More on Supervised Learning

Amir H. Payberah
payberah@kth.se
21/11/2018

Let's Start with an Example

Buying Computer Example (1/3)

- Given the dataset of `m` people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
⋮	⋮	⋮	⋮	⋮	⋮

Buying Computer Example (1/3)

- Given the dataset of m people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
⋮	⋮	⋮	⋮	⋮	⋮

- Predict if a new person buys a computer?

Buying Computer Example (1/3)

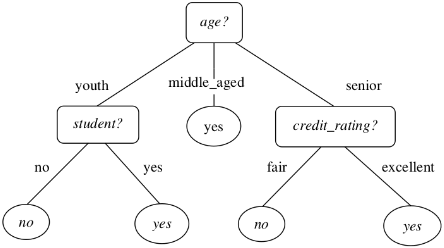
- Given the dataset of $\mathtt{m}$ people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
⋮	⋮	⋮	⋮	⋮	⋮

- Predict if a new person buys a computer?
- Given an instance $\mathbf{x}^{(\mathtt{i})}$, e.g., $\mathtt{x}_1^{(\mathtt{i})} = \mathtt{senior}$, $\mathtt{x}_2^{(\mathtt{i})} = \mathtt{medium}$, $\mathtt{x}_3^{(\mathtt{i})} = \mathtt{no}$, and $\mathtt{x}_4^{(\mathtt{i})} = \mathtt{fair}$, then $\mathtt{y}^{(\mathtt{i})} = ?$

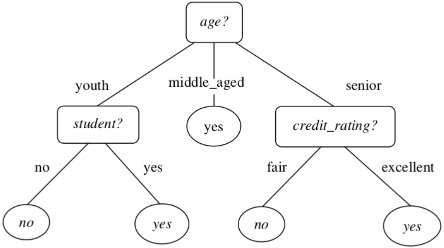
Buying Computer Example (2/3)

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
⋮	⋮	⋮	⋮	⋮	⋮



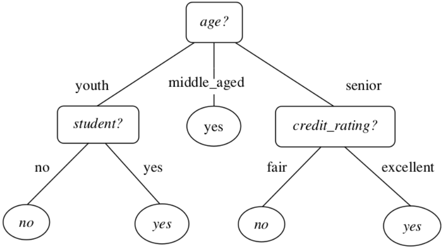
Buying Computer Example (3/3)

- Given an input instance $\mathbf{x}^{(i)}$, for which the class label $y^{(i)}$ is unknown.



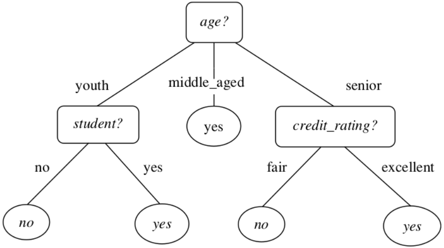
Buying Computer Example (3/3)

- Given an input instance $\mathbf{x}^{(i)}$, for which the class label $y^{(i)}$ is unknown.
- The attribute values of the input (e.g., age or `income`) are tested.



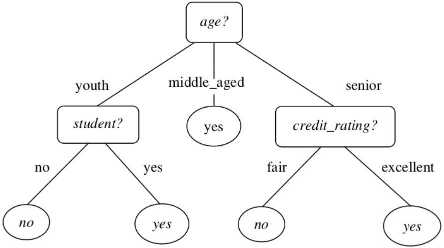
Buying Computer Example (3/3)

- Given an input instance $\mathbf{x}^{(i)}$, for which the class label $y^{(i)}$ is unknown.
- The attribute values of the input (e.g., age or income) are tested.
- A path is traced from the root to a leaf node, which holds the class prediction for that input.



Buying Computer Example (3/3)

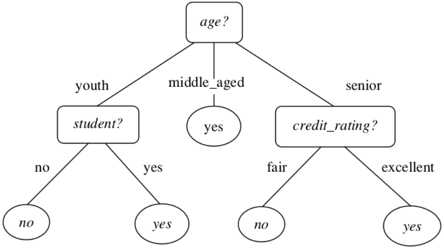
- Given an input instance $\mathbf{x}^{(\mathtt{i})}$, for which the class label $\mathtt{y}^{(\mathtt{i})}$ is unknown.
- The attribute values of the input (e.g., age or `income`) are tested.
- A path is traced from the root to a leaf node, which holds the class prediction for that input.
- E.g., input $\mathbf{x}^{(\mathtt{i})}$ with $\mathtt{x}_1^{(\mathtt{i})} = \mathtt{senior}$, $\mathtt{x}_2^{(\mathtt{i})} = \mathtt{medium}$, $\mathtt{x}_3^{(\mathtt{i})} = \mathtt{no}$, and $\mathtt{x}_4^{(\mathtt{i})} = \mathtt{fair}$.



Decision Tree

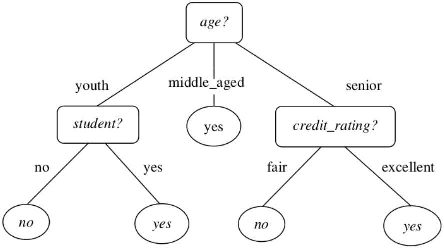
Decision Tree

- A decision tree is a flowchart-like tree structure.



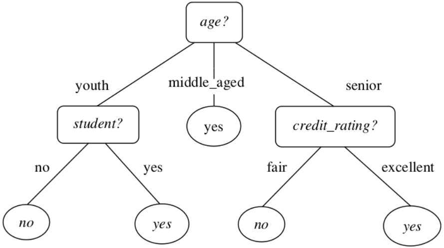
Decision Tree

- A decision tree is a flowchart-like tree structure.
  - The topmost node: represents the root



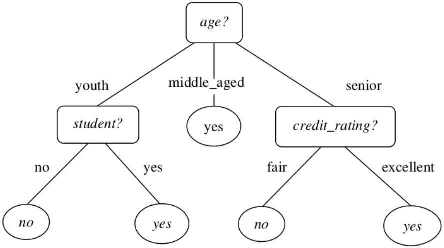
Decision Tree

- A decision tree is a flowchart-like tree structure.
  - The topmost node: represents the root
  - Each branch: represents an outcome of the test



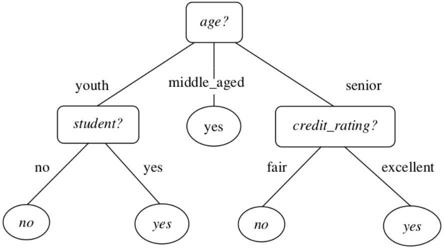
Decision Tree

- A decision tree is a flowchart-like tree structure.
  - The topmost node: represents the root
  - Each branch: represents an outcome of the test
  - Each internal node: denotes a test on an attribute



Decision Tree

- A decision tree is a flowchart-like tree structure.
  - The topmost node: represents the root
  - Each branch: represents an outcome of the test
  - Each internal node: denotes a test on an attribute
  - Each leaf: holds a class label



Training Algorithm (1/2)

- Decision trees are constructed in a top-down recursive divide-and-conquer manner.

Training Algorithm (1/2)

- Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- The algorithm is called with the following parameters.

Training Algorithm (1/2)

- Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- The algorithm is called with the following parameters.
  - Data partition $\mathtt{D}$: initially the complete set of training data and labels $\mathtt{D} = (\mathbf{X}, \mathbf{y})$.

Training Algorithm (1/2)

- Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- The algorithm is called with the following parameters.
  - Data partition $\mathtt{D}$: initially the complete set of training data and labels $\mathtt{D} = (\mathbf{X}, \mathbf{y})$.
  - Feature list: list of features $\{\mathbf{x}_{\mathtt{1}}^{(\mathtt{i})}, \cdots, \mathbf{x}_{\mathtt{n}}^{(\mathtt{i})}\}$ of each data instance $\mathbf{x}^{(\mathtt{i})}$.

Training Algorithm (1/2)

- Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- The algorithm is called with the following parameters.
  - Data partition $\mathtt{D}$: initially the complete set of training data and labels $\mathtt{D} = (\mathbf{X}, \mathbf{y})$.
  - Feature list: list of features $\{\mathbf{x}_1^{(\mathtt{i})}, \cdots, \mathbf{x}_\mathtt{n}^{(\mathtt{i})}\}$ of each data instance $\mathbf{x}^{(\mathtt{i})}$.
  - Feature selection method: determines the splitting criterion.

Training Algorithm (2/2)

- 1. The tree starts as a single node, `N`, representing the training data instances `D`.

Training Algorithm (2/2)

- 1. The tree starts as a single node, N, representing the training data instances D.
- 2. If all instances **x** in D are all of the same class, then node N becomes a leaf.

Training Algorithm (2/2)

- 1. The tree starts as a single node, N, representing the training data instances D.
- 2. If all instances **x** in D are all of the same class, then node N becomes a leaf.
- 3. The algorithm calls feature selection method to determine the splitting criterion.

Training Algorithm (2/2)

- 1. The tree starts as a single node, $\mathtt{N}$, representing the training data instances $\mathtt{D}$.
- 2. If all instances $\mathbf{x}$ in $\mathtt{D}$ are all of the same class, then node $\mathtt{N}$ becomes a leaf.
- 3. The algorithm calls feature selection method to determine the splitting criterion.
  - Indicates (i) the splitting feature $\mathtt{x}_\mathtt{k}$, and (ii) a split-point or a splitting subset.
  - The instances in $\mathtt{D}$ are partitioned accordingly.

Training Algorithm (2/2)

- 1. The tree starts as a single node, $\mathtt{N}$, representing the training data instances $\mathtt{D}$.
- 2. If all instances $\mathbf{x}$ in $\mathtt{D}$ are all of the same class, then node $\mathtt{N}$ becomes a leaf.
- 3. The algorithm calls feature selection method to determine the splitting criterion.
  - Indicates (i) the splitting feature $\mathtt{x}_{\mathtt{k}}$, and (ii) a split-point or a splitting subset.
  - The instances in $\mathtt{D}$ are partitioned accordingly.
- 4. The algorithm repeats the same process recursively to form a decision tree.

Training Algorithm - Termination Conditions

- The training algorithm stops only when any one of the following conditions is true.

Training Algorithm - Termination Conditions

- The training algorithm stops only when any one of the following conditions is true.
- 1. All the instances in partition `D` at a node `N` belong to the same class.
  - It is labeled with that class.

Training Algorithm - Termination Conditions

- The training algorithm stops only when any one of the following conditions is true.
- 1. All the instances in partition `D` at a node `N` belong to the same class.
    - It is labeled with that class.
- 2. No remaining features on which the instances may be further partitioned.

Training Algorithm - Termination Conditions

- The training algorithm stops only when any one of the following conditions is true.
- 1. All the instances in partition `D` at a node `N` belong to the same class.
  - It is labeled with that class.
- 2. No remaining features on which the instances may be further partitioned.
- 3. There are no instances for a given branch, that is, a partition D_j is empty.

Training Algorithm - Termination Conditions

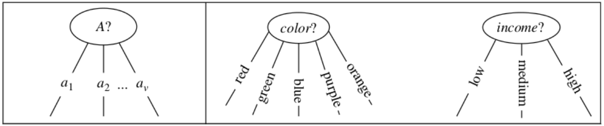
- The training algorithm stops only when any one of the following conditions is true.
- 1. All the instances in partition D at a node N belong to the same class.
  - It is labeled with that class.
- 2. No remaining features on which the instances may be further partitioned.
- 3. There are no instances for a given branch, that is, a partition D_j is empty.
- In conditions 2 and 3:
  - Convert node N into a leaf.
  - Label it either with the most common class in D.
  - Or, the class distribution of the node tuples may be stored.

Training Algorithm - Partitioning Instances (1/3)

- Assume A is the splitting feature
- Three possibilities to partition instances in D based on the feature A.
- 1. A is discrete-valued

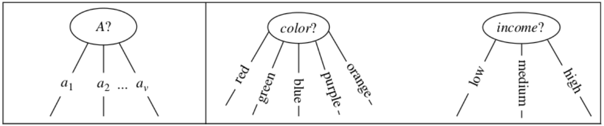
Training Algorithm - Partitioning Instances (1/3)

- Assume A is the splitting feature
- Three possibilities to partition instances in D based on the feature A.
- 1. A is discrete-valued
  - Assume A has v distinct values $\{a_1, a_2, \cdots, a_v\}$



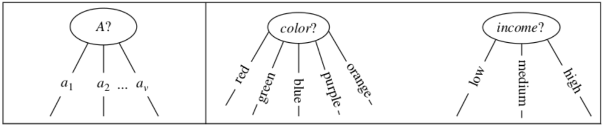
Training Algorithm - Partitioning Instances (1/3)

- Assume A is the splitting feature
- Three possibilities to partition instances in D based on the feature A.
- 1. A is discrete-valued
  - Assume A has v distinct values $\{a_1, a_2, \cdots, a_v\}$
  - A branch is created for each known value a_j of A and labeled with that value.



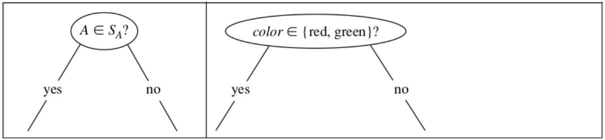
Training Algorithm - Partitioning Instances (1/3)

- Assume $\mathtt{A}$ is the splitting feature
- Three possibilities to partition instances in $\mathtt{D}$ based on the feature $\mathtt{A}$.
- 1. $\mathtt{A}$ is discrete-valued
  - Assume $\mathtt{A}$ has $\mathtt{v}$ distinct values $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$
  - A branch is created for each known value $\mathtt{a}_\mathtt{j}$ of $\mathtt{A}$ and labeled with that value.
  - Partition $\mathtt{D}_\mathtt{j}$ is the subset of tuples in $\mathtt{D}$ having value $\mathtt{a}_\mathtt{j}$ of $\mathtt{A}$.



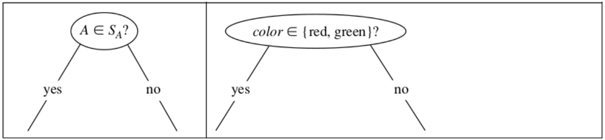
Training Algorithm - Partitioning Instances (2/3)

- 2. `A` is discrete-valued



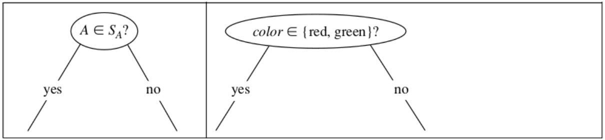
Training Algorithm - Partitioning Instances (2/3)

- 2. `A` is discrete-valued
  - A binary tree must be produced.



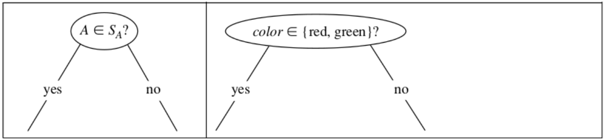
Training Algorithm - Partitioning Instances (2/3)

- 2. $\mathtt{A}$ is discrete-valued
  - A binary tree must be produced.
  - The test at node $\mathtt{N}$ is of the form $\mathtt{A} \in \mathtt{S_A}$?, where $\mathtt{S_A}$ is the splitting subset for $\mathtt{A}$.



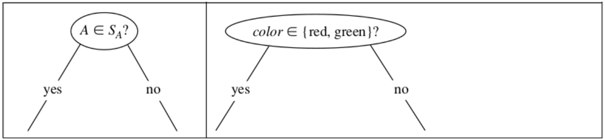
Training Algorithm - Partitioning Instances (2/3)

- 2. `A` is discrete-valued
  - A binary tree must be produced.
  - The test at node `N` is of the form $\mathtt{A} \in \mathtt{S_A}$?, where $\mathtt{S_A}$ is the splitting subset for `A`.
  - The left branch out of `N` corresponds to the instances in `D` that satisfy the test.



Training Algorithm - Partitioning Instances (2/3)

- 2. A is discrete-valued
  - A binary tree must be produced.
  - The test at node N is of the form $A \in S_A$?, where S_A is the splitting subset for A.
  - The left branch out of N corresponds to the instances in D that satisfy the test.
  - The right branch out of N corresponds to the instances in D that do not satisfy the test.



Training Algorithm - Partitioning Instances (3/3)

Training Algorithm - Partitioning Instances (3/3)

- 3. `A` is continuous-valued

Training Algorithm - Partitioning Instances (3/3)

- 3. $\mathtt{A}$ is continuous-valued
  - A test at node $\mathtt{N}$ has two possible outcomes: corresponds to $\mathtt{A} \leq \mathtt{s}$ or $\mathtt{A} > \mathtt{s}$, with $\mathtt{s}$ as the split point.

Training Algorithm - Partitioning Instances (3/3)

- 3. A is continuous-valued
  - A test at node N has two possible outcomes: corresponds to $A \leq s$ or $A > s$, with s as the split point.
  - The instances are partitioned such that D_1 holds the instances in D for which $A \leq s$, while D_2 holds the rest.

Training Algorithm - Partitioning Instances (3/3)

- 3. A is continuous-valued
  - A test at node N has two possible outcomes: corresponds to $A \leq s$ or $A > s$, with s as the split point.
  - The instances are partitioned such that D_1 holds the instances in D for which $A \leq s$, while D_2 holds the rest.
  - Two branches are labeled according to the previous outcomes.

Training Algorithm - Feature Selection Measures (1/2)

- Feature selection measure: how to split instances at a node `N`.

Training Algorithm - Feature Selection Measures (1/2)

- Feature selection measure: how to split instances at a node `N`.
- Pure partiton: if all instances in a partition belong to the same class.

Training Algorithm - Feature Selection Measures (1/2)

- Feature selection measure: how to split instances at a node `N`.
- Pure partiton: if all instances in a partition belong to the same class.
- The best splitting criterion is the one that most closely results in a pure scenario.

Training Algorithm - Feature Selection Measures (2/2)

- It provides a ranking for each feature describing the given training instances.

Training Algorithm - Feature Selection Measures (2/2)

- It provides a ranking for each feature describing the given training instances.
- The feature having the best score for the measure is chosen as the splitting feature for the given instances.

Training Algorithm - Feature Selection Measures (2/2)

- It provides a ranking for each feature describing the given training instances.
- The feature having the best score for the measure is chosen as the splitting feature for the given instances.
- Two popular feature selection measures are:
  - Information gain (ID3 and C4.5)
  - Gini index (CART)

Information Gain (Entropy)

ID3 (1/8)

- ID3 (Iterative Dichotomiser 3) uses information gain as its feature selection measure.

ID3 (1/8)

- ID3 (Iterative Dichotomiser 3) uses information gain as its feature selection measure.
- The feature with the highest information gain is chosen as the splitting feature for node `N`.

ID3 (1/8)

- ID3 (Iterative Dichotomiser 3) uses information gain as its feature selection measure.
- The feature with the highest information gain is chosen as the splitting feature for node `N`.
- The information gain is based on the decrease in entropy after a dataset is split on a feature.

ID3 (2/8)

- What's entropy?

ID3 (2/8)

- What's entropy?
- The average information needed to identify the class label of an instance in `D`.

ID3 (2/8)

- What's entropy?
- The average information needed to identify the class label of an instance in D.

$$\mathtt{entropy}(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- p_i is the probability that an instance in D belongs to class i, with m distinct classes.

ID3 (2/8)

- What's entropy?
- The average information needed to identify the class label of an instance in $\mathtt{D}$.

$$\mathtt{entropy(D)} = -\sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p_i} \log_2(\mathtt{p_i})$$

- $\mathtt{p_i}$ is the probability that an instance in $\mathtt{D}$ belongs to class $\mathtt{i}$, with $\mathtt{m}$ distinct classes.
- $\mathtt{D}$'s entropy is zero when it contains instances of only one class (pure partition).

ID3 (3/8)

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\mathtt{entropy}(\mathtt{D}) = -\sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p_i} \log_2(\mathtt{p_i})$$

ID3 (3/8)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\mathtt{entropy}(\mathtt{D}) = -\sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p_i} \log_2(\mathtt{p_i})$$

$$\text{label} = \mathtt{buys\_computer} \Rightarrow \mathtt{m} = 2$$

$$\mathtt{entropy}(\mathtt{D}) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.94$$

ID3 (4/8)

- Suppose we want to partition instances in $\mathtt{D}$ on some feature $\mathtt{A}$ with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$.

ID3 (4/8)

- Suppose we want to partition instances in $\mathtt{D}$ on some feature $\mathtt{A}$ with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$.
- $\mathtt{A}$ can split $\mathtt{D}$ into $\mathtt{v}$ partitions $\{\mathtt{D}_1, \mathtt{D}_2, \cdots, \mathtt{D}_\mathtt{v}\}$.

ID3 (4/8)

- Suppose we want to partition instances in $\mathtt{D}$ on some feature $\mathtt{A}$ with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$.
- $\mathtt{A}$ can split $\mathtt{D}$ into $\mathtt{v}$ partitions $\{\mathtt{D}_1, \mathtt{D}_2, \cdots, \mathtt{D}_\mathtt{v}\}$.
- The expected information required to classify an instance from $\mathtt{D}$ based on the partitioning by $\mathtt{A}$ is:

$$\mathtt{entropy}(\mathtt{A}, \mathtt{D}) = \sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D}_\mathtt{j}|}{|\mathtt{D}|} \mathtt{entropy}(\mathtt{D}_\mathtt{j})$$

ID3 (4/8)

- Suppose we want to partition instances in $\mathtt{D}$ on some feature $\mathtt{A}$ with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$.

- $\mathtt{A}$ can split $\mathtt{D}$ into $\mathtt{v}$ partitions $\{\mathtt{D}_1, \mathtt{D}_2, \cdots, \mathtt{D}_\mathtt{v}\}$.

- The expected information required to classify an instance from $\mathtt{D}$ based on the partitioning by $\mathtt{A}$ is:

$$\mathtt{entropy}(\mathtt{A}, \mathtt{D}) = \sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D}_\mathtt{j}|}{|\mathtt{D}|} \mathtt{entropy}(\mathtt{D}_\mathtt{j})$$

- $\frac{|\mathtt{D}_\mathtt{j}|}{\mathtt{D}}$ is the weight of the $\mathtt{j}$th partition.

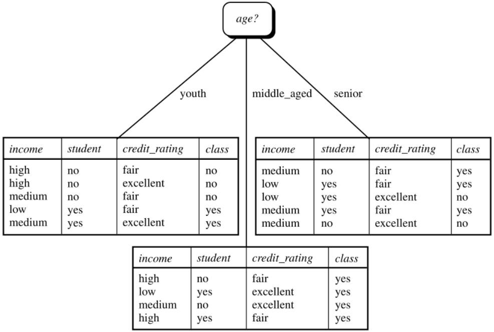
ID3 (4/8)

- Suppose we want to partition instances in $\mathtt{D}$ on some feature $\mathtt{A}$ with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$.
- $\mathtt{A}$ can split $\mathtt{D}$ into $\mathtt{v}$ partitions $\{\mathtt{D}_1, \mathtt{D}_2, \cdots, \mathtt{D}_\mathtt{v}\}$.
- The expected information required to classify an instance from $\mathtt{D}$ based on the partitioning by $\mathtt{A}$ is:

$$\mathtt{entropy}(\mathtt{A}, \mathtt{D}) = \sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D}_\mathtt{j}|}{|\mathtt{D}|} \mathtt{entropy}(\mathtt{D}_\mathtt{j})$$

- $\frac{|\mathtt{D}_\mathtt{j}|}{\mathtt{D}}$ is the weight of the $\mathtt{j}$th partition.
- The smaller the expected information required, the greater the purity of the partitions.

ID3 (5/8)



youth:

income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

middle_aged:

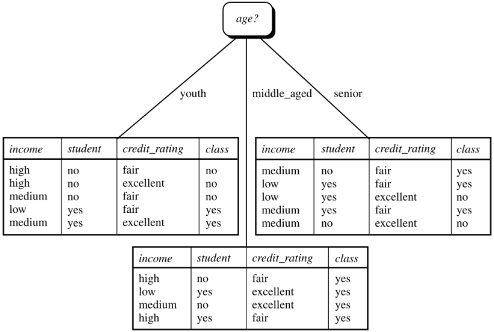
income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

senior:

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

$$\mathtt{entropy}(\mathtt{A},\mathtt{D}) = \sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D_j}|}{|\mathtt{D}|}\mathtt{entropy}(\mathtt{D_j})$$

ID3 (5/8)



youth:

income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

middle_aged:

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

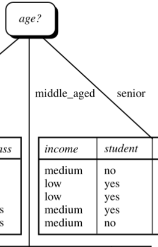
senior:

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

$$\mathtt{entropy}(\mathtt{A},\mathtt{D}) = \sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D_j}|}{|\mathtt{D}|}\mathtt{entropy}(\mathtt{D_j})$$

$$\mathtt{entropy}(\mathtt{age},\mathtt{D}) = \frac{5}{14}\mathtt{entropy}(\mathtt{D_{youth}}) + \frac{4}{14}\mathtt{entropy}(\mathtt{D_{middle\_aged}}) + \frac{5}{14}\mathtt{entropy}(\mathtt{D_{senior}})$$

ID3 (5/8)



income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

$$\mathtt{entropy}(\mathtt{A}, \mathtt{D}) = \sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D}_\mathtt{j}|}{|\mathtt{D}|} \mathtt{entropy}(\mathtt{D}_\mathtt{j})$$

$$\mathtt{entropy}(\mathtt{age}, \mathtt{D}) = \frac{5}{14}\mathtt{entropy}(\mathtt{D}_{\mathtt{youth}}) + \frac{4}{14}\mathtt{entropy}(\mathtt{D}_{\mathtt{middle\_aged}}) + \frac{5}{14}\mathtt{entropy}(\mathtt{D}_{\mathtt{senior}})$$

$$\mathtt{entropy}(\mathtt{age}, \mathtt{D}) = \frac{5}{14}(-\frac{2}{5}\log_2(\frac{2}{5}) - \frac{3}{5}\log_2(\frac{3}{5})) + \frac{4}{14}(-\frac{4}{4}\log_2(\frac{4}{4})) + \frac{5}{14}(-\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5})) = 0.694$$

ID3 (6/8)

- The information gain $\mathtt{Gain}(\mathtt{A}, \mathtt{D})$ is defined as:

$$\mathtt{Gain}(\mathtt{A}, \mathtt{D}) = \mathtt{entropy}(\mathtt{D}) - \mathtt{entropy}(\mathtt{A}, \mathtt{D})$$

ID3 (6/8)

- The information gain $\mathtt{Gain}(\mathtt{A}, \mathtt{D})$ is defined as:

$$\mathtt{Gain}(\mathtt{A}, \mathtt{D}) = \mathtt{entropy}(\mathtt{D}) - \mathtt{entropy}(\mathtt{A}, \mathtt{D})$$

- It shows how much would be gained by branching on $\mathtt{A}$.

ID3 (6/8)

- The information gain $\texttt{Gain}(\texttt{A}, \texttt{D})$ is defined as:

$$\texttt{Gain}(\texttt{A}, \texttt{D}) = \texttt{entropy}(\texttt{D}) - \texttt{entropy}(\texttt{A}, \texttt{D})$$

- It shows how much would be gained by branching on $\texttt{A}$.
- The feature $\texttt{A}$ with the highest $\texttt{Gain}(\texttt{A}, \texttt{D})$ is chosen as the splitting feature at node $\texttt{N}$.

ID3 (7/8)

- Now, we can compute the information gain $\texttt{Gain(A)}$ for the feature $\texttt{A = age}$.

$$\texttt{Gain}(\texttt{age}, \texttt{D}) = \texttt{entropy}(\texttt{D}) - \texttt{entropy}(\texttt{age}, \texttt{D}) = 0.940 - 0.694 = 0.246$$

ID3 (7/8)

- Now, we can compute the information gain $\texttt{Gain(A)}$ for the feature $\texttt{A = age}$.

$$\texttt{Gain}(\texttt{age}, \texttt{D}) = \texttt{entropy}(\texttt{D}) - \texttt{entropy}(\texttt{age}, \texttt{D}) = 0.940 - 0.694 = 0.246$$

- Similarly we have:
  - $\texttt{Gain}(\texttt{income}, \texttt{D}) = 0.029$
  - $\texttt{Gain}(\texttt{student}, \texttt{D}) = 0.151$
  - $\texttt{Gain}(\texttt{credit\_rating}, \texttt{D}) = 0.048$

ID3 (7/8)

- Now, we can compute the information gain $\texttt{Gain(A)}$ for the feature $\texttt{A = age}$.

$$\texttt{Gain}(\texttt{age}, \texttt{D}) = \texttt{entropy}(\texttt{D}) - \texttt{entropy}(\texttt{age}, \texttt{D}) = 0.940 - 0.694 = 0.246$$

- Similarly we have:
  - $\texttt{Gain}(\texttt{income}, \texttt{D}) = 0.029$
  - $\texttt{Gain}(\texttt{student}, \texttt{D}) = 0.151$
  - $\texttt{Gain}(\texttt{credit\_rating}, \texttt{D}) = 0.048$

- The age has the highest information gain among the attributes, it is selected as the splitting feature.

ID3 (8/8)

- The bias problem: information gain prefers to select features having a large number of values.

ID3 (8/8)

- The bias problem: information gain prefers to select features having a large number of values.

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

- For example, a split on RID would result in a large number of partitions.
  - Each partition is pure.
  - Info product $\mathtt{entropy}(\mathtt{RID}, \mathtt{D}) = 0$, thus, the information gained by partitioning on this feature is maximal.

ID3 (8/8)

- The bias problem: information gain prefers to select features having a large number of values.

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

- For example, a split on RID would result in a large number of partitions.
  - Each partition is pure.
  - Info product $\texttt{entropy}(\texttt{RID}, \texttt{D}) = 0$, thus, the information gained by partitioning on this feature is maximal.
- Clearly, such a partitioning is useless for classification.

C4.5 (1/2)

- C4.5 is a successor of ID3 that overcomes its bias problem.
- It normalizes the information gain using a split information value:

$$\mathtt{SplitInfo}(\mathtt{A},\mathtt{D}) = -\sum_{\mathtt{j}=1}^{\mathtt{v}} \frac{|\mathtt{D_j}|}{|\mathtt{D}|} \log_2\left(\frac{|\mathtt{D_j}|}{|\mathtt{D}|}\right)$$

$$\mathtt{GainRatio}(\mathtt{A},\mathtt{D}) = \frac{\mathtt{Gain}(\mathtt{A},\mathtt{D})}{\mathtt{SplitInfo}(\mathtt{A},\mathtt{D})}$$

C4.5 (2/2)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\mathtt{SplitInfo}(\mathtt{A},\mathtt{D}) = -\sum_{j=1}^{v} \frac{|\mathtt{D}_j|}{|\mathtt{D}|} \log_2\left(\frac{|\mathtt{D}_j|}{|\mathtt{D}|}\right)$$

$$\mathtt{SplitInfo}(\mathtt{income},\mathtt{D}) = -\frac{4}{14}\log_2\left(\frac{4}{14}\right) - \frac{6}{14}\log_2\left(\frac{6}{14}\right) - \frac{4}{14}\log_2\left(\frac{4}{14}\right) = 1.557$$

- $\mathtt{Gain}(\mathtt{income},\mathtt{D}) = 0.029$, therefore $\mathtt{GainRatio}(\mathtt{income},\mathtt{D}) = \frac{0.029}{1.557} = 0.019$.

Gini Impurity

CART (1/8)

- CART (Classification And Regression Tree) considers a binary split for each feature.

CART (1/8)

- CART (Classification And Regression Tree) considers a binary split for each feature.
- It uses the Gini index to measure the misclassification (impurity of $\mathtt{D}$).

$$\mathtt{Gini}(\mathtt{D}) = 1 - \sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p}_{\mathtt{i}}^2$$

CART (1/8)

- CART (Classification And Regression Tree) considers a binary split for each feature.
- It uses the Gini index to measure the misclassification (impurity of $\mathtt{D}$).

$$\mathtt{Gini(D)} = 1 - \sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p}_{\mathtt{i}}^2$$

- $\mathtt{p_i}$ is the probability that an instance in $\mathtt{D}$ belongs to class $\mathtt{i}$, with $\mathtt{m}$ distinct classes.

CART (1/8)

- CART (Classification And Regression Tree) considers a binary split for each feature.
- It uses the Gini index to measure the misclassification (impurity of $\mathtt{D}$).

$$\mathtt{Gini}(\mathtt{D}) = 1 - \sum_{i=1}^{m} p_i^2$$

- p_i is the probability that an instance in $\mathtt{D}$ belongs to class $\mathtt{i}$, with $\mathtt{m}$ distinct classes.
- It will be zero if all partitions are pure. Why?

CART (1/8)

- CART (Classification And Regression Tree) considers a binary split for each feature.
- It uses the Gini index to measure the misclassification (impurity of $\mathtt{D}$).

$$\mathtt{Gini}(\mathtt{D}) = 1 - \sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p}_{\mathtt{i}}^{2}$$

- $\mathtt{p}_{\mathtt{i}}$ is the probability that an instance in $\mathtt{D}$ belongs to class $\mathtt{i}$, with $\mathtt{m}$ distinct classes.
- It will be zero if all partitions are pure. Why?
- We need to determine the splitting criterion: splitting feature + splitting subset.

CART (2/8)

- Assume $\mathtt{A}$ is a discrete-valued feature with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$, occurring in $\mathtt{D}$.

CART (2/8)

- Assume $\mathtt{A}$ is a discrete-valued feature with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$, occurring in $\mathtt{D}$.
- $\mathtt{S}_\mathtt{A}$ will be all possible subsets of $\mathtt{A}$.

CART (2/8)

- Assume $\mathtt{A}$ is a discrete-valued feature with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$, occurring in $\mathtt{D}$.
- $\mathtt{S}_\mathtt{A}$ will be all possible subsets of $\mathtt{A}$.
  - E.g., $\mathtt{A} = \mathtt{income} = \{\mathtt{low}, \mathtt{medium}, \mathtt{high}\}$
  - $\mathtt{S}_\mathtt{A} = \{\{\mathtt{low}, \mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{medium}\}, \{\mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{high}\}, \{\mathtt{low}\}, \{\mathtt{medium}\}, \{\mathtt{high}\}, \{\}\}$

CART (2/8)

- Assume $\mathtt{A}$ is a discrete-valued feature with $\mathtt{v}$ distinct values, $\{\mathtt{a}_1, \mathtt{a}_2, \cdots, \mathtt{a}_\mathtt{v}\}$, occurring in $\mathtt{D}$.
- $\mathtt{S}_\mathtt{A}$ will be all possible subsets of $\mathtt{A}$.
  - E.g., $\mathtt{A} = \mathtt{income} = \{\mathtt{low}, \mathtt{medium}, \mathtt{high}\}$
  - $\mathtt{S}_\mathtt{A} = \{\{\mathtt{low}, \mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{medium}\}, \{\mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{high}\}, \{\mathtt{low}\}, \{\mathtt{medium}\}, \{\mathtt{high}\}, \{\}\}$
  - The test is of the form $\mathtt{D}_1 \in \mathtt{s}_\mathtt{A}$?, where $\mathtt{s}_\mathtt{A}$ is a subset of $\mathtt{S}_\mathtt{A}$, e.g., $\mathtt{s}_\mathtt{A} = \{\mathtt{low}, \mathtt{high}\}$.

CART (3/8)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\mathtt{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2$$

CART (3/8)

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\mathtt{Gini}(\mathtt{D}) = 1 - \sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{p}_{\mathtt{i}}^2$$

$$\text{label} = \mathtt{buys\_computer} \Rightarrow \mathtt{m} = 2$$

$$\mathtt{Gini}(\mathtt{D}) = 1 - (\frac{9}{14})^2 - (\frac{5}{14})^2 = 0.459$$

CART (4/8)

- If a binary split on $\mathtt{A}$ partitions $\mathtt{D}$ into $\mathtt{D}_1$ and $\mathtt{D}_2$, the Gini index of $\mathtt{D}$ given that partitioning is:

$$\mathtt{Gini}(\mathtt{A},\mathtt{D}) = \frac{|\mathtt{D}_1|}{\mathtt{D}}\mathtt{Gini}(\mathtt{D}_1) + \frac{|\mathtt{D}_2|}{\mathtt{D}}\mathtt{Gini}(\mathtt{D}_2)$$

CART (4/8)

- If a binary split on $\mathtt{A}$ partitions $\mathtt{D}$ into $\mathtt{D}_1$ and $\mathtt{D}_2$, the Gini index of $\mathtt{D}$ given that partitioning is:

$$\mathtt{Gini}(\mathtt{A},\mathtt{D}) = \frac{|\mathtt{D}_1|}{\mathtt{D}}\mathtt{Gini}(\mathtt{D}_1) + \frac{|\mathtt{D}_2|}{\mathtt{D}}\mathtt{Gini}(\mathtt{D}_2)$$

- The subset that gives the minimum Gini index is selected as its splitting subset.

CART (5/8)

- For a feature $\mathtt{A} = \mathtt{income}$, we consider each of the possible splitting subsets.

CART (5/8)

- For a feature $\mathtt{A} = \mathtt{income}$, we consider each of the possible splitting subsets.
  - $\mathtt{S_A} = \{\{\mathtt{low}, \mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{medium}\}, \{\mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{high}\}, \{\mathtt{low}\}, \{\mathtt{medium}\}, \{\mathtt{high}\}, \{\}\}$

CART (5/8)

- For a feature $\texttt{A} = \texttt{income}$, we consider each of the possible splitting subsets.
  - $\texttt{S}_\texttt{A} = \{\{\texttt{low}, \texttt{medium}, \texttt{high}\}, \{\texttt{low}, \texttt{medium}\}, \{\texttt{medium}, \texttt{high}\}, \{\texttt{low}, \texttt{high}\}, \{\texttt{low}\}, \{\texttt{medium}\}, \{\texttt{high}\}, \{\}\}$

- Assume, we choose the splitting subset $\texttt{s}_\texttt{A} = \{\texttt{low}, \texttt{medium}\}$.

CART (5/8)

- For a feature $\mathtt{A} = \mathtt{income}$, we consider each of the possible splitting subsets.
  - $\mathtt{S_A} = \{\{\mathtt{low}, \mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{medium}\}, \{\mathtt{medium}, \mathtt{high}\}, \{\mathtt{low}, \mathtt{high}\}, \{\mathtt{low}\}, \{\mathtt{medium}\}, \{\mathtt{high}\}, \{\}\}$
- Assume, we choose the splitting subset $\mathtt{s_A} = \{\mathtt{low}, \mathtt{medium}\}$.
- Consider partition $\mathtt{D_1}$ satisfies the condition $\mathtt{D_1} \in \mathtt{s_A}$, and $\mathtt{D_2}$ does not.

$$\mathtt{Gini}_{\mathtt{income}\in\{\mathtt{low},\mathtt{medium}\}}(\mathtt{A}, \mathtt{D}) = \frac{10}{14}\mathtt{Gini}(\mathtt{D_1}) + \frac{4}{14}\mathtt{Gini}(\mathtt{D_2})$$

$$= \frac{10}{14}\mathtt{Gini}(1 - (\frac{7}{10})^2 - (\frac{3}{10})^2) + \frac{4}{14}(1 - (\frac{2}{4})^2 - (\frac{2}{4})^2) = 0.443$$

CART (6/8)

- Similarly, we calculate the Gini index values for splits on the remaining subsets.

$$\texttt{Gini}_{\texttt{income}\in\{\texttt{low},\texttt{medium}\}}(\texttt{A},\texttt{D}) = \texttt{Gini}_{\texttt{income}\in\{\texttt{high}\}}(\texttt{A},\texttt{D}) = 0.443$$
$$\texttt{Gini}_{\texttt{income}\in\{\texttt{low},\texttt{high}\}}(\texttt{A},\texttt{D}) = \texttt{Gini}_{\texttt{income}\in\{\texttt{medium}\}}(\texttt{A},\texttt{D}) = 0.458$$
$$\texttt{Gini}_{\texttt{income}\in\{\texttt{medium},\texttt{high}\}}(\texttt{A},\texttt{D}) = \texttt{Gini}_{\texttt{income}\in\{\texttt{low}\}}(\texttt{A},\texttt{D}) = 0.450$$

CART (6/8)

- Similarly, we calculate the Gini index values for splits on the remaining subsets.

$$\mathtt{Gini}_{\mathtt{income}\in\{\mathtt{low},\mathtt{medium}\}}(\mathtt{A},\mathtt{D}) = \mathtt{Gini}_{\mathtt{income}\in\{\mathtt{high}\}}(\mathtt{A},\mathtt{D}) = 0.443$$
$$\mathtt{Gini}_{\mathtt{income}\in\{\mathtt{low},\mathtt{high}\}}(\mathtt{A},\mathtt{D}) = \mathtt{Gini}_{\mathtt{income}\in\{\mathtt{medium}\}}(\mathtt{A},\mathtt{D}) = 0.458$$
$$\mathtt{Gini}_{\mathtt{income}\in\{\mathtt{medium},\mathtt{high}\}}(\mathtt{A},\mathtt{D}) = \mathtt{Gini}_{\mathtt{income}\in\{\mathtt{low}\}}(\mathtt{A},\mathtt{D}) = 0.450$$

- The best binary split for attribute $\mathtt{A} = \mathtt{income}$ is on $\mathtt{s}_\mathtt{A} = \{\mathtt{low},\mathtt{medium}\}$ because it minimizes the Gini index.

CART (7/8)

- But, which feature?

CART (7/8)

- But, which feature?
- The reduction in impurity that would be incurred by a binary split on feature $\mathtt{A}$ is:

$$\Delta\mathtt{Gini}(\mathtt{A}) = \mathtt{Gini}(\mathtt{D}) - \mathtt{Gini}(\mathtt{A}, \mathtt{D})$$

CART (7/8)

- But, which feature?
- The reduction in impurity that would be incurred by a binary split on feature $\mathtt{A}$ is:

$$\Delta\mathtt{Gini}(\mathtt{A}) = \mathtt{Gini}(\mathtt{D}) - \mathtt{Gini}(\mathtt{A}, \mathtt{D})$$

- The feature that maximizes the reduction in impurity (has the minimum Gini index) is selected as the splitting feature.

CART (8/8)

- Now, we can compute the information gain $\texttt{Gain}(\texttt{A})$ for different features.
  - $\Delta\texttt{Gini}(\texttt{income}) = 0.459 - 0.443 = 0.016$
  - $\Delta\texttt{Gini}(\texttt{age}) = 0.459 - 0.357 = 0.102$
  - $\Delta\texttt{Gini}(\texttt{student}) = 0.459 - 0.367 = 0.092$
  - $\Delta\texttt{Gini}(\texttt{credit\_rating}) = 0.459 - 0.429 = 0.03$

CART (8/8)

- Now, we can compute the information gain $\texttt{Gain}(\texttt{A})$ for different features.
  - $\Delta\texttt{Gini}(\texttt{income}) = 0.459 - 0.443 = 0.016$
  - $\Delta\texttt{Gini}(\texttt{age}) = 0.459 - 0.357 = 0.102$
  - $\Delta\texttt{Gini}(\texttt{student}) = 0.459 - 0.367 = 0.092$
  - $\Delta\texttt{Gini}(\texttt{credit\_rating}) = 0.459 - 0.429 = 0.03$
- The feature $\texttt{A} = \texttt{age}$ and splitting subset $\texttt{s}_\texttt{A} = \{\texttt{youth}, \texttt{senior}\}$ gives the minimum Gini index overall.

Decision Tree in Spark (1/4)

- Two classes in `spark.ml`.
- Regression: `DecisionTreeRegressor`

```
val dt_regressor = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("features")
val model = dt_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```

Decision Tree in Spark (1/4)

- Two classes in `spark.ml`.
- Regression: `DecisionTreeRegressor`

```
val dt_regressor = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("features")
val model = dt_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```

- Classifier: `DecisionTreeClassifier`

```
val dt_classifier = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features")
val model = dt_classifier.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```

Decision Tree in Spark (2/4)

- Input and output columns

Decision Tree in Spark (2/4)

- Input and output columns
- `labelCol` and `featuresCol` identify label and features column's names.

Decision Tree in Spark (2/4)

- Input and output columns
- `labelCol` and `featuresCol` identify label and features column's names.
- `predictionCol` indicates the predicted label.

Decision Tree in Spark (2/4)

- Input and output columns
- `labelCol` and `featuresCol` identify label and features column's names.
- `predictionCol` indicates the predicted label.
- `rawPredictionCol` is a vector of length of number of classes, with the counts of training instance labels at the tree node which makes the prediction.

Decision Tree in Spark (2/4)

- Input and output columns
- `labelCol` and `featuresCol` identify label and features column's names.
- `predictionCol` indicates the predicted label.
- `rawPredictionCol` is a vector of length of number of classes, with the counts of training instance labels at the tree node which makes the prediction.
- `probabilityCol` is a vector of length of number of classes equal to `rawPrediction` normalized to a multinomial distribution.

Decision Tree in Spark (3/4)

- Tunable parameters

Decision Tree in Spark (3/4)

- Tunable parameters
- `maxBins`: number of bins used when discretizing continuous features.

Decision Tree in Spark (3/4)

- Tunable parameters
- maxBins: number of bins used when discretizing continuous features.
- impurity: impurity measure used to choose between candidate splits, e.g., entropy and gini.

```
val maxBins = ...
val dt_classifier = new DecisionTreeClassifier().setMaxBins(maxBins).setImpurity("gini")
```

Decision Tree in Spark (4/4)

- Stopping criteria that determines when the tree stops building.

Decision Tree in Spark (4/4)

- Stopping criteria that determines when the tree stops building.
- `maxDepth`: maximum depth of a tree.

Decision Tree in Spark (4/4)

- Stopping criteria that determines when the tree stops building.
- `maxDepth`: maximum depth of a tree.
- `minInstancesPerNode`: for a node to be split further, each of its children must receive at least this number of training instances.

Decision Tree in Spark (4/4)

- Stopping criteria that determines when the tree stops building.
- `maxDepth`: maximum depth of a tree.
- `minInstancesPerNode`: for a node to be split further, each of its children must receive at least this number of training instances.
- `minInfoGain`: for a node to be split further, the split must improve at least this much (in terms of information gain).

```
val maxDepth = ...
val minInstancesPerNode = ...
val minInfoGain = ...
val dt_classifier = new DecisionTreeClassifier()
                    .setMaxDepth(maxDepth)
                    .setMinInstancesPerNode(minInstancesPerNode)
                    .setMinInfoGain(minInfoGain)
```

Ensemble Methods

Wisdom of the Crowd

- Ask a complex question to thousands of random people, then aggregate their answers.
- In many cases, this aggregated answer is better than an expert's answer.

Wisdom of the Crowd

- Ask a complex question to thousands of random people, then aggregate their answers.
- In many cases, this aggregated answer is better than an expert's answer.
- This is called the wisdom of the crowd.

Wisdom of the Crowd

- Ask a complex question to thousands of random people, then aggregate their answers.
- In many cases, this aggregated answer is better than an expert's answer.
- This is called the wisdom of the crowd.
- Similarly, the aggregated estimations of a group of estimators (e.g., classifiers or regressors), often gets better estimations than with the best individual estimator.
- A group of estimators is an ensemble, and this technique is called Ensemble Learning.

Ensemble Learning

- Two main categories of ensemble learning algorithms.

Ensemble Learning

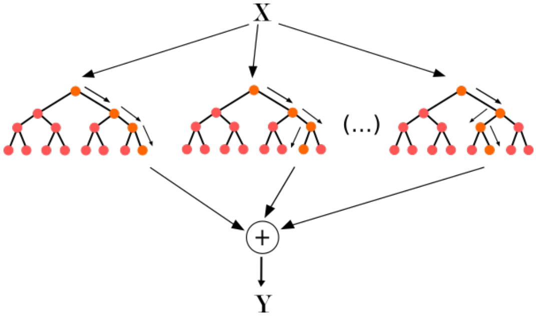
- Two main categories of ensemble learning algorithms.
- Bagging
  - Use the same training algorithm for every estimator, but to train them on different random subsets of the training set.
  - E.g., random forest

Ensemble Learning

- Two main categories of ensemble learning algorithms.
- Bagging
  - Use the same training algorithm for every estimator, but to train them on different random subsets of the training set.
  - E.g., random forest
- Boosting
  - Train estimators sequentially, each trying to correct its predecessor.
  - E.g., adaboost and gradient boosting

Random Forest

- Random forest builds multiple decision trees that are most of the time trained with the bagging method.
- It, then, merges the trees together to get a more accurate and stable prediction.



Random Forest in Spark (1/2)

- Two classes in `spark.ml`.
- Regression: `RandomForestRegressor`

```
val rf_regressor = new RandomForestRegressor().setLabelCol("label")
                                              .setFeaturesCol("features").setNumTrees(10)
val model = rf_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
```

Random Forest in Spark (1/2)

- Two classes in `spark.ml`.
- Regression: `RandomForestRegressor`

```
val rf_regressor = new RandomForestRegressor().setLabelCol("label")
                                              .setFeaturesCol("features").setNumTrees(10)
val model = rf_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
```

- Classifier: `RandomForestClassifier`

```
val rf_classifier = new RandomForestClassifier().setLabelCol("label")
                                                .setFeaturesCol("features").setNumTrees(10)
val model = rf_classifier.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "label", "features").show(5)
```

Random Forest in Spark (2/2)

- `numTrees`: number of trees in the forest.

Random Forest in Spark (2/2)

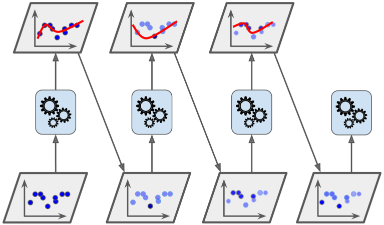
- `numTrees`: number of trees in the forest.
- `subsamplingRate`: specifies the size of the dataset used for training each tree in the forest, as a fraction of the size of the original dataset.
  - Default is 1.0 and decreasing it can speed up training.

Random Forest in Spark (2/2)

- `numTrees`: number of trees in the forest.
- `subsamplingRate`: specifies the size of the dataset used for training each tree in the forest, as a fraction of the size of the original dataset.
  - Default is 1.0 and decreasing it can speed up training.
- `featureSubsetStrategy`: number of features to use as candidates for splitting at each tree node, as a fraction of the total number of features.
  - Possible values: `auto`, `all`, `onethird`, `sqrt`, `log2`, `n`

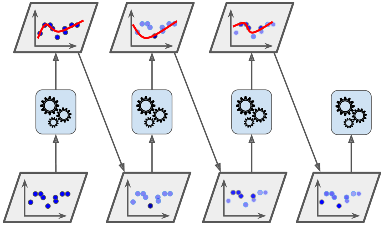
AdaBoost (1/3)

- AdaBoost: train a new estimator by paying more attention to the training instances that the predecessor underfitted.



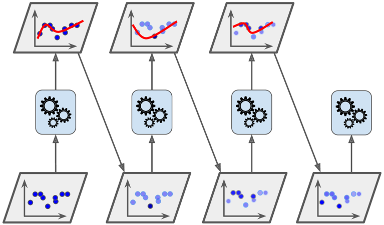
AdaBoost (1/3)

- AdaBoost: train a new estimator by paying more attention to the training instances that the predecessor underfitted.
- Each estimator is trained on a random subset of the total training set.



AdaBoost (1/3)

- AdaBoost: train a new estimator by paying more attention to the training instances that the predecessor underfitted.
- Each estimator is trained on a random subset of the total training set.
- AdaBoost assigns a weight to each training instance, which determines the probability that each instance should appear in the training set.



AdaBoost (2/3)

- Each instance weight $\mathtt{h}^{(\mathtt{i})}$ is initially set to $\frac{\mathtt{1}}{\mathtt{m}}$ for $\mathtt{m}$ instances.

AdaBoost (2/3)

- Each instance weight $\mathtt{h}^{(\mathtt{i})}$ is initially set to $\frac{1}{\mathtt{m}}$ for $\mathtt{m}$ instances.
- An estimator $\mathtt{j}$ is trained and its weighted error rate $\mathtt{r_j}$ is computed as follows:

$$\mathtt{r_j} = \frac{\sum_{\mathtt{i}=1, \hat{\mathtt{y}}_\mathtt{j}^{(\mathtt{i})} \neq \mathtt{y}_\mathtt{j}^{(\mathtt{i})}}^{\mathtt{m}} \mathtt{h}^{(\mathtt{i})}}{\sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{h}^{(\mathtt{i})}}$$

AdaBoost (2/3)

- Each instance weight $\mathtt{h}^{(\mathtt{i})}$ is initially set to $\frac{1}{\mathtt{m}}$ for $\mathtt{m}$ instances.
- An estimator $\mathtt{j}$ is trained and its weighted error rate $\mathtt{r}_\mathtt{j}$ is computed as follows:

$$\mathtt{r}_\mathtt{j} = \frac{\sum_{\mathtt{i}=1,\hat{\mathtt{y}}_\mathtt{j}^{(\mathtt{i})} \neq \mathtt{y}_\mathtt{j}^{(\mathtt{i})}}^{\mathtt{m}} \mathtt{h}^{(\mathtt{i})}}{\sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{h}^{(\mathtt{i})}}$$

- The $\mathtt{j}$th estimator's weight $\alpha_\mathtt{j}$ is then computed as follows:

$$\alpha_\mathtt{j} = \eta \frac{1 - \mathtt{r}_\mathtt{j}}{\mathtt{r}_\mathtt{j}}$$

AdaBoost (3/3)

- Next the instance weights are updated:

$$\mathtt{h}^{(\mathtt{i})} = \begin{cases} \mathtt{h}^{(\mathtt{i})} & \text{if} \quad \hat{\mathtt{y}}_{\mathtt{j}}^{(\mathtt{i})} = \mathtt{y}_{\mathtt{j}}^{(\mathtt{i})} \\ \mathtt{h}^{(\mathtt{i})} \mathtt{e}^{\alpha_{\mathtt{j}}} & \text{if} \quad \hat{\mathtt{y}}_{\mathtt{j}}^{(\mathtt{i})} \neq \mathtt{y}_{\mathtt{j}}^{(\mathtt{i})} \end{cases}$$

AdaBoost (3/3)

- Next the instance weights are updated:

$$\mathtt{h}^{(\mathtt{i})} = \begin{cases} \mathtt{h}^{(\mathtt{i})} & \text{if} \quad \hat{\mathtt{y}}_{\mathtt{j}}^{(\mathtt{i})} = \mathtt{y}_{\mathtt{j}}^{(\mathtt{i})} \\ \mathtt{h}^{(\mathtt{i})}\mathtt{e}^{\alpha_{\mathtt{j}}} & \text{if} \quad \hat{\mathtt{y}}_{\mathtt{j}}^{(\mathtt{i})} \neq \mathtt{y}_{\mathtt{j}}^{(\mathtt{i})} \end{cases}$$

- Then, a new estimator is trained using the updated weights, and the whole process is repeated.

AdaBoost (3/3)

- Next the instance weights are updated:

$$\mathtt{h}^{(\mathtt{i})} = \begin{cases} \mathtt{h}^{(\mathtt{i})} & \text{if} \quad \hat{\mathtt{y}}_{\mathtt{j}}^{(\mathtt{i})} = \mathtt{y}_{\mathtt{j}}^{(\mathtt{i})} \\ \mathtt{h}^{(\mathtt{i})} \mathtt{e}^{\alpha_{\mathtt{j}}} & \text{if} \quad \hat{\mathtt{y}}_{\mathtt{j}}^{(\mathtt{i})} \neq \mathtt{y}_{\mathtt{j}}^{(\mathtt{i})} \end{cases}$$

- Then, a new estimator is trained using the updated weights, and the whole process is repeated.

- To make predictions, AdaBoost computes the predictions of all the estimators and weighs them using the estimator weights $\alpha_{\mathtt{j}}$.

Gradient Boosting (1/3)

- Just like AdaBoost, Gradient Boosting works by sequentially adding estimators to an ensemble, each one correcting its predecessor.
- However, instead of tweaking the instance weights at every iteration, this method tries to fit the new estimator to the residual errors made by the previous estimator.

Gradient Boosting (2/3)

- Let's go through a regression example using Gradient Boosted Regression Trees.
- Fit the first estimator on the training set.

```
tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

- Now train the second estimator on the residual errors made by the first estimator.

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

Gradient Boosting (3/3)

- Then we train the third estimator on the residual errors made by the second estimator.

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

- Now we have an ensemble containing three trees.
- It can make predictions on a new instance simply by adding up the predictions of all the trees.

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

Gradient Boosting in Spark (1/2)

- Two classes in `spark.ml`.
- Regression: `GBTRegressor`

```
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("features")
                             .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```

Gradient Boosting in Spark (1/2)

- Two classes in `spark.ml`.
- Regression: `GBTRegressor`

```
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("features")
                             .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```

- Classifier: `GBTClassifier`

```
val gbt = new GBTClassifier().setLabelCol("label").setFeaturesCol("features")
                              .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```

Summary

Summary

- Decision tree
  - Top-down training algorithm
  - Termination condition
  - Feature selection: entropy, gini
- Ensemble models
  - Bagging: random forest
  - Boosting: AdaBoost, Gradient Boosting

Reference

- Aurélien Géron, Hands-On Machine Learning (Ch. 5, 6, 7)
- Matei Zaharia et al., Spark - The Definitive Guide (Ch. 27)

Questions?