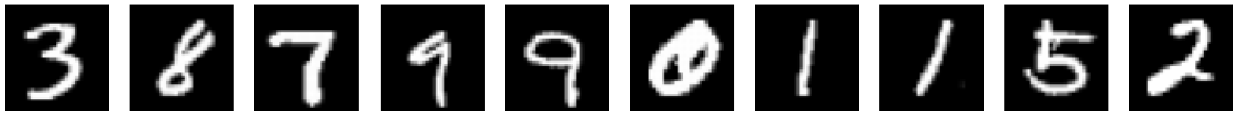
- Nearest neighbor is widely used
  - Super-powers: can instantly learn new classes and predict from one or many examples
- Naïve Bayes represents a common assumption as part of density estimation, more typical as part of an approach rather than the final predictor
  - Super-powers: Fast estimation from lots of data; not terrible estimation from limited data
- Logistic Regression is widely used
  - Super-powers: Effective prediction from high-dimensional features; good confidence estimates
- Linear Regression is widely used
  - Super-powers: Can extrapolate, explain relationships, and predict continuous values from many variables
- Almost all algorithms involve nearest neighbor, logistic regression, or linear regression
  - The main learning challenge is typically **feature learning**

HW 1 summary

MNIST Digit Classification

Predict label (0-9) from pixel intensities (784x1 vector)

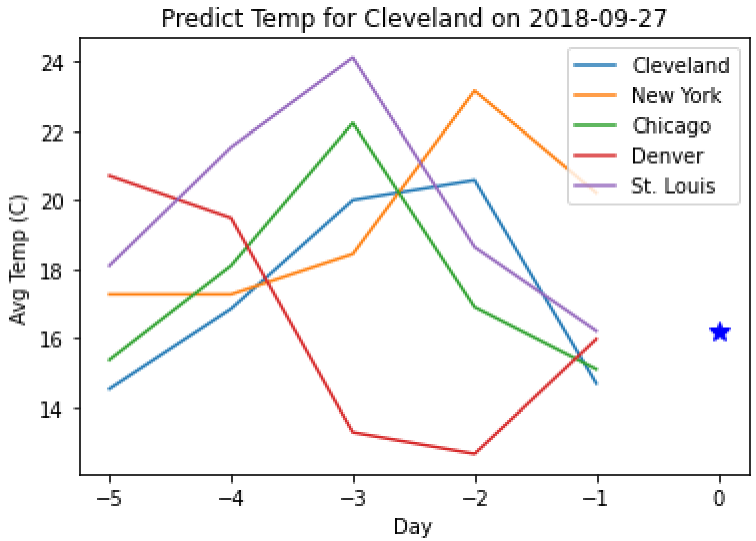
1. Implement and test KNN, Naïve Bayes, Linear Logistic Regression
2. Plot Error vs Training Size
3. Select best parameter using validation



Temperature Regression

Predict Cleveland's next day temperature from recent temperatures of US cities

1. Implement and test KNN, Naive Bayes (NB), and Linear Regression (LR)
2. Identify most important features with L1 linear regression, and re-train/evaluate with most important features



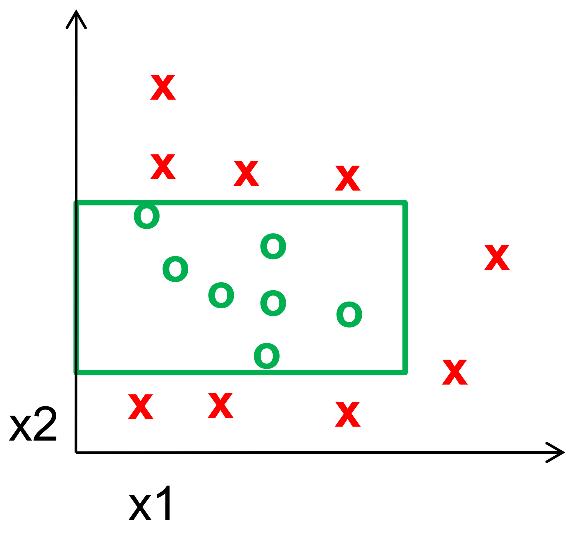
Stretch Goals

1. Improve MNIST Classification
2. Improve Temperature Regression
3. Generate a train/test classification dataset in which Naive Bayes outperforms 1-NN and Logistic Regression

Completing HWs

- Read assignment and tips
- Code by adding to starter code notebook (which mainly has data loading and visualization functions)
- Complete report, including expected points
- Submit the report, notebook pdf/html, and zip/ipynb code
  - Mainly grader will look at report first, notebook pdf for clarification, and run code rarely
  - Notebook does not need to include all outputs

- So far, we’ve seen two main choices for how to use features
  1. Nearest neighbor uses all the features jointly to find similar examples
  2. Linear models make predictions out of weighted sums of the features
- If you wanted to give someone a rule to split the ‘o’ from the ‘x’, what other idea might you try?

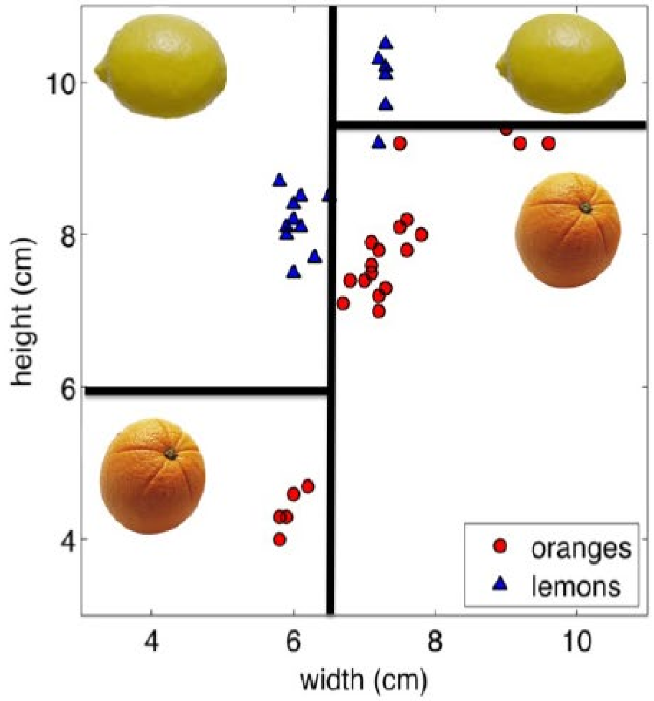



```
If x2 < 0.6 and x2 > 0.2 and x2 < 0.7, ‘o’
Else ‘x’
```

Can we learn how to make these kinds of rules automatically?

Decision trees

- Training: Iteratively choose the attribute and split value that best separates the classes for the data in the current node
- Combines feature selection/modeling with prediction



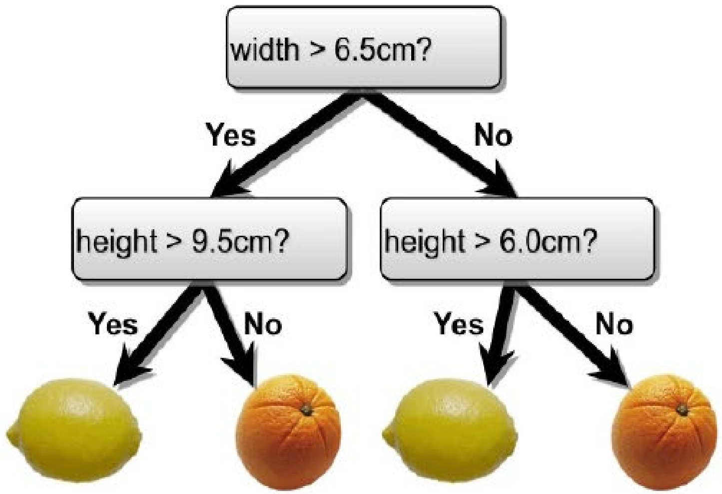
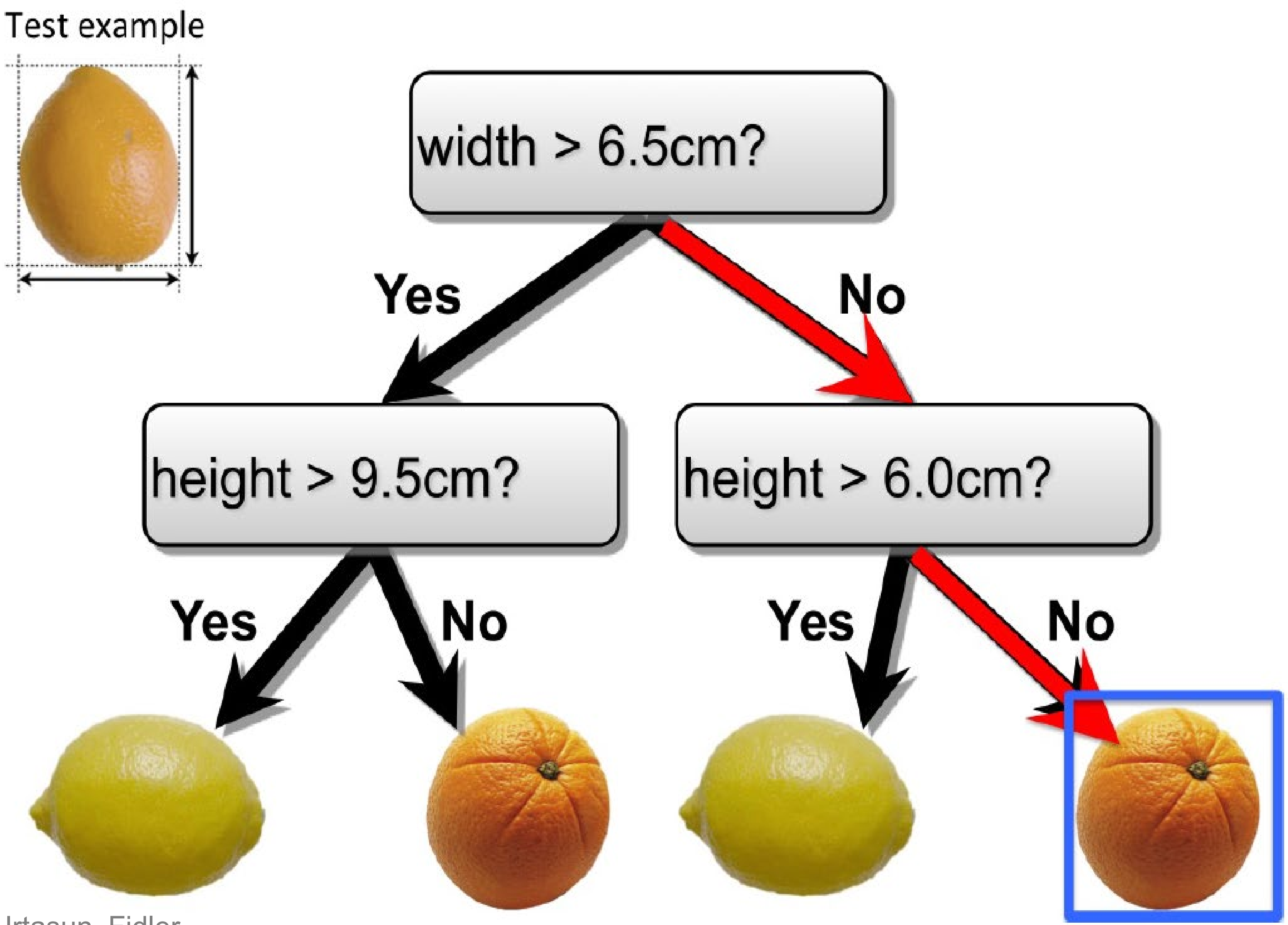


Fig Credit: Zemel, Urtasun, Fidler

Decision Tree Classification



Slide Credit: Zemel, Urtasun, Fidler

Example with discrete inputs

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).

Example with discrete inputs

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).

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

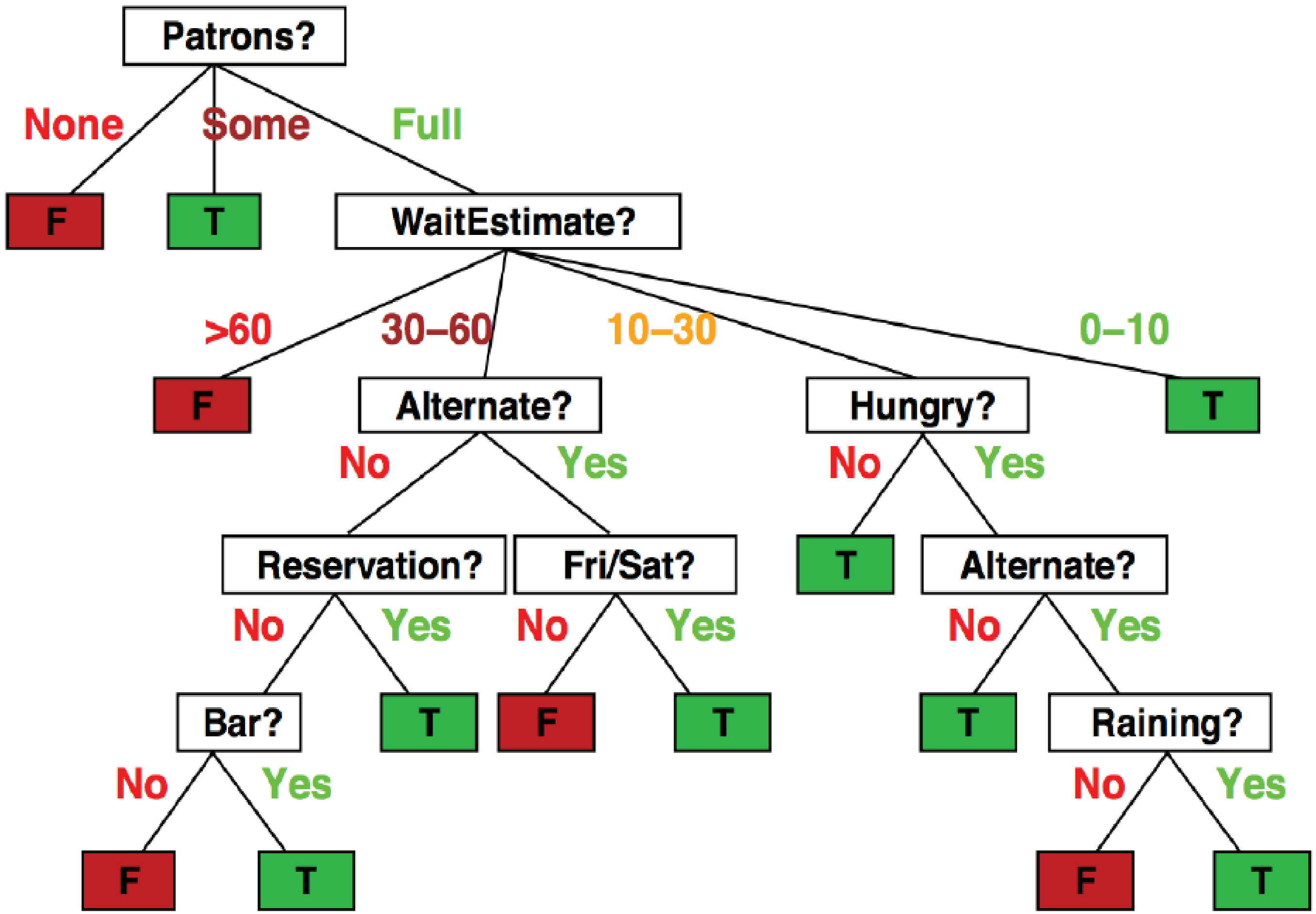
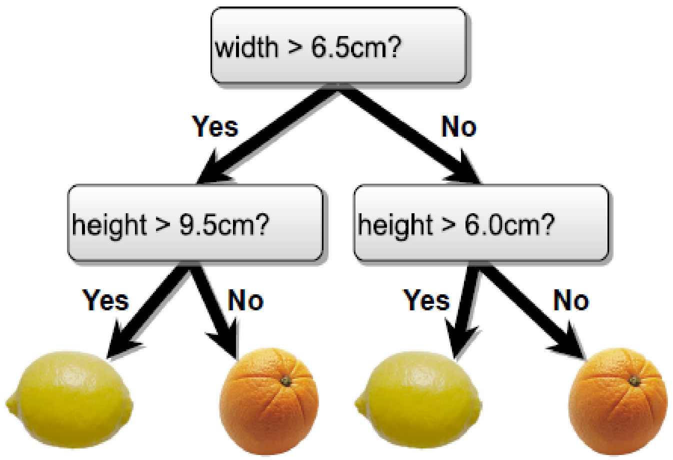


Figure Source: Zemel, Urtasun, Fidler

Decision Trees



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

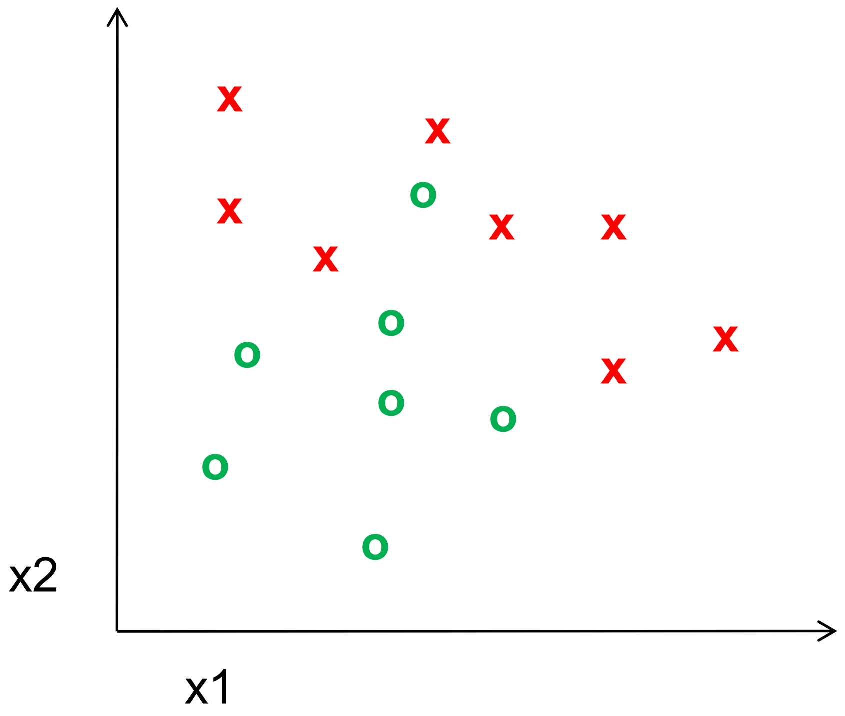
Figure Source: Zemel, Urtasun, Fidler

Decision tree algorithm

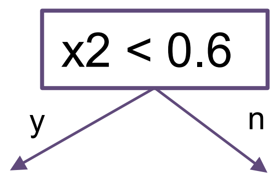
Training

Recursively, for each node in tree:

1. If labels in the node are mixed:
   a. Choose attribute and split values based on data that reaches each node
   b. Branch and create 2 (or more) nodes
2. Return



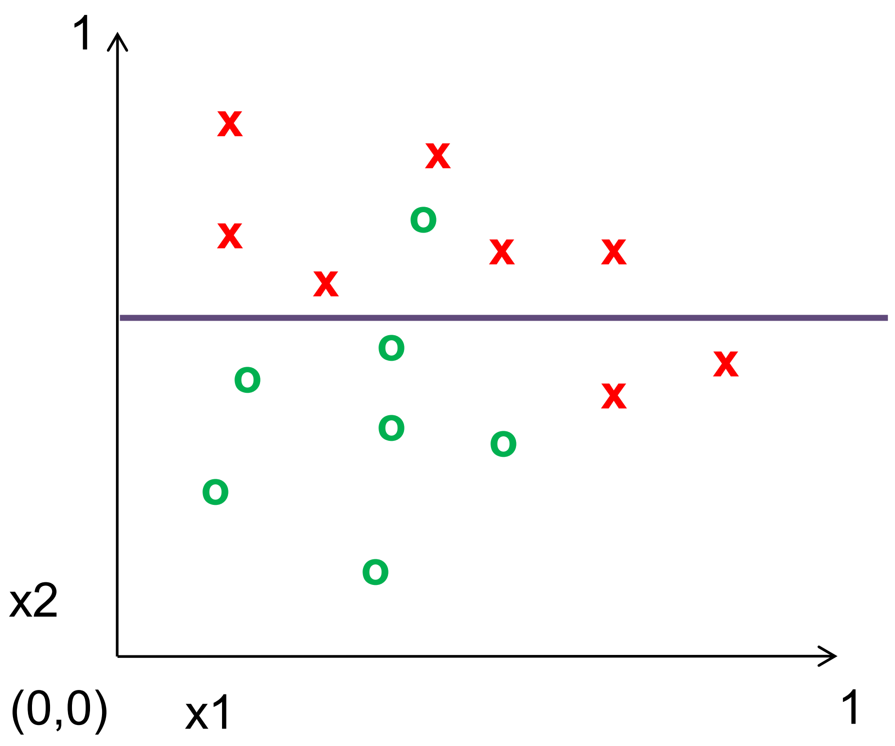
Decision tree algorithm



Training

Recursively, for each node in tree:

1. If labels in the node are mixed:
   a. Choose attribute and split values based on data that reaches each node
   b. Branch and create 2 (or more) nodes
2. Return

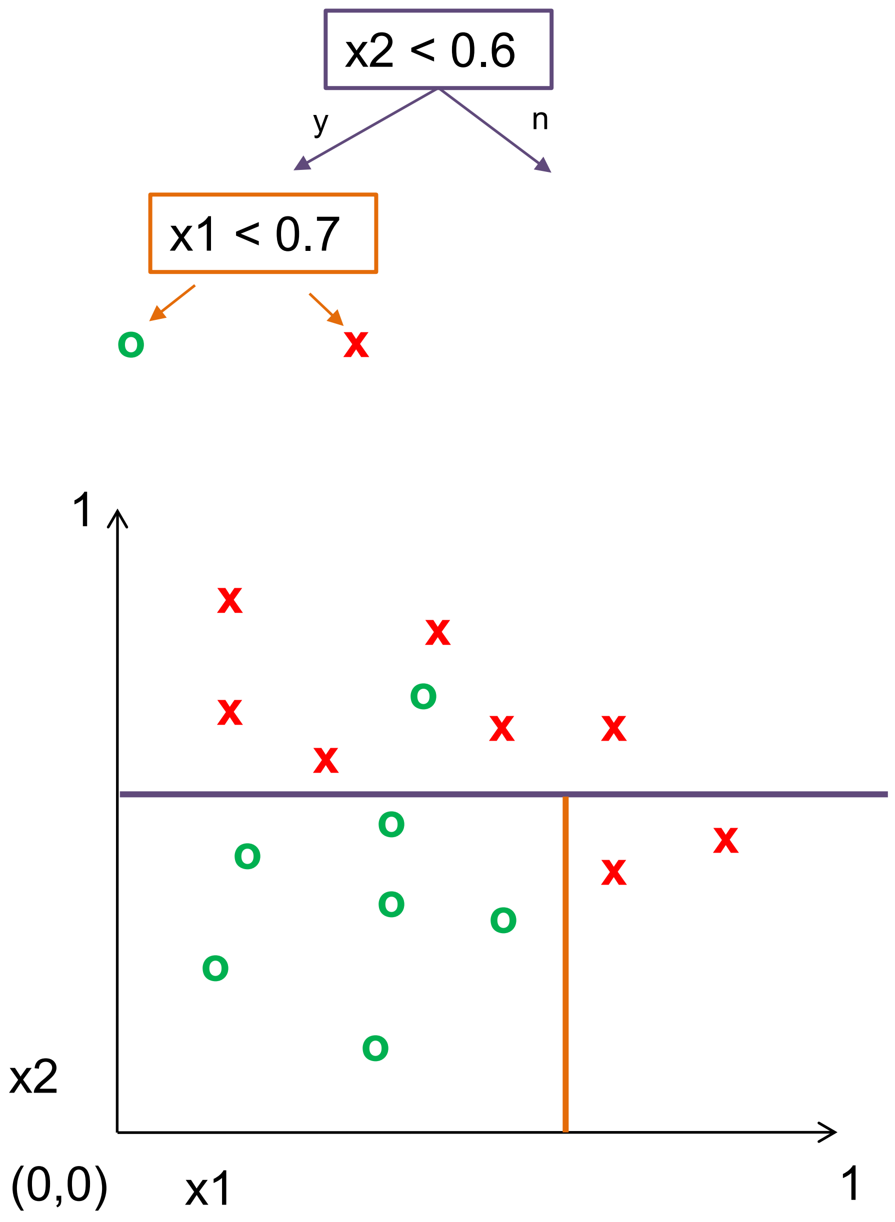


Decision tree algorithm

Training

Recursively, for each node in tree:

1. If labels in the node are mixed:
   a. Choose attribute and split values based on data that reaches each node
   b. Branch and create 2 (or more) nodes
2. Return

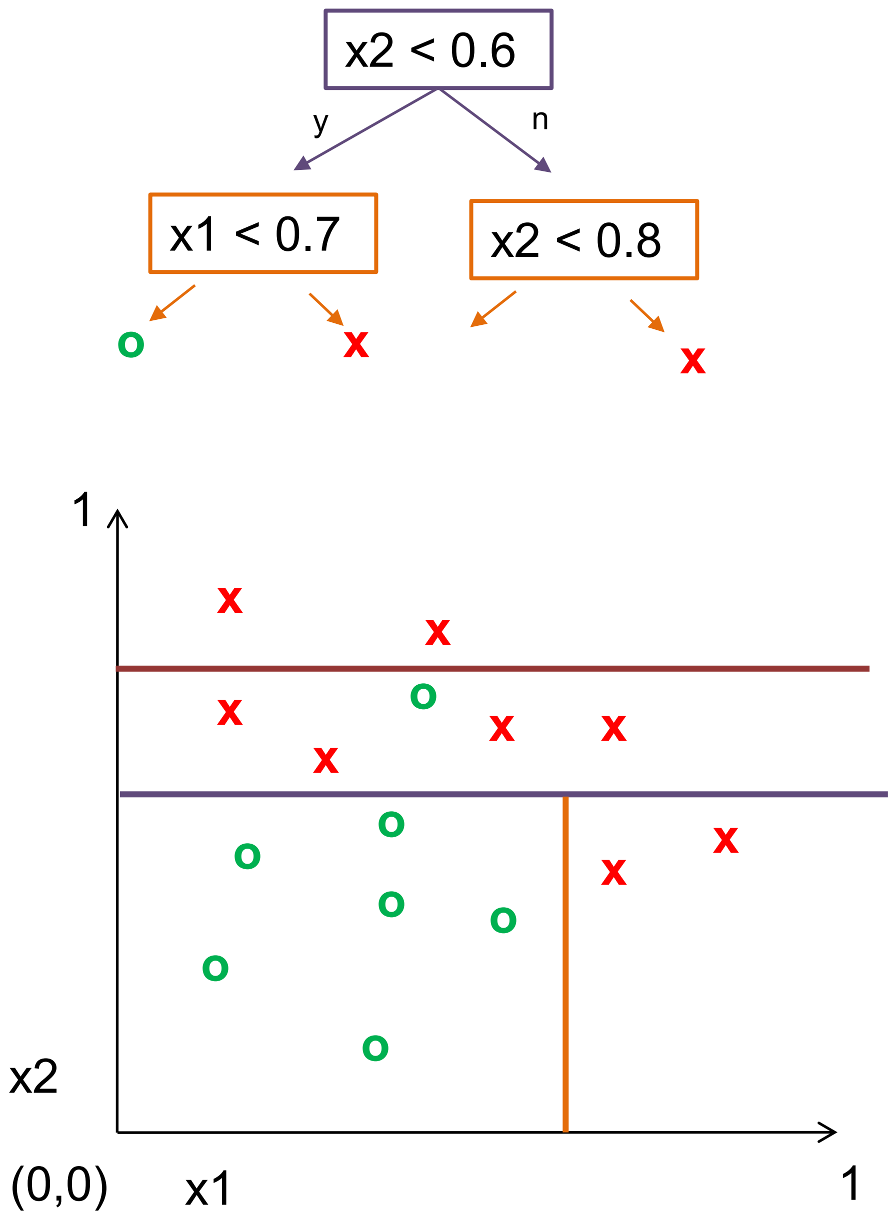


Decision tree algorithm

Training

Recursively, for each node in tree:

1. If labels in the node are mixed:
   a. Choose attribute and split values based on data that reaches each node
   b. Branch and create 2 (or more) nodes
2. Return

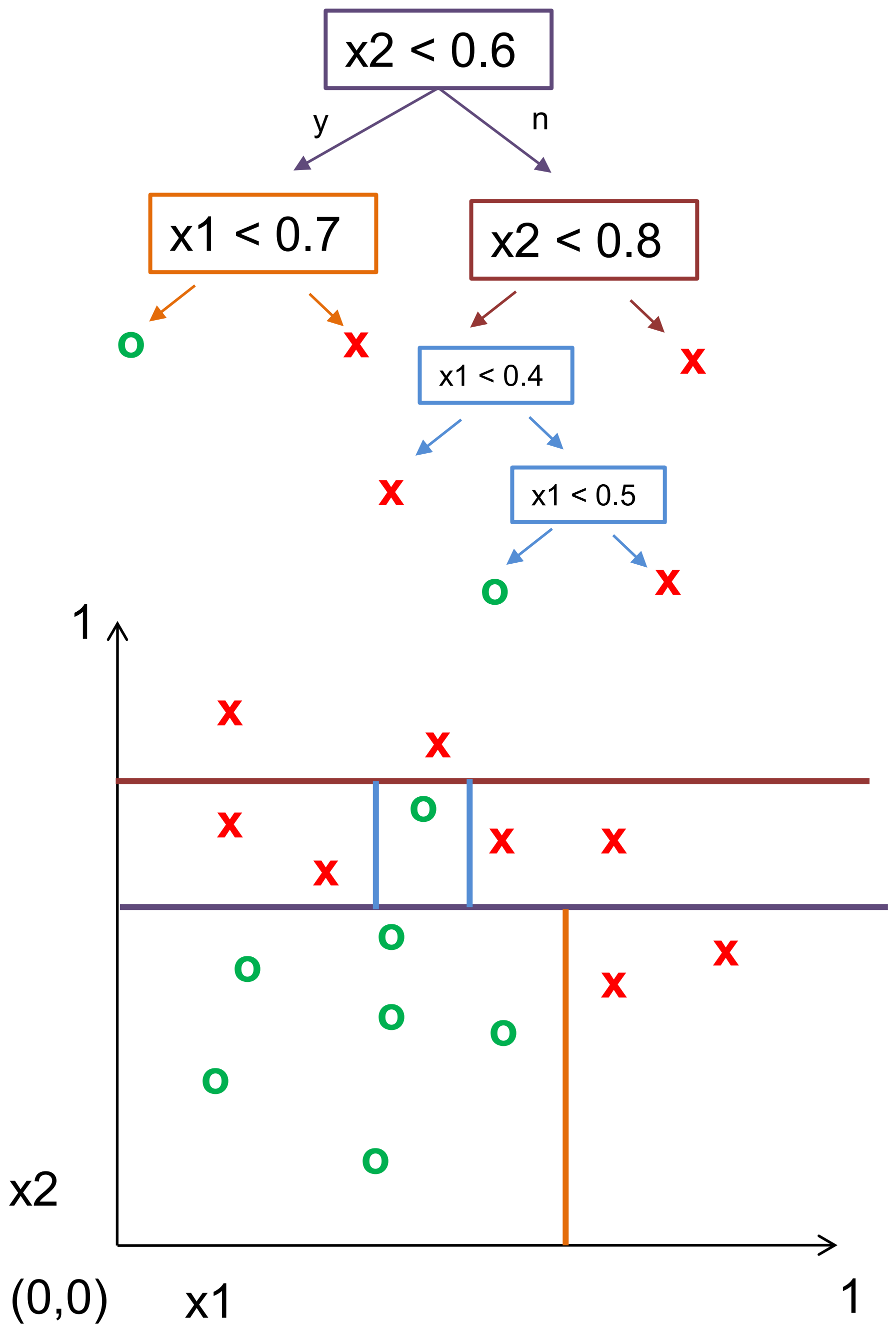


Decision tree algorithm

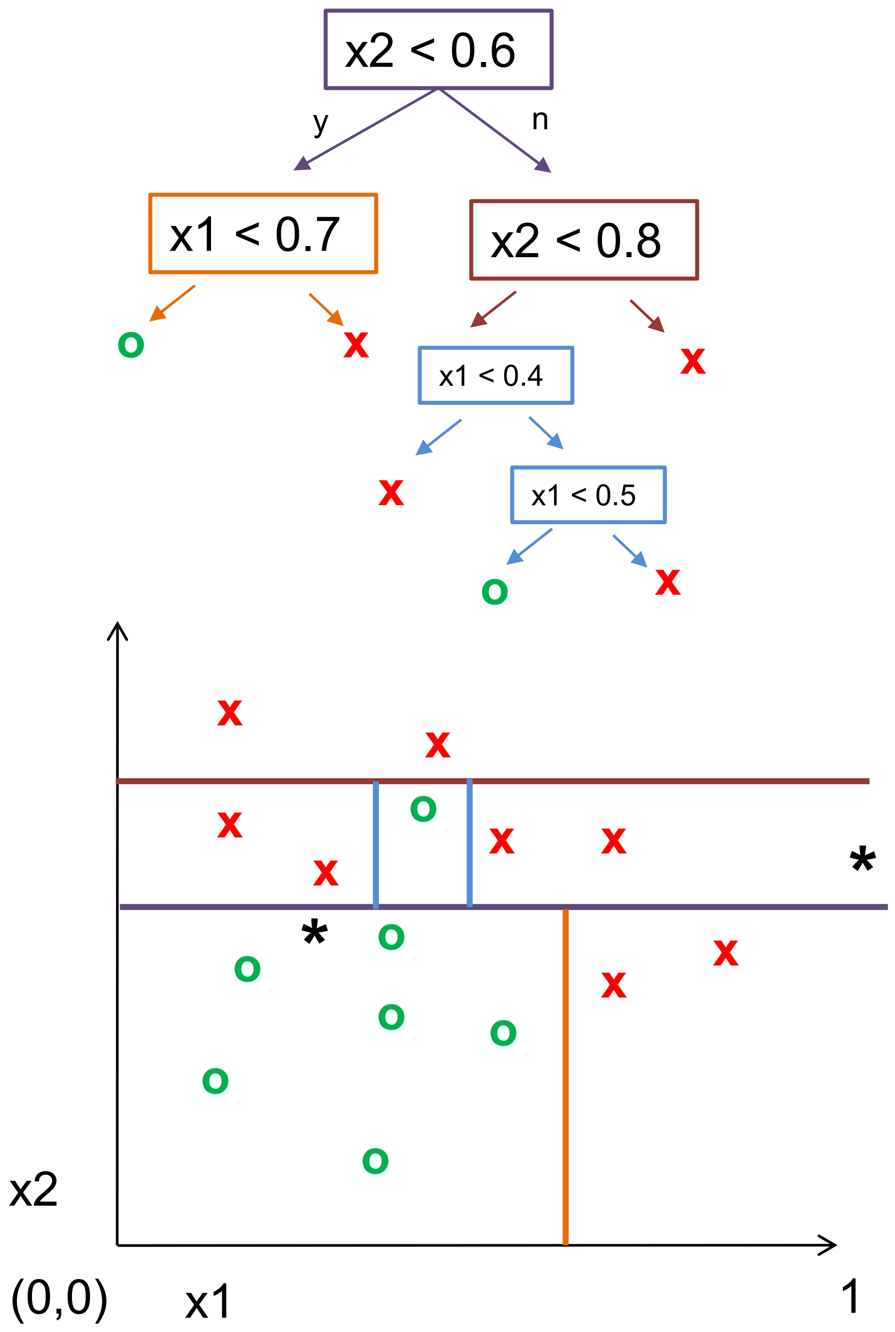
Training

Recursively, for each node in tree:

1. If labels in the node are mixed:
   a. Choose attribute and split values based on data that reaches each node
   b. Branch and create 2 (or more) nodes
2. Return



Decision tree algorithm

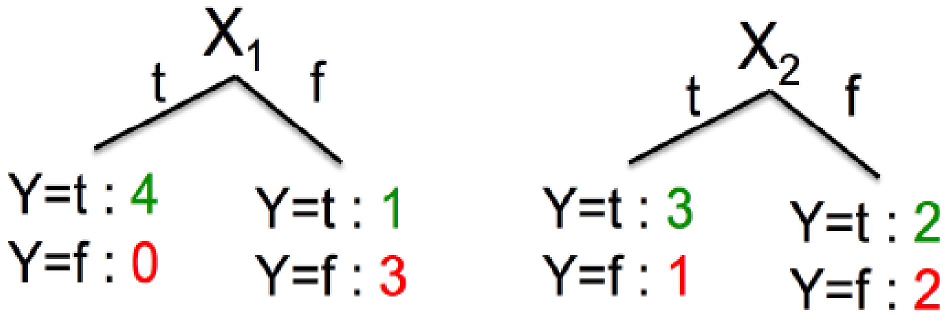


Prediction

1. Check conditions to descend tree
2. Return label of leaf node

How do you choose what/where to split?

- Which attribute is better to split on, X_1 or X_2?



X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

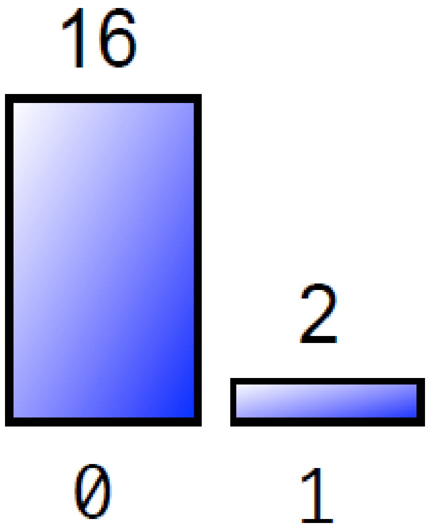
Idea: Use counts at leaves to define probability distributions, so we can measure uncertainty

Slide Source: Zemel, Urtasun, Fidler

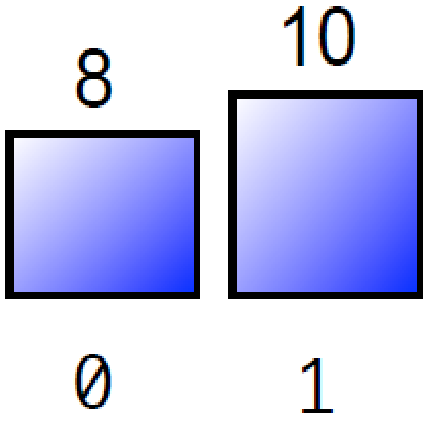
Quantifying Uncertainty: Coin Flip Example

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 ... ?



versus

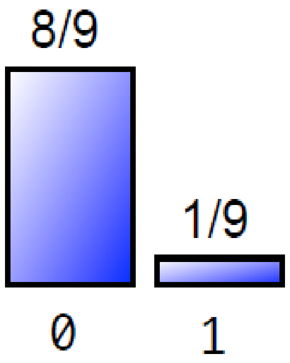


Slide Source: Zemel, Urtasun, Fidler

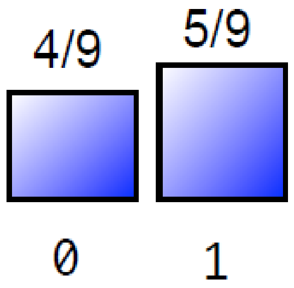
Quantifying Uncertainty: Coin Flip Example

Entropy H:

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



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



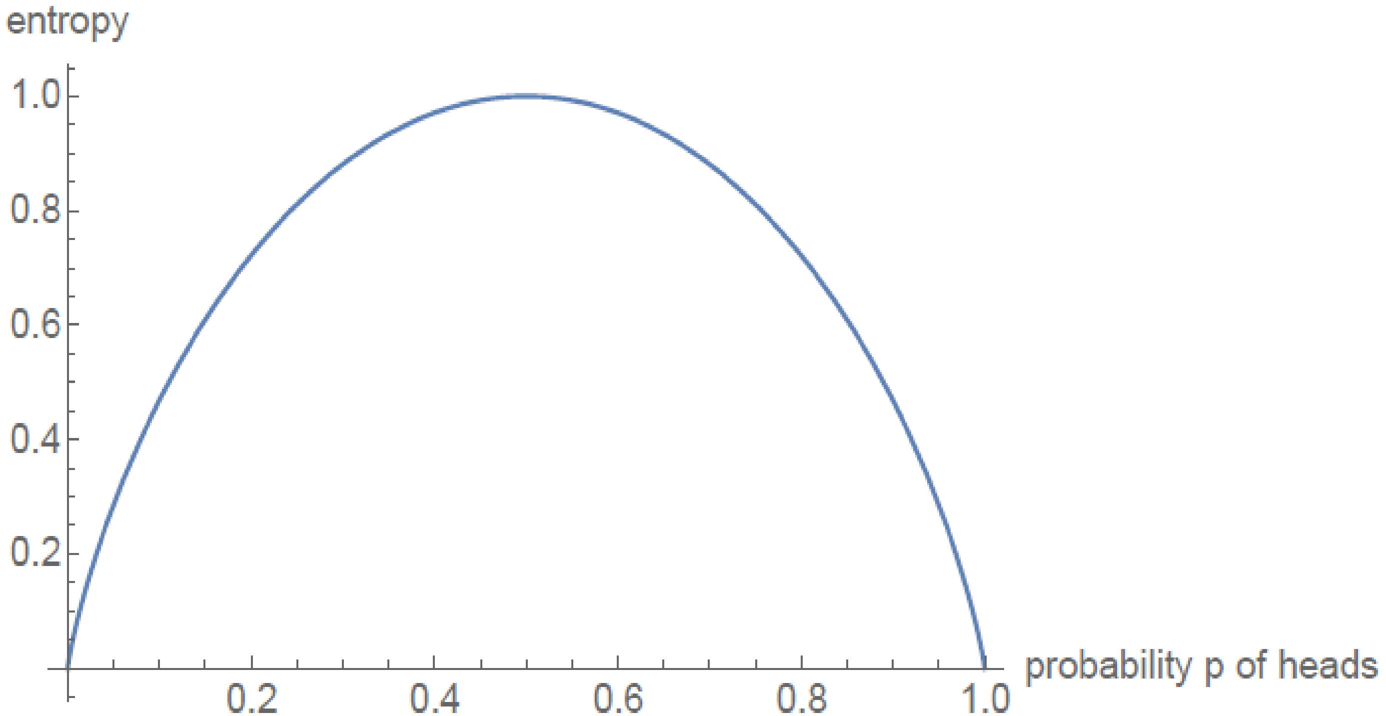
$$-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9} \approx 0.99$$

- How surprised are we by a new value in the sequence?
- How much information does it convey?

Slide Source: Zemel, Urtasun, Fidler

Quantifying Uncertainty: Coin Flip Example

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



Slide Source: Zemel, Urtasun, Fidler

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

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

Slide Source: Zemel, Urtasun, Fidler

Specific 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)

Slide Source: Zemel, Urtasun, Fidler

Conditional Entropy

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

- The expected conditional entropy:

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

Slide Source: Zemel, Urtasun, Fidler

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?

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

Slide Source: Zemel, Urtasun, Fidler

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: $H(Y|X) \leq H(Y)$

Slide Source: Zemel, Urtasun, Fidler

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?

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

- Also called information gain in Y due to X
- If X is completely uninformative about Y: $IG(Y|X) = 0$
- If X is completely informative about Y: $IG(Y|X) = H(Y)$
- How can we use this to construct our decision tree?

Slide Source: Zemel, Urtasun, Fidler

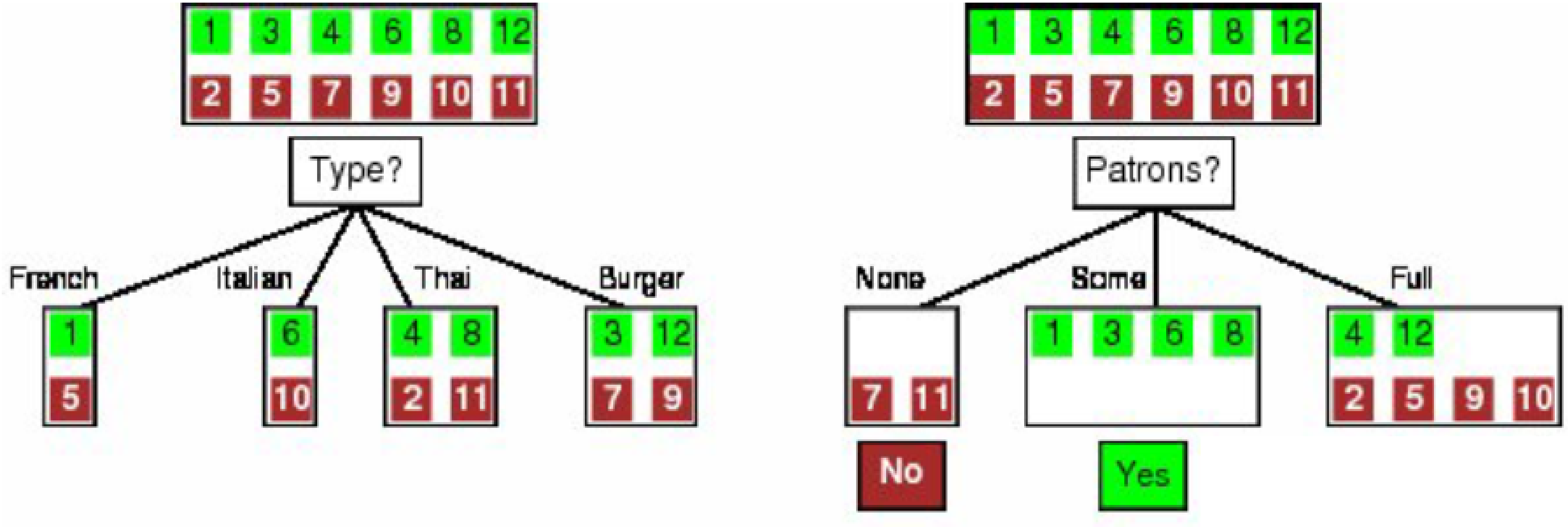
Constructing decision tree

Training

Recursively, for each node in tree:

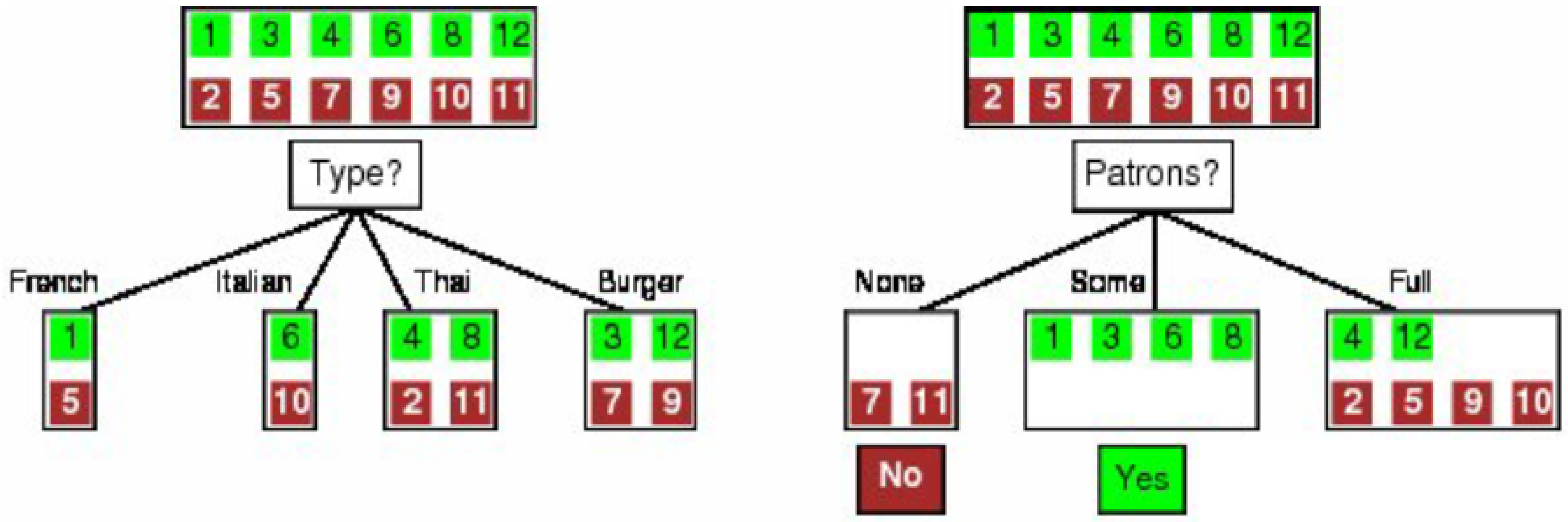
1. If labels in the node are mixed:
   a. Choose attribute and split values based on data that reaches each node
   b. Branch and create 2 (or more) nodes
2. Return

1. Measure information gain
   - For each discrete attribute: compute information gain of split
   - For each continuous attribute: select most informative threshold and compute its information gain. Can be done efficiently based on sorted values.
2. Select attribute / threshold with highest information gain



Pause, stretch, and think: Is it better to split based on type or patrons?

Slide Source: Zemel, Urtasun, Fidler



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

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

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

Slide Source: Zemel, Urtasun, Fidler

What if you need to predict a continuous value?

- Regression Tree
  - Same idea, but choose splits to minimize sum squared error $\sum_{n \in node}(f_{node}(x_n) - y_n)^2$
  - $f_{node}(x_n)$ returns the average prediction value of data points in the leaf node containing x_n

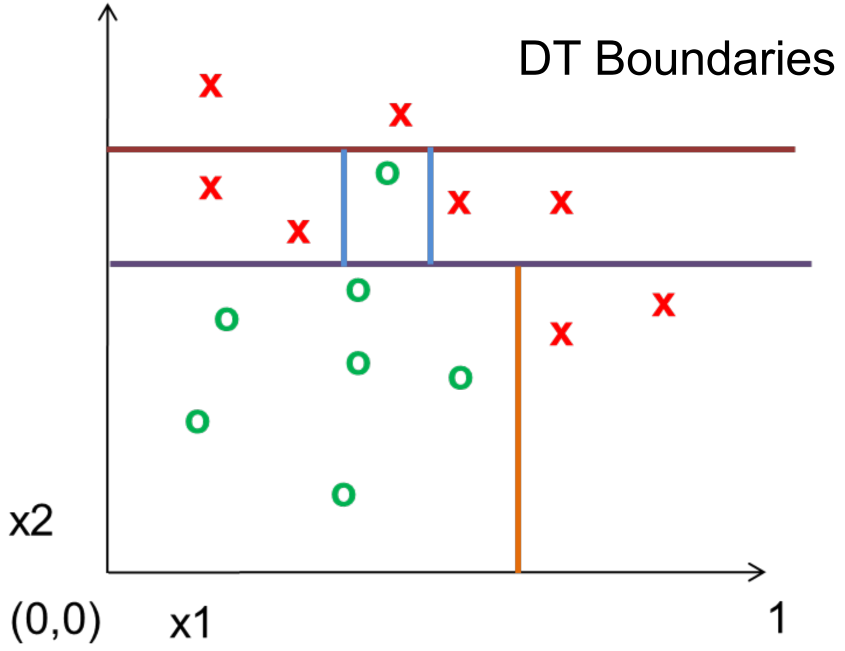
Variants

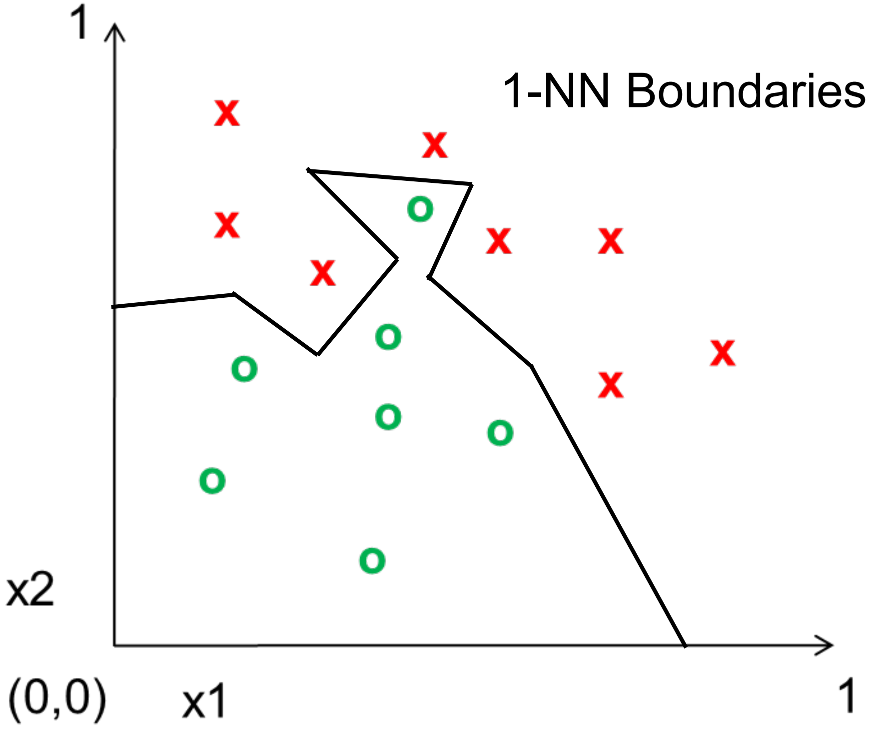
- Different splitting criteria, e.g. Gini index: $1 - \sum_i p_i^2$ (very similar result, a little faster to compute)
- Most commonly, split on one attribute at a time
  – In case of continuous vector data, can also split on linear projections of features
- Can stop early
  – when leaf node contains fewer than N_{min} points
  – when max tree depth is reached
- Can also predict multiple continuous values or multiple classes

Decision Tree vs. 1-NN

- Both have piecewise-linear decisions
- Decision tree is typically "axis-aligned"
- Decision tree has ability for early stopping to improve generalization

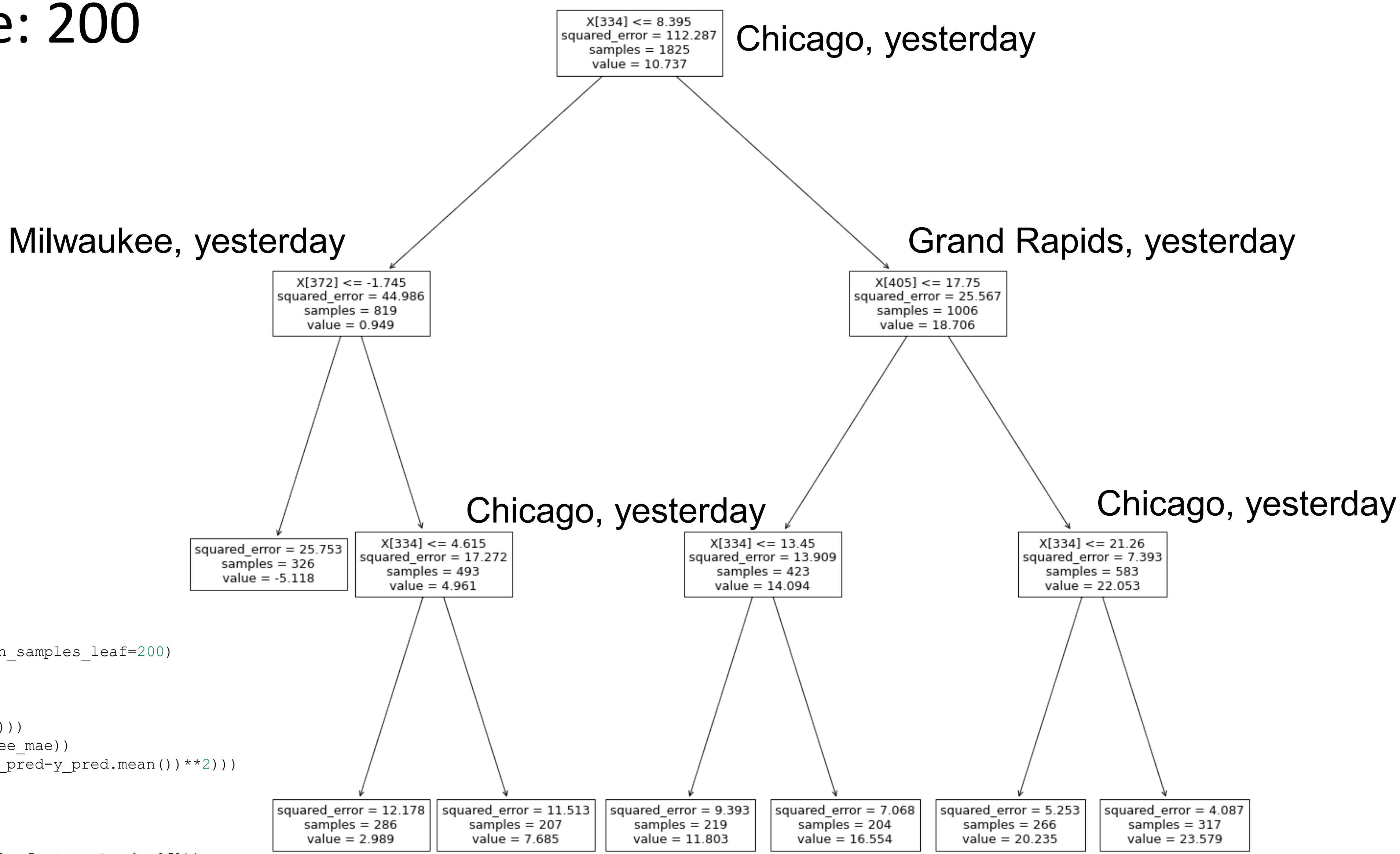
- True power of decision trees arrives with ensembles (lots of small or randomized trees)





Regression Tree for Temperature Prediction

- Min leaf size: 200
- RMSE= 3.42
- R^2=0.88



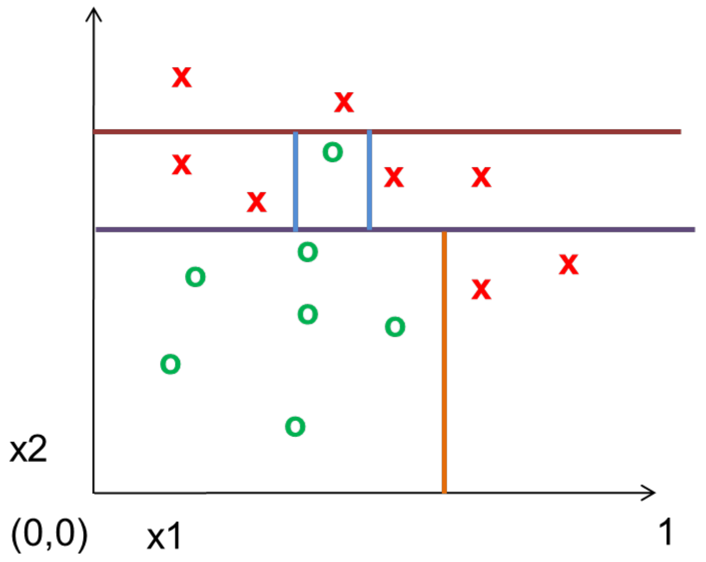

```
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(random_state=0, min_samples_leaf=200)
model.fit(x_train, y_train)
y_pred = model.predict(x_val)
tree_rmse = np.sqrt(np.mean((y_pred-y_val)**2))
tree_mae = np.sqrt(np.median(np.abs(y_pred-y_val)))
print('LR: RMSE={}, MAE={}'.format(tree_rmse, tree_mae))
print('R^2: {}'.format(1-tree_rmse**2/np.mean((y_pred-y_pred.mean())**2)))
plt.figure(figsize=(20,20))
tree.plot_tree(model)
plt.show()
for f in [334, 372, 405]:
  print('{}: {}, {}'.format(f, feature_to_city[f], feature_to_day[f]))
```

Classification/Regression Trees Summary

- Key Assumptions
  - Samples with similar features have similar predictions
- Model Parameters
  - Tree structure with split criteria at each internal node and prediction at each leaf node
- Designs
  - Limits on tree growth
  - What kinds of splits are considered
  - Criterion for choosing attribute/split (e.g. gini impurity score is another common choice)
- When to Use
  - Want an explainable decision function (e.g. for medical diagnosis)
  - As part of an ensemble (as we'll see Thursday)
- When Not to Use
  - One tree is not a great performer, but a forest is

Things to remember

- Decision/regression trees learn to split up the feature space into partitions with similar values



- Entropy is a measure of uncertainty

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

- Information gain measures how much particular knowledge reduces prediction uncertainty

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

Thursday

- Ensembles: model averaging and forests