Announcements:
- Project Milestone feedback this week
- Thu this week: HW3 due / HW4 released (start early 😊)

Large-Scale Machine Learning (1)
New Topic: ML that scales!

High dim. data
- Locality sensitive hashing
- Clustering
- Dimensionality reduction

Graph data
- PageRank, SimRank
- Community Detection
- Spam Detection

Infinite data
- Sampling data streams
- Filtering data streams
- Queries on streams

Machine learning
- Decision Trees
- SVM / Max Margin
- Parallel SGD

Apps
- Recommender systems
- Association Rules
- Duplicate document detection
Supervised Learning

Given some data:

- “Learn” a function to map from the input to the output

- Given:
  
  Training examples \((x_i, y_i = f(x_i))\) for some unknown function \(f\)

- Find:
  
  A good approximation to \(f\)
Many Other ML Paradigms

- **Supervised:**
  - Given “labeled data” \( \{x, y\} \), learn \( f(x) = y \)

- **Unsupervised:**
  - Given only “unlabeled data” \( \{x\} \), learn \( f(x) \)

- **Semi-supervised:**
  - Given some labeled \( \{x, y\} \) and some unlabeled data \( \{x\} \), learn \( f(x) = y \)

- **Active learning:**
  - When we predict \( f(x) = y \), we then receive true \( y^* \)

- **Transfer learning:**
  - Learn \( f(x) \) so that it works well on new domain \( f(z) \)
Supervised Learning

- Would like to do **prediction**: estimate a function $f(x)$ so that $y = f(x)$

- Where $y$ can be:
  - **Continuous / Real number**: Regression
  - **Categorical**: Classification
  - **Complex object**:
    - Ranking of items, Parse tree, etc.

- **Data is labeled**:
  - Have many pairs $\{(x, y)\}$
    - $x$ ... vector of binary, categorical, real valued features
    - $y$ ... class, or a real number
Supervised Learning

- **Task**: Given data \((X, Y)\) build a model \(f()\) to predict \(Y'\) based on \(X'\)
- **Strategy**: Estimate \(y = f(x)\) on \((X, Y)\)
  - Hope that the same \(f(x)\) also works to predict unknown \(Y'\)
  - The “hope” is called **generalization**
    - **Overfitting**: If \(f(x)\) predicts well \(Y\) but is unable to predict \(Y'\)
  - We want to build a model that **generalizes** well to unseen data
Why Large-Scale ML?

- **Brawn or Brains?**
  - In 2001, Microsoft researchers ran a test to evaluate 4 of different approaches to ML-based language translation.

- **Findings:**
  - Size of the dataset used to train the model mattered more than the model itself.
  - As the dataset grew large, performance difference between the models became small.

Why Large-Scale ML?

- The Unreasonable Effectiveness of Big Data
  - In 2017, Google revisited the same type of experiment with the latest Deep Learning models in computer vision

- Findings:
  - Performance increases logarithmically based on volume of training data
  - Complexity of modern ML models (i.e., deep neural nets) allows for even further performance gains

- Large datasets + large ML models => amazing results!!

Decision Trees
Decision Tree Learning

- Given one attribute (e.g., lifespan), try to predict the value of new people’s lifespans by means of some of the other available attribute
- **Input attributes:**
  - $d$ features/attributes: $x^{(1)}, x^{(2)}, \ldots, x^{(d)}$
  - Each $x^{(j)}$ has **domain** $O_j$
    - Categorical: $O_j = \{ \text{brown, blue, gray} \}$
    - Numerical: $H_j = (0, 10)$
  - $Y$ is output variable with domain $O_Y$:
    - Categorical: Classification, Numerical: Regression
- **Data D:**
  - $n$ examples $(x_i, y_i)$ where $x_i$ is a $d$-dim feature vector, $y_i \in O_Y$ is output variable
- **Task:**
  - Given an input data vector $x$ predict output label $y$
A Decision Tree is a tree-structured plan of a set of attributes to test in order to predict the output.
Decision Trees

- **Decision trees:**
  - Split the data at each internal node
  - Each leaf node makes a prediction

- **Lecture today:**
  - Binary splits: $X^{(j)} < v$
  - Numerical attributes
  - Regression
How to make predictions?

- **Input:** Example $x_i$
- **Output:** Predicted $\hat{y}_i$

- “Drop” $x_i$ down the tree until it hits a leaf node
- Predict the value stored in the leaf that $x_i$ hits
Decision Trees: feature space

Alternative view:
How to construct a tree?

- Training dataset $D^*$, $|D^*| = 100$ examples

![Diagram of a tree with nodes labeled A, B, C, D, E, F, G, H, I, and edges labeled with $X$ and $v$]
How to construct a tree?

- Imagine we are currently at some node $G$
  - Let $D_G$ be the data that reaches $G$
- There is a decision we have to make: Do we continue building the tree?
  - If yes, which variable and which value do we use for a split?
    - Continue building the tree recursively
  - If not, how do we make a prediction?
    - We need to build a “predictor node”
3 steps in constructing a tree

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>BuildSubtree</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Require:</strong> Node $n$, Data $D \subseteq D^*$</td>
<td></td>
</tr>
<tr>
<td>1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$</td>
<td>(1)</td>
</tr>
<tr>
<td>2: if StoppingCriteria($D_L$) then</td>
<td>(2)</td>
</tr>
<tr>
<td>3: $n \rightarrow \text{left_prediction} = \text{FindPrediction}(D_L)$</td>
<td>(3)</td>
</tr>
<tr>
<td>4: else</td>
<td></td>
</tr>
<tr>
<td>5: BuildSubtree $(n \rightarrow \text{left}, D_L)$</td>
<td></td>
</tr>
<tr>
<td>6: if StoppingCriteria($D_R$) then</td>
<td></td>
</tr>
<tr>
<td>7: $n \rightarrow \text{right_prediction} = \text{FindPrediction}(D_R)$</td>
<td></td>
</tr>
<tr>
<td>8: else</td>
<td></td>
</tr>
<tr>
<td>9: BuildSubtree $(n \rightarrow \text{right}, D_R)$</td>
<td></td>
</tr>
</tbody>
</table>

- Requires at least a single pass over the data!
How to construct a tree?

(1) How to split? Pick attribute & value that optimizes some criterion

- **Regression: Purity**
  - Find split \((X^{(i)}, v)\) that creates \(D, D_L, D_R\): parent, left, right child datasets and maximizes:

  \[
  |D| \cdot Var(D) - \left( |D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R) \right)
  \]

  - \(Var(D) = \frac{1}{n} \sum_{i \in D} (y_i - \bar{y})^2 \) ... variance of \(y_i\) in \(D\)
How to construct a tree?

(1) How to split? Pick attribute & value that optimizes some criterion

- Classification: Information Gain
  - Measures how much a given attribute $X$ tells us about the class $Y$
  - $IG(Y \mid X)$: We must transmit $Y$ over a binary link. How many bits on average would it save us if both ends of the line knew $X$?
Why Information Gain? Entropy

**Entropy:** What’s the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from $X$’s distribution?

The entropy of $X$: $H(X) = -\sum_{j=1}^{m} p(X_j) \log p(X_j)$

- “High Entropy”: $X$ is from a uniform (flat) distribution
  - A histogram of the frequency distribution of values of $X$ is flat
- “Low Entropy”: $X$ is from a varied (peaks/valleys) distrib.
  - A histogram of the frequency distribution of values of $X$ would have many lows and one or two peaks
Why Information Gain? Entropy

- Suppose I want to predict $Y$ and I have input $X$
  - $X = \text{College Major}$
  - $Y = \text{Likes Movie “Casablanca”}$

### From this data we estimate

- $P(Y = Yes) = 0.5$
- $P(X = Math \& Y = No) = 0.25$
- $P(X = Math) = 0.5$
- $P(Y = Yes \mid X = History) = 0$

### Note:

- $H(Y) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$
- $H(X) = 1.5$
Why Information Gain? Entropy

- Suppose I want to predict $Y$ and I have input $X$
  - $X = \text{College Major}$
  - $Y = \text{Likes “Casablanca”}$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
</tbody>
</table>

- **Def: Specific Conditional Entropy**
  - $H(Y \mid X = v) = \text{The entropy of } Y \text{ among only those records in which } X \text{ has value } v$

- **Example:**
  - $H(Y \mid X = \text{Math}) = 1$
  - $H(Y \mid X = \text{History}) = 0$
  - $H(Y \mid X = \text{CS}) = 0$
Why Information Gain?

- Suppose I want to predict $Y$ and I have input $X$
  - $X = \text{College Major}$
  - $Y = \text{Likes “Casablanca”}$

- **Def:** Conditional Entropy
  - $H(Y \mid X) = \text{The average specific conditional entropy of } Y$
  - = if you choose a record at random what will be the conditional entropy of $Y$, conditioned on that row’s value of $X$
  - = Expected number of bits to transmit $Y$ if both sides will know the value of $X$
  - $= \sum_j P(X = v_j) H(Y \mid X = v_j)$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>Math</td>
<td>No</td>
</tr>
<tr>
<td>CS</td>
<td>Yes</td>
</tr>
<tr>
<td>Math</td>
<td>Yes</td>
</tr>
<tr>
<td>History</td>
<td>No</td>
</tr>
</tbody>
</table>
Why Information Gain?

- Suppose I want to predict $Y$ and I have input $X$

  - $H(Y \mid X) = \text{The average specific conditional entropy of } Y$

    $= \sum_j P(X = v_j) H(Y \mid X = v_j)$

- Example:

<table>
<thead>
<tr>
<th>$v_j$</th>
<th>$P(X=v_j)$</th>
<th>$H(Y \mid X=v_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>History</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>CS</td>
<td>0.25</td>
<td>0</td>
</tr>
</tbody>
</table>

- **So:** $H(Y \mid X) = 0.5 \times 1 + 0.25 \times 0 + 0.25 \times 0 = 0.5$
Why Information Gain?

- Suppose I want to predict $Y$ and I have input $X$

- Def: Information Gain
  \[ IG(Y|X) = \text{I must transmit } Y. \text{ How many bits on average would it save me if both ends of the line knew } X? \]
  \[ IG(Y|X) = H(Y) - H(Y|X) \]

- Example:
  \[ H(Y) = 1 \]
  \[ H(Y|X) = 0.5 \]
  \[ \text{Thus } IG(Y|X) = 1 - 0.5 = 0.5 \]
What is Information Gain used for?

- Suppose you are trying to predict whether someone is going to live past 80 years
- From historical data you might find:
  - \( IG(\text{LongLife} \mid \text{HairColor}) = 0.01 \)
  - \( IG(\text{LongLife} \mid \text{Smoker}) = 0.4 \)
  - \( IG(\text{LongLife} \mid \text{Gender}) = 0.25 \)
  - \( IG(\text{LongLife} \mid \text{LastDigitOfSSN}) = 0.00001 \)
- IG tells us how much information about \( Y \) is contained in \( X \)
  - So attribute \( X \) that has high \( IG(Y \mid X) \) is a good split!
3 steps in constructing a tree

Algorithm 1: BuildSubtree

Require: Node $n$, Data $D \subseteq D^*$
1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$  \hspace{1cm} (1)
2: if StoppingCriteria($D_L$) then \hspace{1cm} (2)
3: $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$ \hspace{1cm} (3)
4: else
5: \hspace{0.5cm} BuildSubtree $(n \rightarrow \text{left}, D_L)$
6: if StoppingCriteria($D_R$) then
7: \hspace{0.5cm} $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8: else
9: \hspace{0.5cm} BuildSubtree $(n \rightarrow \text{right}, D_R)$
When to stop?

(2) When to stop?
- Many different heuristic options to avoid overfitting
- **Two ideas:**
  - (1) **When the leaf is "pure"**
    - The target variable does not vary too much: \( \text{Var}(y) < \epsilon \)
  - (2) **When # of examples in the leaf is too small**
    - For example, \( |D| \leq 100 \)
  - (3) **Stop at a fixed depth**
    - For example, max depth = 4.
(3) How to predict?

- Many options
  - Regression:
    - Typically: Predict average $y_i$ of the examples in the leaf
    - Build a linear regression model on the examples in the leaf
  - Classification:
    - Predict most common $y_i$ of the examples in the leaf
Building Decision Trees Using MapReduce
Problem: Building a tree

- Given a large dataset with hundreds of attributes
- Build a decision tree!

General considerations:
- **Tree is small** (can keep it memory):
  - Shallow (~10 levels)
- **Dataset too large to keep in memory (Petabytes)**
- **Dataset too big to scan over on a single machine**
- **MapReduce to the rescue!**

---

Algorithm 1: BuildSubTree

Require: Node $n$, Data $D \subseteq D^*$

1. $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$
2. if StoppingCriteria($D_L$) then
3.   $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$
4. else
5.   BuildSubTree($n \rightarrow \text{left}, D_L$)
6. if StoppingCriteria($D_R$) then
7.   $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8. else
9.   BuildSubTree($n \rightarrow \text{right}, D_R$)
Parallel Learner for Assembling Numerous Ensemble Trees [Panda et al., VLDB ‘09]

- A sequence of MapReduce jobs that builds a decision tree
- Spark MLlib Decision Trees are based on PLANET

Setting:
- Hundreds of numerical (discrete & continuous, but not categorical) attributes
- Target variable is numerical: Regression
- Splits are binary: $X^{(i)} < v$
- Decision tree is small enough for each Mapper to keep it in memory
- Data too large to keep in memory
PLANEt Architecture

Input data → MapReduce → Master

Model → Master → Intermediate results

Attribute metadata → Master → Intermediate results

MapReduce: Given a set of split candidates compute their quality

Master keeps track of the model and decides how to grow the tree
The tree will be built in levels

- One level at a time:

Steps:
- 1) Master decides candidate splits \((n, X^{(j)}, v)\)
- 2) MapReduce computes quality of those splits
- 3) Master then grows the tree for a level
- 4) Goto (1)
Hard part: Computing “quality” of a split
1) Master tells the Mappers which splits \((n, X^{(j)}, v)\) to consider
2) Each Mapper gets a subset of data and computes partial statistics for a given split
3) Reducers collect partial statistics and output the final quality for a given split \((n, X^{(j)}, v)\)
4) Master makes final decision where to split
PLANET Overview

- **We build the tree level by level**
  - One MapReduce step builds **one level of the tree**

- **Mapper**
  - Considers a number of candidate splits \((\text{node, attribute, value})\) on its subset of the data
  - For each split it stores **partial statistics**
  - Partial split-statistics is sent to **Reducers**

- **Reducer**
  - Collects all partial statistics and determines best split

- **Master** grows the tree for one level
PLANT Overview

- **Mapper** loads the **DT model** and info about which **attribute splits** (split is a tuple `<NodeID, Attribute, Value>`) to consider
  - Each mapper sees a subset of the data \( D^* \)
  - Mapper “drops”/classifies each datapoint \( d \) using the tree to find the leaf node \( L \) where \( d \) lands
  - For each leaf node \( L \) mapper keeps statistics about
    - (1) the data reaching \( L \)
    - (2) the data in left/right subtree under some split \( S \)
- **Reducer** aggregates the statistics (1), (2) and determines the best split for each tree node
PLANET: Components

- Master
  - Monitors everything (runs multiple MapReduce jobs)

- Three types of MapReduce jobs:
  - 1. MapReduce **Initialization** (run once first)
     - For each attribute identify values to be considered for splits
  - 2. MapReduce **FindBestSplit** (run multiple times)
     - MapReduce job to find best split (when there is too much data to fit in memory)
  - 3. MapReduce **InMemoryBuild** (run once last)
     - Similar to **BuildSubTree** (but for small data)
     - Grows an entire sub-tree once the data fits in memory

- Model file
  - A file describing the state of the model
PLANET: Components

(1) Master Node
(2) MapReduce Initialization (run once first)
(3) MapReduce FindBestSplit (run multiple times)
(4) MapReduce InMemoryBuild (run once last)
PLANET: Master

- **Master controls the entire process**
- **Determines the state of the tree and grows it:**
  - (1) Decides if nodes should be split
  - (2) If there is little data entering a tree node, Master runs an *InMemoryBuild* MapReduce job to grow the entire subtree below that node
  - (3) For larger nodes, Master launches MapReduce *FindBestSplit* to evaluate candidates for best split
    - Master also collects results from *FindBestSplit* and chooses the best split for a node
  - (4) Updates the model
PLANET: Components

1. Master Node
2. MapReduce **Initialization** (run once first)
3. MapReduce **FindBestSplit** (run multiple times)
4. MapReduce **InMemoryBuild** (run once last)
Initialization: Attribute metadata

- **Initialization job:** Identifies all the attribute values which need to be considered for splits
  - Initialization process generates “attribute metadata” to be loaded in memory by other tasks

- **Main question:**
  Which splits to even consider?

- **A split is defined by a triple:**
  (node $n$, attribute $X^{(j)}$, value $v$)
Initialization: Attribute metadata

- **Which splits to even consider?**
  - For small data we can sort the values along a particular feature and consider every possible split
  - But data values may not be uniformly populated so many splits may not really make a difference

X(j): 1.2 1.3 1.4 1.6 2.1 7.2 8.1 9.8 10.1 10.2 10.3 10.4 11.5 11.7 12.8 12.9

- **Idea:** Consider a limited number of splits such that splits “move” about the same amount of data
Initialization: Attribute metadata

- Splits for numerical attributes:
  - For attribute $X^{(j)}$ we would like to consider every possible value $v \in O_j$
  - Compute an approx. equi-depth histogram on $D^*$
    - Idea: Select buckets such that counts per bucket are equal
  - Use boundary points of histogram as splits
**Goal:** Equal number of elements per bucket ($B$ buckets total)

- Construct by first **sorting** and then taking $B-1$ equally-spaced splits

**Faster construction:**
- Sample & take equally-spaced splits in the sample
  - Nearly equal buckets
PLANET: Components

(1) Master Node
(2) MapReduce Initialization (run once first)
(3) MapReduce FindBestSplit (run multiple times)
(4) MapReduce InMemoryBuild (run once last)
**Goal:** For a particular split node $n$ find attribute $X^{(j)}$ and value $v$ that maximize Purity:

\[ |D| \cdot Var(D) - \left( |D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R) \right) \]

- $D$ ... training data $(x_i, y_i)$ reaching the node $n$
- $D_L$ ... training data $x_i$, where $x_i^{(j)} < v$
- $D_R$ ... training data $x_i$, where $x_i^{(j)} \geq v$

\[ Var(D) = \frac{1}{n} \sum_{i \in D} y_i^2 - \left( \frac{1}{n} \sum_{i \in D} y_i \right)^2 \]
To compute Purity we need

\[ \text{Var}(D) = \frac{1}{n} \sum_{i \in D} y_i^2 - \left( \frac{1}{n} \sum_{i \in D} y_i \right)^2 \]

Important observation: Variance can be computed from sufficient statistics: \( N, S=\Sigma y_i, Q=\Sigma y_i^2 \)

- Each Mapper \( m \) processes subset of data \( D_m \), and computes \( N_m, S_m, Q_m \) for its own \( D_m \)
- Reducer combines the statistics and computes global variance and then Purity:

\[ \text{Var}(D) = \frac{1}{\sum_m N_m} \sum_m Q_m - \left( \frac{1}{\sum_m N_m} \sum_m S_m \right)^2 \]
FindBestSplit: Map

- **Mapper:**
  - Initialized by loading results of **Initialization task**
    - **Current model** (to find which node each datapoint \(x_i\) ends up)
    - **Attribute metadata** (all split points for each attribute)
    - Load the set of **candidate splits**: \{(node, attribute, value)\}
  - For each data record run the Map algorithm:
    - For each tree node store statistics of the data entering the node and at the end emit (to all reducers):
      - \(<\text{NodeID}, \{ S=\Sigma y, Q=\Sigma y^2, N=\Sigma 1 \} >\>
    - For each split store statistics and at the end emit:
      - \(<\text{SplitID}, \{ S, Q, N \} >\>
      - \(\text{SplitID} = (\text{node id}, \text{attribute } X^{(j)}, \text{split value } v)\)
FindBestSplit: Reducer

Reducer:

- (1) Load all the \(<\text{NodeID}, \text{List}\{S_m, Q_m, N_m\}>\) pairs and aggregate the per node statistics
- (2) For all the \(<\text{SplitID}, \text{List}\{S_m, Q_m, N_m\}>\) aggregate the statistics

\[
Var(D) = \frac{1}{\sum_m N_m} \sum_m Q_m - \left(\frac{1}{\sum_m N_m} \sum_m S_m\right)^2
\]

- For each NodeID, output the best split found
Overall system architecture

- **Master gives the mappers:**
  1. Tree
  2. Set of nodes
  3. Set of candidate splits

**Nodes:** F, G, H, I
**Split candidates:** (G, X(1), v(1)), (G, X(1), v(2)), (H, X(3), v(3)), (H, X(4), v(4))

Mappers output 2 types of key-value pairs:
- (NodeID: S,Q,N)
- (Split: S,Q,N)

For every (NodeID, Split) Reducer(s) compute the Purity and output the best split.
Overall system architecture

- **Example:** Need to split nodes F, G, H, I
- **Map and Reduce:**
  - **FindBestSplit::Map** (each mapper)
    - Load the current model \( M \)
    - Drop every example \( x_i \) down the tree
    - If it hits \( F/G/H/I \), update in-memory hash tables:
      - For each node: \( T_n: \text{(Node)} \rightarrow \{S, Q, N\} \)
      - For each \( \text{(Split, Node)}: T_{n,j,s}: \text{(Node, Attribute, SplitValue)} \rightarrow \{S, Q, N\} \)
  - **Map::Finalize:** output the key-value pairs from above hashtables
  - **FindBestSplit::Reduce** (each reducer)
    - Collect:
      - \( T_1: <\text{Node, List}\{S, Q, N\}> \rightarrow <\text{Node, \{\Sigma S, \Sigma Q, \Sigma N\}>} \)
      - \( T_2: <(\text{Node, Attr., Val}), \text{List}\{S, Q, N\}> \rightarrow <(\text{Node, Attr., Val}), \{\Sigma S, \Sigma Q, \Sigma N\}> \)
    - Compute Purity for each node using \( T_1, T_2 \)
    - Return **best split** to Master (which then decides on globally best split)
Back to the Master

- Collects outputs from FindBestSplit reducers <Split.NodeID, Attribute, Value, Purity>

- For each node decides the best split
  - If data in $D_L/D_R$ is small enough, later run a MapReduce job InMemoryBuild on the node
  - Else run MapReduce FindBestSplit job for both nodes
Decision Trees: Conclusion
Decision Trees

- **Characteristics**
  - Classification & Regression
    - Multiple (~10) classes
  - Real valued and categorical features
  - Few (hundreds) of features
  - Usually dense features
  - Complicated decision boundaries
    - Early stopping to avoid overfitting!
- **Example applications**
  - User profile classification
  - Landing page bounce prediction
Decision Trees

- Decision trees are the single most popular data mining tool:
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap
  - It’s possible to mitigate overfitting (i.e., with ensemble methods or early stopping)
  - They do classification as well as regression!
Learning Ensembles

- Learn multiple trees and combine their predictions
  - Gives better performance in practice
- Bagging:
  - Learns multiple trees over independent samples of the training data
    - For a dataset $D$ on $n$ data points: Create dataset $D'$ of $n$ points but sample from $D$ with replacement
      - Dataset $D'$ will include duplicate data points
  - Predictions from each tree are averaged to compute the final model prediction
Bagging Decision Trees

Random Forest

Instance

Tree-1

Class-A

Tree-2

Class-B

Majority-Voting

Final-Class

Tree-n

Class-B

...
Improvement: Random Forests

- Train a **Bagged Decision Tree**
- But use a modified tree learning algorithm that selects (at each candidate split) **a random subset of the features**
  - If we have $d$ features, consider $\sqrt{d}$ random features

- **This is called: Feature bagging**
  - **Benefit:** Breaks correlation between trees
    - Otherwise, if one feature is very strong predictor, then every tree will select it, causing trees to be correlated.

- Random Forests achieve state-of-the-art results in many classification problems!