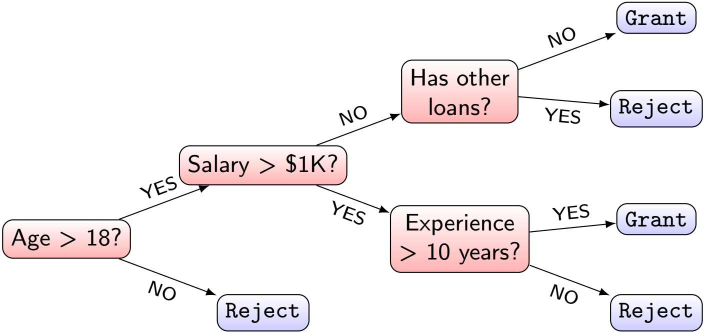
# Decision Trees. Bagging. Random Forest

Evgeny Burnaev

Skoltech, Moscow, Russia

# Outline

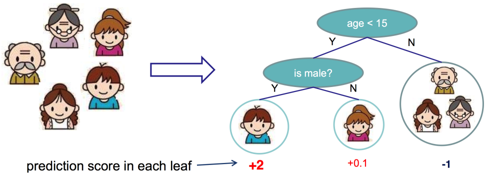
# Decision making at a bank



A decision tree consists of

- Nodes
  — Test for the value of a certain attribute
- Edges
  — Correspond to the outcome of a test
  — Connect to the next node of leaf
- Leaves
  — Terminal nodes that predict the outcome

Input: age, gender, occupation,... $\Rightarrow$ Does the person like computer games?

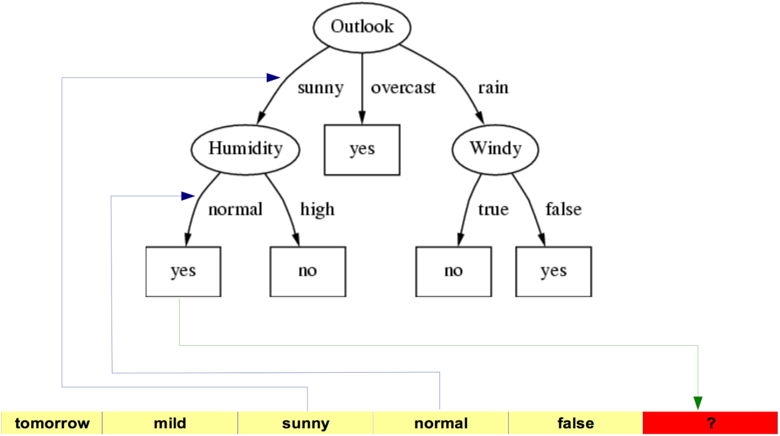


# Example: weather prediction

| Day | Temperature | Outlook | Humidity | Windy | Play Golf? |
|---|---|---|---|---|---|
| 07-05 | hot | sunny | high | false | no |
| 07-06 | hot | sunny | high | true | no |
| 07-07 | hot | overcast | high | false | yes |
| 07-09 | cool | rain | normal | false | yes |
| 07-10 | cool | overcast | normal | true | yes |
| 07-12 | mild | sunny | high | false | no |
| 07-14 | cool | sunny | normal | false | yes |
| 07-15 | mild | rain | normal | false | yes |
| 07-20 | mild | sunny | normal | true | yes |
| 07-21 | mild | overcast | high | true | yes |
| 07-22 | hot | overcast | normal | false | yes |
| 07-23 | mild | rain | high | true | no |
| 07-26 | cool | rain | normal | true | no |
| 07-30 | mild | rain | high | false | yes |

| | | | | | |
|---|---|---|---|---|---|
| today | cool | sunny | normal | false | ? |
| tomorrow | mild | sunny | normal | false | ? |

# Decision Tree Learning



- Typical decision tree learning algorithms
  - Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  - Divide the problem in subproblems
  - Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Typical decision tree learning algorithms
  — Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  — Divide the problem in subproblems
  — Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Typical decision tree learning algorithms
  - Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  - Divide the problem in subproblems
  - Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Typical decision tree learning algorithms
  — Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  — Divide the problem in subproblems
  — Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Typical decision tree learning algorithms
  - Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  - Divide the problem in subproblems
  - Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Typical decision tree learning algorithms
  - Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  - Divide the problem in subproblems
  - Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Typical decision tree learning algorithms
  - Top-Down Induction of Decision Trees
- Learn trees in a Top-Down fashion
  - Divide the problem in subproblems
  - Solve each subproblem
- Basic Divide-And-Conquer algorithm
  1. Select a test for root node
     Create branch for each possible outcome of the test
  2. Split instances into subsets
     One for each branch extending from the node
  3. Repeat recursively for each branch, using only instances that reach the branch
  4. Stop recursion for a branch if all its instances have the same class

- Function ID3
  - **Input**: example set $S_m$
  - **Output**: Decision Tree $DT$
- If all examples in $S_m$ belong to the same class $c$
  - Return a new leaf and label it with $c$
- Else
  1. Select an attribute $x_j$ according to some heuristic approach
  2. Generate a new node in $DT$ with $x_j$ as test
  3. For each possible value $e_r$ of $x_j$
     a) Let $S_r$ = all examples in $S_m$ with $x_j = e_r$
     b) Use ID3 to construct a decision tree $DT_r$ for examples in $S_r$
     c) Generate an edge that connects $DT$ and $DT_r$

# ID3 Algorithm

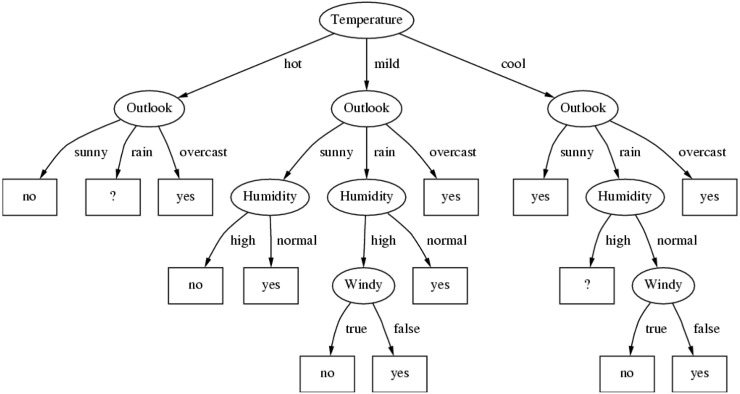
- Function ID3
  - **Input**: example set $S_m$
  - **Output**: Decision Tree $DT$
- If all examples in $S_m$ belong to the same class $c$
  - Return a new leaf and label it with $c$
- Else
  1. Select an attribute $x_j$ according to some heuristic approach
  2. Generate a new node in $DT$ with $x_j$ as test
  3. For each possible value $e_r$ of $x_j$
     a) Let $S_r$ = all examples in $S_m$ with $x_j = e_r$
     b) Use ID3 to construct a decision tree $DT_r$ for examples in $S_r$
     c) Generate an edge that connects $DT$ and $DT_r$

- Function ID3
  - **Input**: example set $S_m$
  - **Output**: Decision Tree $DT$
- If all examples in $S_m$ belong to the same class $c$
  - Return a new leaf and label it with $c$
- Else
  1. Select an attribute $x_j$ according to some heuristic approach
  2. Generate a new node in $DT$ with $x_j$ as test
  3. For each possible value $e_r$ of $x_j$
     a) Let $S_r$ = all examples in $S_m$ with $x_j = e_r$
     b) Use ID3 to construct a decision tree $DT_r$ for examples in $S_r$
     c) Generate an edge that connects $DT$ and $DT_r$

- Function ID3
  - **Input**: example set $S_m$
  - **Output**: Decision Tree $DT$
- If all examples in $S_m$ belong to the same class $c$
  - Return a new leaf and label it with $c$
- Else
  1. Select an attribute $x_j$ according to some heuristic approach
  2. Generate a new node in $DT$ with $x_j$ as test
  3. For each possible value $e_r$ of $x_j$
     a) Let $S_r$ = all examples in $S_m$ with $x_j = e_r$
     b) Use ID3 to construct a decision tree $DT_r$ for examples in $S_r$
     c) Generate an edge that connects $DT$ and $DT_r$

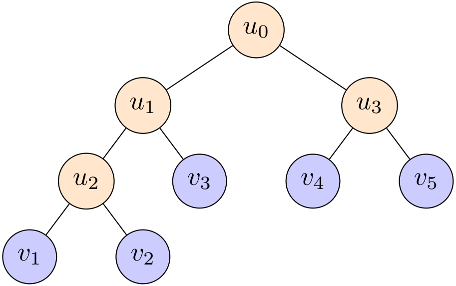
- Function ID3
  - **Input**: example set $S_m$
  - **Output**: Decision Tree $DT$
- If all examples in $S_m$ belong to the same class $c$
  - Return a new leaf and label it with $c$
- Else
  1. Select an attribute $x_j$ according to some heuristic approach
  2. Generate a new node in $DT$ with $x_j$ as test
  3. For each possible value $e_r$ of $x_j$
     a) Let $S_r$ = all examples in $S_m$ with $x_j = e_r$
     b) Use ID3 to construct a decision tree $DT_r$ for examples in $S_r$
     c) Generate an edge that connects $DT$ and $DT_r$

- Function ID3
  - **Input**: example set $S_m$
  - **Output**: Decision Tree $DT$
- If all examples in $S_m$ belong to the same class $c$
  - Return a new leaf and label it with $c$
- Else
  1. Select an attribute $x_j$ according to some heuristic approach
  2. Generate a new node in $DT$ with $x_j$ as test
  3. For each possible value $e_r$ of $x_j$
     a) Let $S_r =$ all examples in $S_m$ with $x_j = e_r$
     b) Use ID3 to construct a decision tree $DT_r$ for examples in $S_r$
     c) Generate an edge that connects $DT$ and $DT_r$

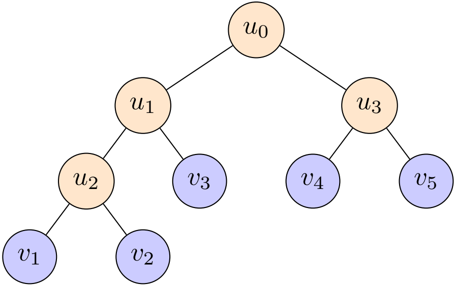



- also explains all of the training data
- will it generalize well to new data?

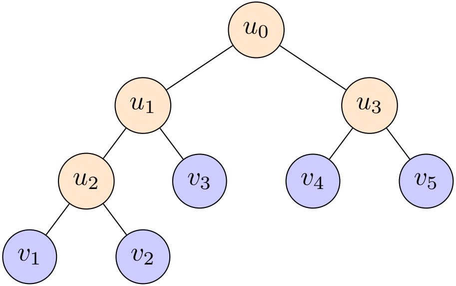
- Decision tree is often a binary tree $DT$
- Internal nodes $u \in DT$: *predicates* $\beta_u : X \to \{0, 1\}$
- Leafs $v \in DT$: *predictions* $y$



- Decision tree is often a binary tree $DT$
- Internal nodes $u \in DT$: *predicates* $\beta_u : X \to \{0, 1\}$
- Leafs $v \in DT$: *predictions* $y$
- Algorithm $h(\mathbf{x})$ starts at $u = u_0$
  - Compute $b = \beta_u(\mathbf{x})$
  - If $b = 0$, $u \leftarrow \mathrm{LeftChild}(u)$
  - If $b = 1$, $u \leftarrow \mathrm{RightChild}(u)$
  - If $u$ is a leaf, return some $y$



- Decision tree is often a binary tree $DT$
- Internal nodes $u \in DT$: *predicates* $\beta_u : X \to \{0, 1\}$
- Leafs $v \in DT$: *predictions* $y$
- Algorithm $h(\mathbf{x})$ starts at $u = u_0$
  - Compute $b = \beta_u(\mathbf{x})$
  - If $b = 0$, $u \leftarrow \mathrm{LeftChild}(u)$
  - If $b = 1$, $u \leftarrow \mathrm{RightChild}(u)$
  - If $u$ is a leaf, return some $y$
- In practice for a real variable: $\beta_u(\mathbf{x}; j, t) = 1[x_j < t]$



- Input: training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
   1. Greedily split $S_m$ into $S_1$ and $S_2$:

      $$S_1(j,t) = \{(\mathbf{x}, y) \in S_m | x_j \leq t\}, \qquad S_2(j,t) = \{(\mathbf{x}, y) \in S_m | x_j > t\}$$

      optimizing a given loss: $Q(S_m, j, t) \to \min\limits_{(j,t)}$
   2. Create internal node $u$ corresponding to the predicate $1[x_j < t]$
   3. If a stopping criterion is satisfied for $u$, declare it a leaf, setting some $c_u \in Y$ as leaf prediction
   4. If not, repeat 1–2 for $S_1(j,t)$ and $S_2(j,t)$
- Output: a decision tree $DT$

- Input: training set $S_m = \left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^{m}$
  1. Greedily split $S_m$ into $S_1$ and $S_2$:

     $$S_1(j,t) = \{(\mathbf{x}, y) \in S_m | x_j \leq t\}, \qquad S_2(j,t) = \{(\mathbf{x}, y) \in S_m | x_j > t\}$$

     optimizing a given loss: $Q(S_m, j, t) \to \min\limits_{(j,t)}$
  2. Create internal node $u$ corresponding to the predicate $1[x_j < t]$
  3. If a stopping criterion is satisfied for $u$,
     declare it a leaf, setting some $c_u \in Y$ as leaf prediction
  4. If not, repeat 1–2 for $S_1(j,t)$ and $S_2(j,t)$
- Output: a decision tree $DT$

- Input: training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
  1. Greedily split $S_m$ into $S_1$ and $S_2$:
     $$S_1(j,t) = \{(\mathbf{x}, y) \in S_m | x_j \leq t\}, \qquad S_2(j,t) = \{(\mathbf{x}, y) \in S_m | x_j > t\}$$
     optimizing a given loss: $Q(S_m, j, t) \to \min\limits_{(j,t)}$
  2. Create internal node $u$ corresponding to the predicate $1[x_j < t]$
  3. If a stopping criterion is satisfied for $u$, declare it a leaf, setting some $c_u \in Y$ as leaf prediction
  4. If not, repeat 1–2 for $S_1(j,t)$ and $S_2(j,t)$
- Output: a decision tree $DT$

- Input: training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$
  1. Greedily split $S_m$ into $S_1$ and $S_2$:

     $$S_1(j,t) = \{(\mathbf{x}, y) \in S_m | x_j \leq t\}, \qquad S_2(j,t) = \{(\mathbf{x}, y) \in S_m | x_j > t\}$$

     optimizing a given loss: $Q(S_m, j, t) \to \min\limits_{(j,t)}$
  2. Create internal node $u$ corresponding to the predicate $1[x_j < t]$
  3. If a stopping criterion is satisfied for $u$,
     declare it a leaf, setting some $c_u \in Y$ as leaf prediction
  4. If not, repeat 1–2 for $S_1(j,t)$ and $S_2(j,t)$
- Output: a decision tree $DT$

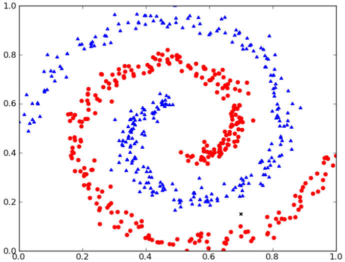
- Input: training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
  1. Greedily split $S_m$ into $S_1$ and $S_2$:

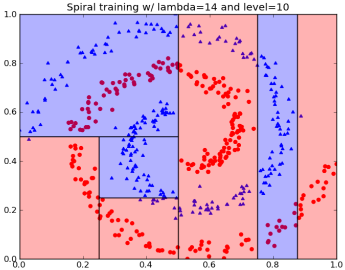
     $$S_1(j,t) = \{(\mathbf{x}, y) \in S_m | x_j \leq t\}, \qquad S_2(j,t) = \{(\mathbf{x}, y) \in S_m | x_j > t\}$$

     optimizing a given loss: $Q(S_m, j, t) \to \min\limits_{(j,t)}$
  2. Create internal node $u$ corresponding to the predicate $1[x_j < t]$
  3. If a stopping criterion is satisfied for $u$,
     declare it a leaf, setting some $c_u \in Y$ as leaf prediction
  4. If not, repeat 1–2 for $S_1(j,t)$ and $S_2(j,t)$
- Output: a decision tree $DT$

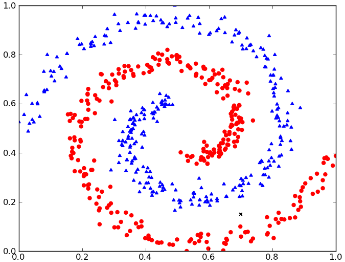
- Input: training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
  1. Greedily split $S_m$ into $S_1$ and $S_2$:
     $$S_1(j,t) = \{(\mathbf{x}, y) \in S_m | x_j \leq t\}, \qquad S_2(j,t) = \{(\mathbf{x}, y) \in S_m | x_j > t\}$$
     optimizing a given loss: $Q(S_m, j, t) \to \min\limits_{(j,t)}$
  2. Create internal node $u$ corresponding to the predicate $1[x_j < t]$
  3. If a stopping criterion is satisfied for $u$,
     declare it a leaf, setting some $c_u \in Y$ as leaf prediction
  4. If not, repeat 1–2 for $S_1(j,t)$ and $S_2(j,t)$
- Output: a decision tree $DT$

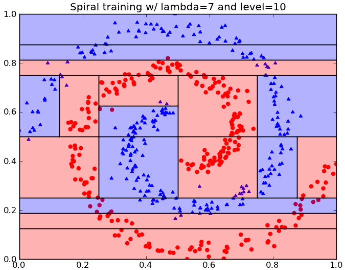
# Greedy tree learning for binary classification



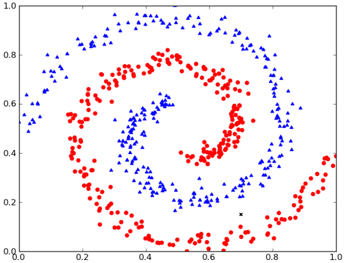



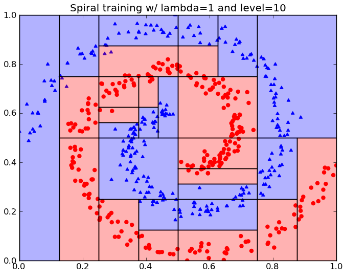
# Greedy tree learning for binary classification



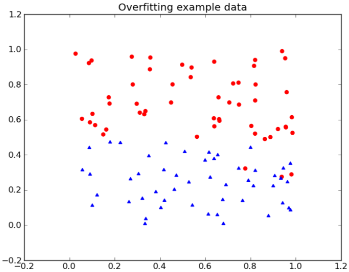



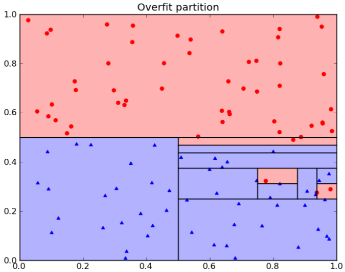
# Greedy tree learning for binary classification



# With decision trees, overfitting is extra-easy!

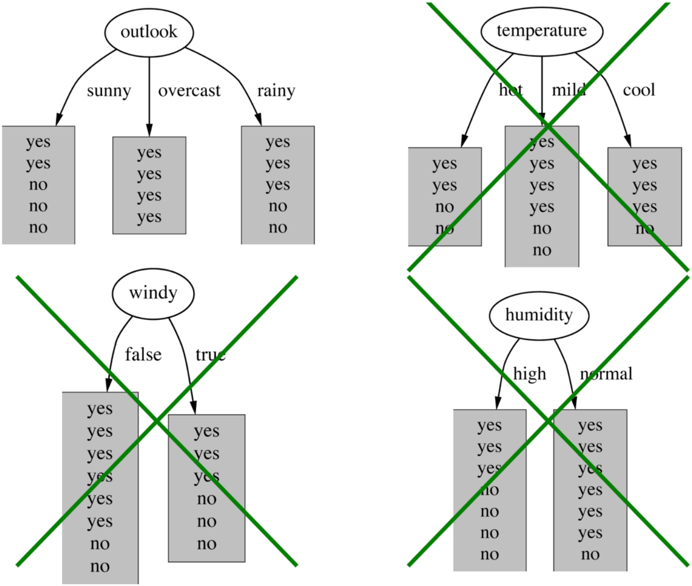


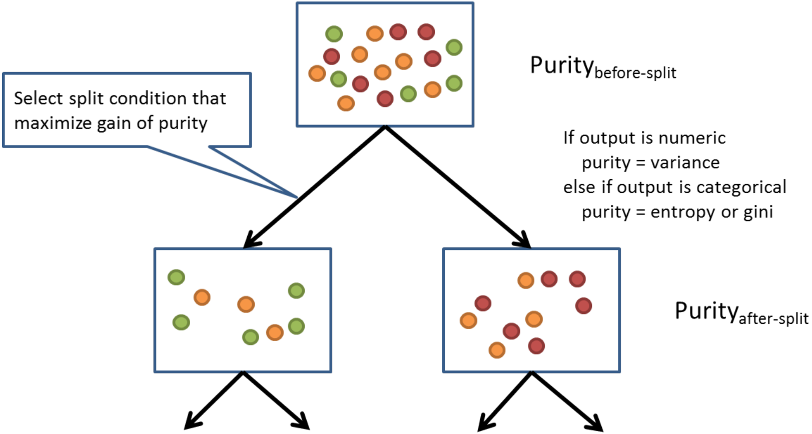
- Type of predicate in internal nodes
- The loss function $Q(S_m, j, t)$
- The stopping criterion
- Hacks: missing values, pruning, etc.

- CART, C4.5, ID3

# Which attribute to select as the root?



# The idea: maximize purity




Picture credit: https://dzone.com/refcardz/machine-learning-predictive

- We want to grow a simple tree
  - A good attribute split the data so that each successor node is as pure as possible, e.g. it contains mostly a singlel class
- We want a measure that prefers attributes with high degree of order:
  - Maximum order: all examples are of the same class
  - Minimum order: all classes are equally likely
- Entropy is a measure for (un)-orderliness

# What is a good Attribute?

- We want to grow a simple tree
  - A good attribute split the data so that each successor node is as pure as possible, e.g. it contains mostly a singlel class
- We want a measure that prefers attributes with high degree of order:
  - Maximum order: all examples are of the same class
  - Minimum order: all classes are equally likely
- Entropy is a measure for (un)-orderliness

# What is a good Attribute?

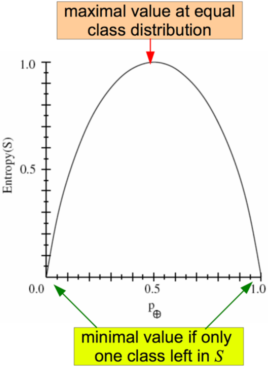
- We want to grow a simple tree
  - A good attribute split the data so that each successor node is as pure as possible, e.g. it contains mostly a singlel class
- We want a measure that prefers attributes with high degree of order:
  - Maximum order: all examples are of the same class
  - Minimum order: all classes are equally likely
- Entropy is a measure for (un)-orderliness

- $S$ is a set of examples
- $p_+$ is the proportion of examples in positive class
- $p_- = 1 - p_+$ is the proportion of examples in negative class
- Entropy

$$E(S) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_-$$

- Amount of un-orderliness in the class distribution of $S$
- $p_i$ is the proportion of examples in $S$ from the i-th class

$$E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_K \log_2 p_K = -\sum_{k=1}^{K} p_k \log_2 p_k$$



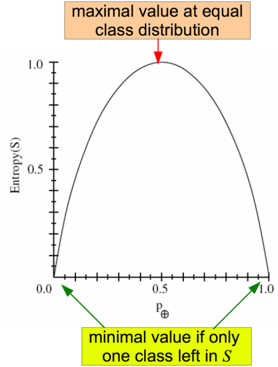
## Entropy (binary classification)

- $S$ is a set of examples
- $p_+$ is the proportion of examples in positive class
- $p_- = 1 - p_+$ is the proportion of examples in negative class
- Entropy

$$E(S) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_-$$

- Amount of un-orderliness in the class distribution of $S$
- $p_i$ is the proportion of examples in $S$ from the i-th class

$$E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_K \log_2 p_K = -\sum_{k=1}^{K} p_k \log_2 p_k$$

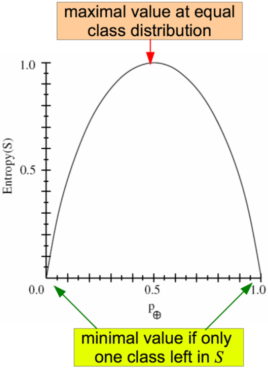


- $S$ is a set of examples
- $p_+$ is the proportion of examples in positive class
- $p_- = 1 - p_+$ is the proportion of examples in negative class
- Entropy

$$E(S) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_-$$

- Amount of un-orderliness in the class distribution of $S$
- $p_i$ is the proportion of examples in $S$ from the i-th class

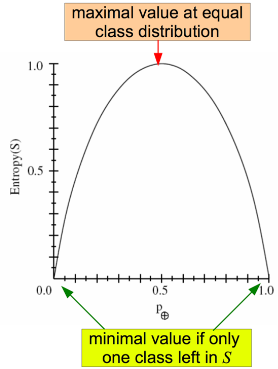
$$E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_K \log_2 p_K = -\sum_{k=1}^{K} p_k \log_2 p_k$$



- $S$ is a set of examples
- $p_+$ is the proportion of examples in positive class
- $p_- = 1 - p_+$ is the proportion of examples in negative class
- Entropy

$$E(S) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_-$$

- Amount of un-orderliness in the class distribution of $S$



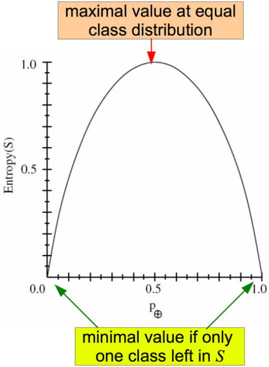

- $p_i$ is the proportion of examples in $S$ from the i-th class

$$E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_K \log_2 p_K = -\sum_{k=1}^{K} p_k \log_2 p_k$$

- $S$ is a set of examples
- $p_+$ is the proportion of examples in positive class
- $p_- = 1 - p_+$ is the proportion of examples in negative class
- Entropy

$$E(S) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_-$$

- Amount of un-orderliness in the class distribution of $S$
- $p_i$ is the proportion of examples in $S$ from the i-th class




$$E(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \ldots - p_K \log_2 p_K = -\sum_{k=1}^{K} p_k \log_2 p_k$$

- **Outlook** = **sunny**: 3 examples **yes**, 2 examples **no**

$$E(Outlook = sunny) = -\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right) = 0.971$$

- **Outlook** = **overcast**: 4 examples **yes**, 0 examples **no**

$$E(Outlook = overcast) = -1\log(1) - 0\log(0) = 0$$

- **Outlook** = **rainy**: 2 examples **yes**, 3 examples **no**

$$E(Outlook = rainy) = -\frac{3}{5}\log\left(\frac{3}{5}\right) - \frac{2}{5}\log\left(\frac{2}{5}\right) = 0.971$$

- **Problem**:
  - Entropy can be applied only to a single (sub-)set of examples
  - Quality of an entire split, corresponding to the attribute $x_j$?
- **Solution**:
  - Average over all sets resulting from the split, weighted by their size

$$I(S, j) = \sum_r \frac{|S_r|}{|S|} \cdot E(S_r)$$

  - **Example**: average entropy for attribute **Outlook**

$$I(Outlook) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.693$$

- **Problem**:
  - Entropy can be applied only to a single (sub-)set of examples
  - Quality of an entire split, corresponding to the attribute $x_j$?
- **Solution**:
  - Average over all sets resulting from the split, weighted by their size

$$I(S, j) = \sum_r \frac{|S_r|}{|S|} \cdot E(S_r)$$

  - **Example**: average entropy for attribute **Outlook**

$$I(Outlook) = \frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971 = 0.693$$

- When an attribute $x_j$ splits the set $S$ into subsets $S_r$
  - we compute the average entropy
  - and compare the sum to the entropy of the original set $S$
- Information Gain for Attribute $x_j$

$$Q(S, j) = E(S) - I(S, j) = E(S) - \sum_r \frac{|S_r|}{|S|} \cdot E(S_r)$$

  - We select an attribute maximizing the difference, i.e. attribute that reduces the un-orderliness most
  - Maximizing IG $\Leftrightarrow$ minimizing average entropy
- $Gain(S, Humidity) = 0.151$; **Gain(S, Outlook)** $= 0.246$; $Gain(S, Wind) = 0.048$; $Gain(S, Temperature) = 0.029$

- When an attribute $x_j$ splits the set $S$ into subsets $S_r$
  - we compute the average entropy
  - and compare the sum to the entropy of the original set $S$
- Information Gain for Attribute $x_j$

$$Q(S,j) = E(S) - I(S,j) = E(S) - \sum_r \frac{|S_r|}{|S|} \cdot E(S_r)$$

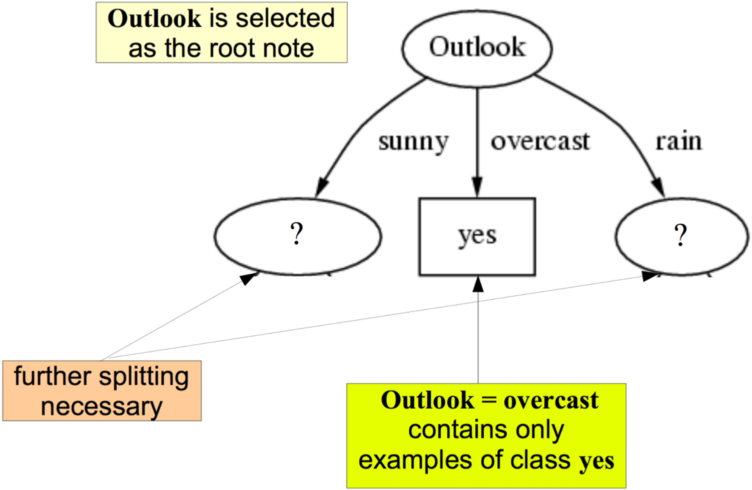
  - We select an attribute maximizing the difference, i.e. attribute that reduces the un-orderliness most
  - Maximizing IG $\Leftrightarrow$ minimizing average entropy
- $Gain(S, Humidity) = 0.151$; **Gain(S, Outlook)** $= 0.246$; $Gain(S, Wind) = 0.048$; $Gain(S, Temperature) = 0.029$

- When an attribute $x_j$ splits the set $S$ into subsets $S_r$
  - we compute the average entropy
  - and compare the sum to the entropy of the original set $S$
- Information Gain for Attribute $x_j$

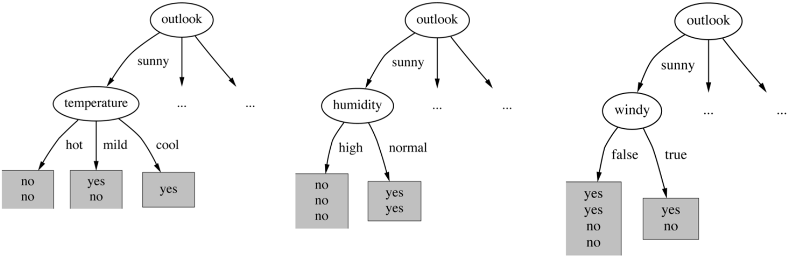
$$Q(S, j) = E(S) - I(S, j) = E(S) - \sum_r \frac{|S_r|}{|S|} \cdot E(S_r)$$

  - We select an attribute maximizing the difference, i.e. attribute that reduces the un-orderliness most
  - Maximizing IG $\Leftrightarrow$ minimizing average entropy
- $Gain(S, Humidity) = 0.151$; **Gain(S, Outlook)** $= 0.246$; $Gain(S, Wind) = 0.048$; $Gain(S, Temperature) = 0.029$

# Example


**Outlook** is selected as the root note
Outlook
sunny
overcast
rain
?
yes
?
further splitting necessary
**Outlook = overcast** contains only examples of class **yes**

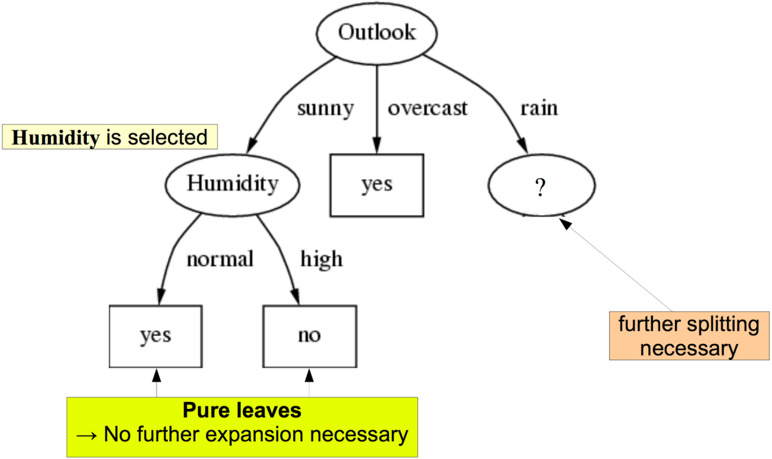
# Example



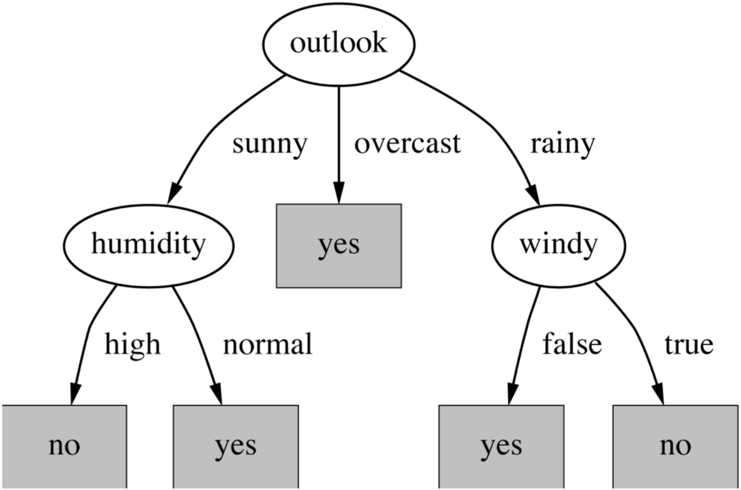

$$\left.\begin{aligned}\text{Gain}(\textit{Temperature}) &= 0.571 \text{ bits}\\ \text{Gain}(\textit{Humidity}) &= 0.971 \text{ bits}\\ \text{Gain}(\textit{Windy}) &= 0.020 \text{ bits}\end{aligned}\right\}$$

**Humidity is selected**

# Example


Outlook
sunny
overcast
rain
Humidity is selected
Humidity
yes
?
normal
high
yes
no
further splitting necessary
Pure leaves
→ No further expansion necessary

# Example: final decision tree



# Formal definition: Loss function $Q(S, j)$ for a binary tree

- $S_t$: the subset of $S$ at step $t$
- With the current split, let $S_l \subseteq S_t$ go left and $S_r \subseteq S_t$ go right
- Choose predicate to optimize

$$Q(S_t, j) = E(S_t) - \frac{|S_l|}{|S_t|} E(S_l) - \frac{|S_r|}{|S_t|} E(S_r) \to \max$$

- $E(S)$: impurity criterion
- Generally

$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} L(y_i, c)$$

- $S_t$: the subset of $S$ at step $t$
- With the current split, let $S_l \subseteq S_t$ go left and $S_r \subseteq S_t$ go right
- Choose predicate to optimize

$$Q(S_t, j) = E(S_t) - \frac{|S_l|}{|S_t|}E(S_l) - \frac{|S_r|}{|S_t|}E(S_r) \to \max$$

- $E(S)$: impurity criterion
- Generally

$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} L(y_i, c)$$

- $S_t$: the subset of $S$ at step $t$
- With the current split, let $S_l \subseteq S_t$ go left and $S_r \subseteq S_t$ go right
- Choose predicate to optimize

$$Q(S_t, j) = E(S_t) - \frac{|S_l|}{|S_t|} E(S_l) - \frac{|S_r|}{|S_t|} E(S_r) \to \max$$

- $E(S)$: impurity criterion
- Generally

$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} L(y_i, c)$$

## Formal definition: Loss function $Q(S, j)$ for a binary tree

- $S_t$: the subset of $S$ at step $t$
- With the current split, let $S_l \subseteq S_t$ go left and $S_r \subseteq S_t$ go right
- Choose predicate to optimize

$$Q(S_t, j) = E(S_t) - \frac{|S_l|}{|S_t|}E(S_l) - \frac{|S_r|}{|S_t|}E(S_r) \to \max$$

- $E(S)$: impurity criterion
- Generally

$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} L(y_i, c)$$

## Formal definition: Loss function $Q(S, j)$ for a binary tree

- $S_t$: the subset of $S$ at step $t$
- With the current split, let $S_l \subseteq S_t$ go left and $S_r \subseteq S_t$ go right
- Choose predicate to optimize

$$Q(S_t, j) = E(S_t) - \frac{|S_l|}{|S_t|} E(S_l) - \frac{|S_r|}{|S_t|} E(S_r) \to \max$$

- $E(S)$: impurity criterion
- Generally

$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} L(y_i, c)$$

- Problematic: attributes with a large number of values
  - Extreme case: each example has its own value
  - E.g. ID; Day attribute in weather data
- Subsets are more likely to be pure if there is a large number of different attribute values
  - Information gain is biased towards choosing attributes with a large number of values
- Problems:
  - Overfitting
  - Data is fragmented into too many small sets

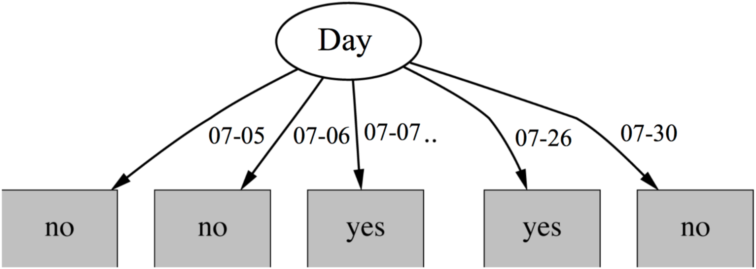
# High-branching attributes in case of general trees

- Problematic: attributes with a large number of values
  - Extreme case: each example has its own value
  - E.g. ID; Day attribute in weather data
- Subsets are more likely to be pure if there is a large number of different attribute values
  - Information gain is biased towards choosing attributes with a large number of values
- Problems:
  - Overfitting
  - Data is fragmented into too many small sets

## High-branching attributes in case of general trees

- Problematic: attributes with a large number of values
  - Extreme case: each example has its own value
  - E.g. ID; Day attribute in weather data
- Subsets are more likely to be pure if there is a large number of different attribute values
  - Information gain is biased towards choosing attributes with a large number of values
- Problems:
  - Overfitting
  - Data is fragmented into too many small sets

# Decision Tree for Day attribute




- Entropy of split

$$I(Day) = \frac{1}{14}\left(E([0,1]) + E([0,1]) + \ldots + E([0,1])\right) = 0$$

- Information gain is maximal for **Day** ($0.940$ bits)

- Intrinsic information of a split, done w.r.t.
- Entropy of distribution of instances into branches
- i.e. how much information do we need to tell which branch as instance belongs to

$$IntI(S,j) = -\sum_r \frac{|S_r|}{|S|} \log_2 \frac{|S_r|}{|S|}$$

- Empirical Observation: attributes with higher intrinsic information are less useful
- Intrinsic information of **Day** attribute

$$IntI(Day) = 14 \times \left(-\frac{1}{14} \cdot \log\left(\frac{1}{14}\right)\right) = 3.807$$

- Intrinsic information of a split, done w.r.t.
- Entropy of distribution of instances into branches
- i.e. how much information do we need to tell which branch as instance belongs to

$$IntI(S, j) = -\sum_{r} \frac{|S_r|}{|S|} \log_2 \frac{|S_r|}{|S|}$$

- Empirical Observation: attributes with higher intrinsic information are less useful
- Intrinsic information of **Day** attribute

$$IntI(Day) = 14 \times \left(-\frac{1}{14} \cdot \log\left(\frac{1}{14}\right)\right) = 3.807$$

- Intrinsic information of a split, done w.r.t.
- Entropy of distribution of instances into branches
- i.e. how much information do we need to tell which branch as instance belongs to

$$IntI(S,j) = -\sum_r \frac{|S_r|}{|S|} \log_2 \frac{|S_r|}{|S|}$$

- Empirical Observation: attributes with higher intrinsic information are less useful
- Intrinsic information of **Day** attribute

$$IntI(Day) = 14 \times \left(-\frac{1}{14} \cdot \log\left(\frac{1}{14}\right)\right) = 3.807$$

- Intrinsic information of a split, done w.r.t.
- Entropy of distribution of instances into branches
- i.e. how much information do we need to tell which branch as instance belongs to

$$IntI(S, j) = -\sum_{r} \frac{|S_r|}{|S|} \log_2 \frac{|S_r|}{|S|}$$

- Empirical Observation: attributes with higher intrinsic information are less useful
- Intrinsic information of **Day** attribute

$$IntI(Day) = 14 \times \left(-\frac{1}{14} \cdot \log\left(\frac{1}{14}\right)\right) = 3.807$$

- Intrinsic information of a split, done w.r.t.
- Entropy of distribution of instances into branches
- i.e. how much information do we need to tell which branch as instance belongs to

$$IntI(S,j) = -\sum_r \frac{|S_r|}{|S|} \log_2 \frac{|S_r|}{|S|}$$

- Empirical Observation: attributes with higher intrinsic information are less useful
- Intrinsic information of **Day** attribute

$$IntI(Day) = 14 \times \left(-\frac{1}{14} \cdot \log\left(\frac{1}{14}\right)\right) = 3.807$$

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio

$$GR(S,j) = \frac{Q(S,j)}{IntI(S,j)}$$

- **Example**: GR of **Day** attribute

$$GR(Day) = \frac{0.940}{3.807} = 0.246$$

- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio

$$GR(S, j) = \frac{Q(S, j)}{IntI(S, j)}$$

- **Example**: GR of **Day** attribute

$$GR(Day) = \frac{0.940}{3.807} = 0.246$$

- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio
$$GR(S,j) = \frac{Q(S,j)}{IntI(S,j)}$$
- **Example**: GR of **Day** attribute
$$GR(Day) = \frac{0.940}{3.807} = 0.246$$
- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio

$$GR(S, j) = \frac{Q(S, j)}{IntI(S, j)}$$

- **Example**: GR of **Day** attribute

$$GR(Day) = \frac{0.940}{3.807} = 0.246$$

- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio

$$GR(S,j) = \frac{Q(S,j)}{IntI(S,j)}$$

- **Example**: GR of **Day** attribute

$$GR(Day) = \frac{0.940}{3.807} = 0.246$$

- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio

$$GR(S,j) = \frac{Q(S,j)}{IntI(S,j)}$$

- **Example**: GR of **Day** attribute

$$GR(Day) = \frac{0.940}{3.807} = 0.246$$

- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Modification of the information gain that reduces its bias towards multi-values attributes
- Takes number and size of branches into account when choosing an attribute
- Gain Ratio
$$GR(S,j) = \frac{Q(S,j)}{IntI(S,j)}$$
- **Example**: GR of **Day** attribute
$$GR(Day) = \frac{0.940}{3.807} = 0.246$$
- $GR(Outlook) = 0.157$; $GR(Humidity) = 0.152$; $GR(Temperature) = 0.019$; $GR(Windy) = 0.049$
- **Day** attribute would still win... $\Rightarrow$ careful analysis!!!
- Anyway, GR is more reliable than IG!!!

- Many alternatives measures to Information Gain
- Most popular alternative: Gini index

$$Gini(S) = \sum_k p_k(1 - p_k) = 1 - \sum_k p_k^2$$

- Average Gini index (instead of average entropy/information)

$$Gini(S, j) = \sum_r \frac{|S_r|}{|S|} \cdot Gini(S_r)$$

- Gini Gain:
  - Can be defined analogously to information gain
  - Typically avg. Gini is minimized instead of maximizing Gini gain

- Many alternatives measures to Information Gain
- Most popular alternative: Gini index

$$Gini(S) = \sum_k p_k(1 - p_k) = 1 - \sum_k p_k^2$$

- Average Gini index (instead of average entropy/information)

$$Gini(S, j) = \sum_r \frac{|S_r|}{|S|} \cdot Gini(S_r)$$

- Gini Gain:
  - Can be defined analogously to information gain
  - Typically avg. Gini is minimized instead of maximizing Gini gain

- Many alternatives measures to Information Gain
- Most popular alternative: Gini index

$$Gini(S) = \sum_k p_k(1 - p_k) = 1 - \sum_k p_k^2$$

- Average Gini index (instead of average entropy/information)

$$Gini(S, j) = \sum_r \frac{|S_r|}{|S|} \cdot Gini(S_r)$$

- Gini Gain:
  - Can be defined analogously to information gain
  - Typically avg. Gini is minimized instead of maximizing Gini gain

- Many alternatives measures to Information Gain
- Most popular alternative: Gini index

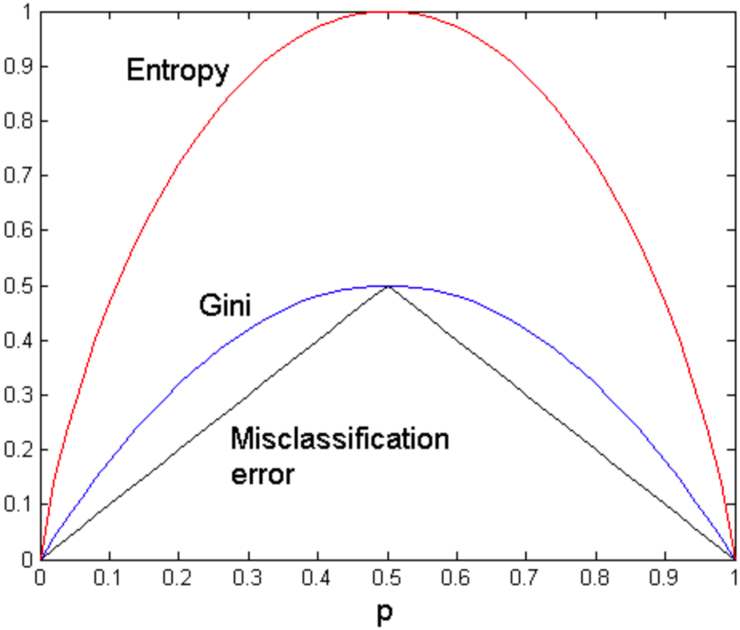
$$Gini(S) = \sum_k p_k(1 - p_k) = 1 - \sum_k p_k^2$$

- Average Gini index (instead of average entropy/information)

$$Gini(S, j) = \sum_r \frac{|S_r|}{|S|} \cdot Gini(S_r)$$

- Gini Gain:
  - Can be defined analogously to information gain
  - Typically avg. Gini is minimized instead of maximizing Gini gain

## Comparison among Splitting Criteria: binary case



- **Regression:**
  - Impurity
  $$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} (y_i - c)^2$$
  - Sum of squared residuals minimized by
  $$c = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} y_i$$
  - Impurity $\equiv$ variance of the target
- **Classification:**
  - Let (share of $y_i$'s equal to $k$)
  $$p_k = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i = k]$$
  - Miss rate:
  $$E(S) = \min_{c \in \mathbb{Y}} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i \neq c]$$

  Impurity criterion is defined as

  Minimizing miss rate $\quad 1 - E(S)$

- **Regression:**
  - Impurity
  $$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} (y_i - c)^2$$
  - Sum of squared residuals minimized by
  $$c = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} y_i$$
  - Impurity $\equiv$ variance of the target
- **Classification:**
  - Let (share of $y_i$'s equal to $k$)
  $$p_k = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i = k]$$
  - Miss rate:
  $$E(S) = \min_{c \in \mathbb{Y}} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i \neq c]$$

  Impurity criterion is defined as

  Minimizing miss rate $\quad 1 - E(S)$

- **Regression:**
  - Impurity
  $$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} (y_i - c)^2$$
  - Sum of squared residuals minimized by
  $$c = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} y_i$$
  - Impurity $\equiv$ variance of the target
- **Classification:**
  - Let (share of $y_i$'s equal to $k$)
  $$p_k = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i = k]$$
  - Miss rate:
  $$E(S) = \min_{c \in \mathbb{Y}} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i \neq c]$$

  Impurity criterion is defined as

  Minimizing miss rate $\quad 1 - E(S)$

- **Regression:**
  - Impurity
$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} (y_i - c)^2$$
  - Sum of squared residuals minimized by
$$c = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} y_i$$
  - Impurity $\equiv$ variance of the target
- **Classification:**
  - Let (share of $y_i$'s equal to $k$)
$$p_k = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i = k]$$
  - Miss rate:
$$E(S) = \min_{c \in \mathbb{Y}} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i \neq c]$$
Impurity criterion is defined as

Minimizing miss rate $\quad 1 - E(S)$

- **Regression:**
  - Impurity
$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} (y_i - c)^2$$
  - Sum of squared residuals minimized by
$$c = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} y_i$$
  - Impurity $\equiv$ variance of the target
- **Classification:**
  - Let (share of $y_i$'s equal to $k$)
$$p_k = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i = k]$$
  - Miss rate:
$$E(S) = \min_{c \in \mathbb{Y}} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i \neq c]$$

Impurity criterion is defined as

Minimizing miss rate $\quad 1 - E(S)$

- **Regression:**
  - Impurity
$$E(S) = \min_{c \in Y} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} (y_i - c)^2$$
  - Sum of squared residuals minimized by
$$c = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} y_i$$
  - Impurity $\equiv$ variance of the target
- **Classification:**
  - Let (share of $y_i$'s equal to $k$)
$$p_k = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i = k]$$
  - Miss rate:
$$E(S) = \min_{c \in \mathbb{Y}} \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} [y_i \neq c]$$

  Impurity criterion is defined as

  Minimizing miss rate $\quad 1 - E(S)$

- Permit numeric attributes
- Allow missing values
- Be robust in the presence of noise
- Be able to approximate arbitrary concept description
- Computationally efficient

As a result C4.5 has been developed!

- Standard method: binary splits: E.g. temperature $< 20$
- Unlike nominal attributes, there are many possible split points
- Selection of "best" split points computationally is more demanding
- Split points can be placed between values or directly at values
- Sort values before performing scan!

- Splitting (multi-way) on a nominal attribute exhausts all information in that attributed
  - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes
  - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read
- Remedy:
  - Pre-discretize numeric attributed, or
  - Multi-way splits instead of binary ones

- If an attribute with a missing value needs to be tested:
  - split the instance into fractional instances (pieces)
  - one piece for each outgoing branch of the node
  - a piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
- Info gain or gain ratio work with fractional instances
  - use sums of weights instead of counts
- During classification split the instance in the same way
  - Merge probability distribution using weights of fractional instances

- If an attribute with a missing value needs to be tested:
  - split the instance into fractional instances (pieces)
  - one piece for each outgoing branch of the node
  - a piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
- Info gain or gain ratio work with fractional instances
  - use sums of weights instead of counts
- During classification split the instance in the same way
  - Merge probability distribution using weights of fractional instances

# Missing values

- If an attribute with a missing value needs to be tested:
  - split the instance into fractional instances (pieces)
  - one piece for each outgoing branch of the node
  - a piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
- Info gain or gain ratio work with fractional instances
  - use sums of weights instead of counts
- During classification split the instance in the same way
  - Merge probability distribution using weights of fractional instances

- The smaller the complexity of a concept, usually the better its generalization ability is
- We need to try to keep the learned concepts simple
- **Pre-pruning**:
  — Stop growing a branch when information becomes unreliable
- **Post-pruning**
  — Grow a decision tree that correctly classifies all training data
  — Simplify it later by replacing some nodes with leafs
- Post-pruning is preferred in practice, pre-pruning can "stop early"

- The smaller the complexity of a concept, usually the better its generalization ability is
- We need to try to keep the learned concepts simple
- **Pre-pruning**:
  - Stop growing a branch when information becomes unreliable
- **Post-pruning**
  - Grow a decision tree that correctly classifies all training data
  - Simplify it later by replacing some nodes with leafs
- Post-pruning is preferred in practice, pre-pruning can "stop early"

- The smaller the complexity of a concept, usually the better its generalization ability is
- We need to try to keep the learned concepts simple
- **Pre-pruning**:
  - Stop growing a branch when information becomes unreliable
- **Post-pruning**
  - Grow a decision tree that correctly classifies all training data
  - Simplify it later by replacing some nodes with leafs
- Post-pruning is preferred in practice, pre-pruning can "stop early"

- The smaller the complexity of a concept, usually the better its generalization ability is
- We need to try to keep the learned concepts simple
- **Pre-pruning**:
  - Stop growing a branch when information becomes unreliable
- **Post-pruning**
  - Grow a decision tree that correctly classifies all training data
  - Simplify it later by replacing some nodes with leafs
- Post-pruning is preferred in practice, pre-pruning can “stop early”

- Based on statistical significant test
  - Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- Most popular test: chi-square test
- ID3 uses chi-square test in addition to information gain
  - Only statistically significant attributes are allowed to be selected by information gain procedure
- Pre-pruning is faster than post-pruning. However, the process may stop too early

- Based on statistical significant test
  - Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- Most popular test: chi-square test
- ID3 uses chi-square test in addition to information gain
  - Only statistically significant attributes are allowed to be selected by information gain procedure
- Pre-pruning is faster than post-pruning. However, the process may stop too early

- Based on statistical significant test
  - Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- Most popular test: chi-square test
- ID3 uses chi-square test in addition to information gain
  - Only statistically significant attributes are allowed to be selected by information gain procedure
- Pre-pruning is faster than post-pruning. However, the process may stop too early

# Pre-pruning

- Based on statistical significant test
  - Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- Most popular test: chi-square test
- ID3 uses chi-square test in addition to information gain
  - Only statistically significant attributes are allowed to be selected by information gain procedure
- Pre-pruning is faster than post-pruning. However, the process may stop too early

# Post-pruning

- Basic idea
  - First grow a full tree to capture all possible attributes interactions
  - Later remove those that are due to chance
  - As long as the performance not decreases try simplification operators
- Two subtree simplification operators
  - Subtree replacement
  - Subtree raising
- Possible performance evaluation strategies
  - Error estimation e.g. on a separate pruning set
  - Significance testing
  - MDL principle

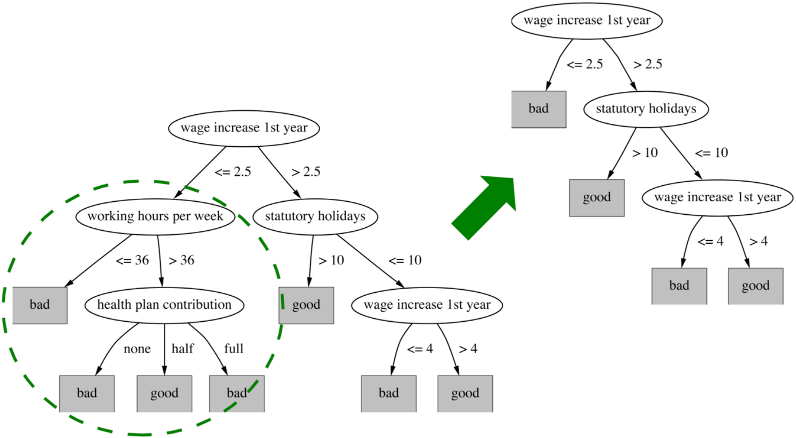
- Basic idea
  - First grow a full tree to capture all possible attributes interactions
  - Later remove those that are due to chance
  - As long as the performance not decreases try simplification operators
- Two subtree simplification operators
  - Subtree replacement
  - Subtree raising
- Possible performance evaluation strategies
  - Error estimation e.g. on a separate pruning set
  - Significance testing
  - MDL principle

# Post-pruning

- Basic idea
  - First grow a full tree to capture all possible attributes interactions
  - Later remove those that are due to chance
  - As long as the performance not decreases try simplification operators
- Two subtree simplification operators
  - Subtree replacement
  - Subtree raising
- Possible performance evaluation strategies
  - Error estimation e.g. on a separate pruning set
  - Significance testing
  - MDL principle

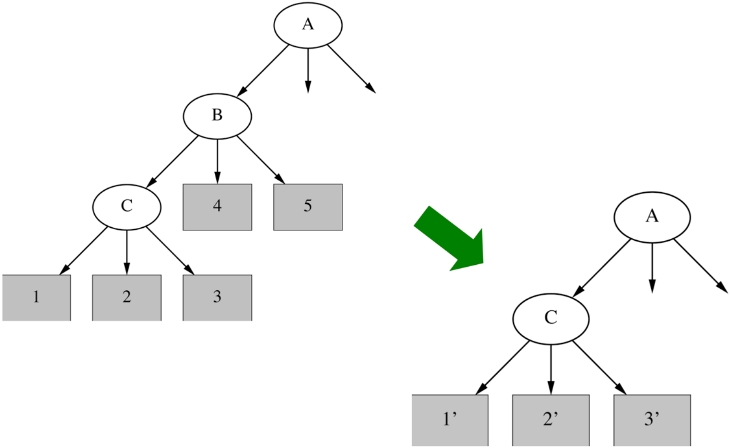
# Subtree replacement

- Bottom-up
- Consider replacing a tree only after all its subtrees



# Subtree raising

- Delete node B
- Redistribute instances of leaves 4 and 5 into C



- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator
- Reduced Error Pruning
  - Use hold-out set for pruning
- C4.5 method
  - Derive confidence interval from training data
  - Assume that the true error is on the upper bound of this confidence interval
- Optimize the accuracy of a decision tree on a separate pruning (validation) set. Prune as long as the error on the validation set does not increase

- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator
- Reduced Error Pruning
  - Use hold-out set for pruning
- C4.5 method
  - Derive confidence interval from training data
  - Assume that the true error is on the upper bound of this confidence interval
- Optimize the accuracy of a decision tree on a separate pruning (validation) set. Prune as long as the error on the validation set does not increase

- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator
- Reduced Error Pruning
  - Use hold-out set for pruning
- C4.5 method
  - Derive confidence interval from training data
  - Assume that the true error is on the upper bound of this confidence interval
- Optimize the accuracy of a decision tree on a separate pruning (validation) set. Prune as long as the error on the validation set does not increase

- Prune only if it does not increase the estimated error
  - Error on the training data is NOT a useful estimator
- Reduced Error Pruning
  - Use hold-out set for pruning
- C4.5 method
  - Derive confidence interval from training data
  - Assume that the true error is on the upper bound of this confidence interval
- Optimize the accuracy of a decision tree on a separate pruning (validation) set. Prune as long as the error on the validation set does not increase

- Significantly impacts learning performance
- Multiple choices available:
  - Maximum tree depth
  - Minimum number of objects in leaf
  - Maximum number of leafs in tree
  - Stop if all objects fall into same leaf
  - Constrain quality improvement
    (stop when improvement gains drop below $s\%$)
- Typically selected via exhaustive search and cross-validation

## Stopping rules for decision tree learning

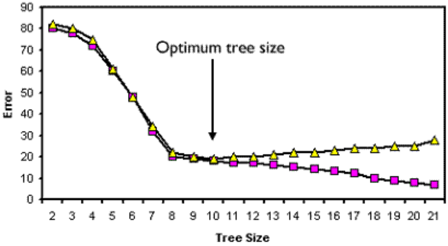
- Significantly impacts learning performance
- Multiple choices available:
  - Maximum tree depth
  - Minimum number of objects in leaf
  - Maximum number of leafs in tree
  - Stop if all objects fall into same leaf
  - Constrain quality improvement
    (stop when improvement gains drop below $s\%$)
- Typically selected via exhaustive search and cross-validation

# Stopping rules for decision tree learning

- Significantly impacts learning performance
- Multiple choices available:
  - Maximum tree depth
  - Minimum number of objects in leaf
  - Maximum number of leafs in tree
  - Stop if all objects fall into same leaf
  - Constrain quality improvement
    (stop when improvement gains drop below $s\%$)
- Typically selected via exhaustive search and cross-validation

## Stopping rules for decision tree learning

- Significantly impacts learning performance
- Multiple choices available:
  - Maximum tree depth
  - Minimum number of objects in leaf
  - Maximum number of leafs in tree
  - Stop if all objects fall into same leaf
  - Constrain quality improvement
    (stop when improvement gains drop below $s\%$)
- Typically selected via exhaustive search and cross-validation




- Learn a large tree (effectively overfit the training set)
- Detect overfitting via $K$-fold cross-validation
- Optimize structure by removing least important nodes

- Assume
  - $d$ attributes
  - $m$ training instances
  - tree depth $O(\log m)$
- Building a tree $O(dm \log(m))$
- Subtree replacement $O(m)$
- Subtree raising $O(m(\log(m))^2)$
- Total cost: $O(dm \log(m)) + O(m(\log(m))^2)$

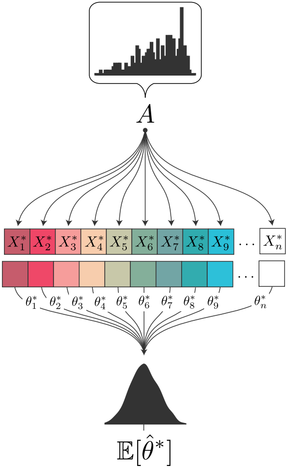
# Complexity of tree induction

- Assume
  - $d$ attributes
  - $m$ training instances
  - tree depth $O(\log m)$
- Building a tree $O(dm \log(m))$
- Subtree replacement $O(m)$
- Subtree raising $O(m(\log(m))^2)$
- Total cost: $O(dm \log(m)) + O(m(\log(m))^2)$

- Assume
  - $d$ attributes
  - $m$ training instances
  - tree depth $O(\log m)$
- Building a tree $O(dm \log(m))$
- Subtree replacement $O(m)$
- Subtree raising $O(m(\log(m))^2)$
- Total cost: $O(dm \log(m)) + O(m(\log(m))^2)$

# The bootstrapping procedure

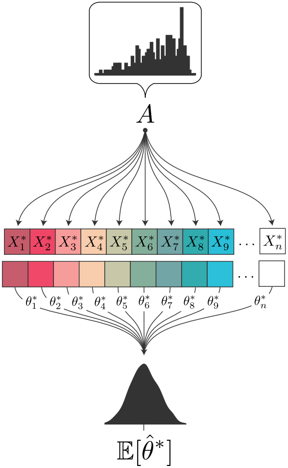
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$




Picture credit: http://www.drbunsen.org/bootstrap-in-picture

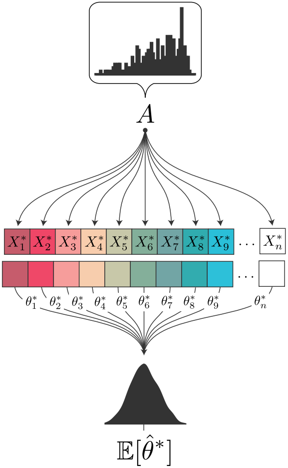
# The bootstrapping procedure

- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
- Bootstrapping: generate new samples $S_m^*$ of $(\mathbf{x}_i, y_i)$ drawn from $S_m$ uniformly at random with replacement (replicated $(\mathbf{x}_i, y_i)$ possible!)




Picture credit: http://www.drbunsen.org/bootstrap-in-picture

- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
- Bootstrapping: generate new samples $S_m^*$ of $(\mathbf{x}_i, y_i)$ drawn from $S_m$ uniformly at random with replacement (replicated $(\mathbf{x}_i, y_i)$ possible!)
- **Ensemble learning idea:**
  1. Generate $N$ bootstrapped samples $S_m^1, \ldots, S_m^N$
  2. Learn $N$ hypotheses $h_1, \ldots, h_N$
  3. Average predictions to obtain $h(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N h_i(\mathbf{x})$
  4. Profit!

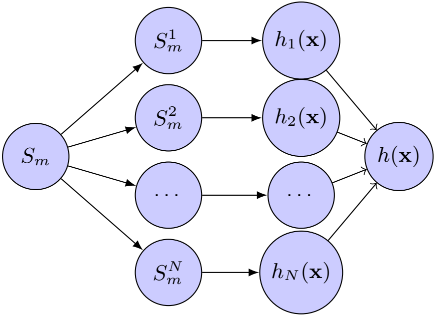



Picture credit: http://www.drbunsen.org/bootstrap-in-picture

# Bagging: bootstrap aggregation

- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
- Weak learners via bootstrapping $h_i(\mathbf{x}) = h_i(\mathbf{x}|S_m^i)$
- Ensemble average

$$h(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} h_i(\mathbf{x}) = $$
$$= \frac{1}{N}\sum_{i=1}^{N} h_i(\mathbf{x}|S_m^i)$$



# The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

## The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
    1. Pick $p$ random features out of $d$
    2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
    3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
    4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

## The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \dots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

# The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \dots, N$:
    1. Pick $p$ random features out of $d$
    2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
    3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
    4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

## The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

# The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

# The Random Forest algorithm

- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \ldots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

# The Random Forest algorithm

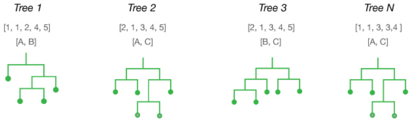
- Bagging over (weak) decision trees
- Reduce error via averaging over instances and features
- Input: a sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in Y$
- The algorithm iterates for $i = 1, \dots, N$:
  1. Pick $p$ random features out of $d$
  2. Bootstrap a sample $S_m^i = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in Y$
  3. Learn a decision tree $h_i(\mathbf{x})$ using bootstrapped $S_m^i$
  4. Stop when leafs in $h_i$ contain less that $n_{\min}$ instances

$\mathbf{x}_i \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$

$S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^5$

Bootstrap $S_m^i, i \in \{1, 2, 3, 4\}$

Learn $\text{Tree}_i(\mathbf{x})$ using $S_m^i$




Picture credit: http://www.thefactmachine.com/random-forests

# Random Forest: synthetic examples