**K-Nearest Neighbor (K-NN)**

- Given training data \( \mathcal{D} = \{ (x_1, y_1), \ldots, (x_N, y_N) \} \) and a test point

- Prediction Rule: Look at the \( K \) most similar training examples

![Diagram of K-Nearest Neighbor (K-NN)]
**K-Nearest Neighbor (K-NN)**

- Given training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and a test point
- Prediction Rule: Look at the $K$ most similar training examples

For classification: assign the majority class label (majority voting)
For regression: assign the average response
Given training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and a test point

Prediction Rule: Look at the $K$ most similar training examples

For classification: assign the majority class label (majority voting)
For regression: assign the average response

The algorithm requires:
- Parameter $K$: number of nearest neighbors to look for
- Distance function: To compute the similarities between examples
Given training data \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \) and a test point

- Prediction Rule: Look at the \( K \) most similar training examples

For classification: assign the majority class label (majority voting)
For regression: assign the average response

The algorithm requires:
- Parameter \( K \): number of nearest neighbors to look for
- Distance function: To compute the similarities between examples

Special Case: 1-Nearest Neighbor
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order

Note: K-Nearest Neighbors is called a non-parametric method.

Unlike other supervised learning algorithms, K-Nearest Neighbors doesn’t learn an explicit mapping $f$ from the training data. It simply uses the training data at the test time to make predictions.
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors

Note: K-Nearest Neighbors is called a non-parametric method. Unlike other supervised learning algorithms, K-Nearest Neighbors doesn’t learn an explicit mapping $f$ from the training data. It simply uses the training data at the test time to make predictions.
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
- Use majority rule (for classification) or averaging (for regression)
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

**Note:** K-Nearest Neighbors is called a non-parametric method
- Unlike other supervised learning algorithms, K-Nearest Neighbors doesn’t learn an explicit mapping $f$ from the training data
**K-Nearest Neighbors Algorithm**

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

**Note:** $K$-Nearest Neighbors is called a *non-parametric* method
- Unlike other supervised learning algorithms, $K$-Nearest Neighbors doesn’t learn an explicit mapping $f$ from the training data
- It simply uses the training data at the test time to make predictions
The K-NN algorithm requires computing distances of the test example from each of the training examples.
K-NN: Computing the distances

- The $K$-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($x_i \in \mathbb{R}^D$): Euclidean distance is commonly used.

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{\|x_i\|^2 + \|x_j\|^2 - 2x_i^T x_j}$$
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($x_i \in \mathbb{R}^D$): Euclidean distance is commonly used

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||x_i||^2 + ||x_j||^2 - 2x_i^T x_j}$$

Generalization of the distance between points in 2 dimensions.
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($x_i \in \mathbb{R}^D$): Euclidean distance is commonly used.

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||x_i||^2 + ||x_j||^2 - 2x_i^T x_j}$$

Generalization of the distance between points in 2 dimensions.

$||x_i|| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$ is called the norm of $x_i$.

Norm of a vector $x$ is also its length.
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($x_i \in \mathbb{R}^D$): Euclidean distance is commonly used.

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||x_i||^2 + ||x_j||^2 - 2x_i^T x_j}$$

Generalization of the distance between points in 2 dimensions.

$||x_i|| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$ is called the norm of $x_i$.

Norm of a vector $x$ is also its length.

$x_i^T x_j = \sum_{m=1}^{D} x_{im} x_{jm}$ is called the dot (or inner) product of $x_i$ and $x_j$.

Dot product measures the similarity between two vectors (orthogonal vectors have dot product=0, parallel vectors have high dot product).
K-NN: Feature Normalization

- Note: Features should be on the same scale

- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
Note: **Features should be on the same scale**

Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize

One way is:
- Replace $x_{im}$ by $z_{im} = \frac{(x_{im}-\bar{x}_m)}{\sigma_m}$ (make them zero mean, unit variance)
Note: Features should be on the same scale

Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize

One way is:
- Replace $x_{im}$ by $z_{im} = \frac{(x_{im} - \bar{x}_m)}{\sigma_m}$ (make them zero mean, unit variance)
- $\bar{x}_m = \frac{1}{N} \sum_{i=1}^{N} x_{im}$: empirical mean of $m^{th}$ feature
- $\sigma_m^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{im} - \bar{x}_m)^2$: empirical variance of $m^{th}$ feature
K-NN: Some other distance measures

- **Binary-valued features**
  - Use Hamming distance: \( d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm}) \)
  - Hamming distance counts the number of features where the two examples disagree

- **Mixed feature types (some real-valued and some binary-valued)?**
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part

- **Can also assign weights to features:**
  \( d(x_i, x_j) = \sum_{m=1}^{D} w_m d(x_{im}, x_{jm}) \)
Suppose the outputs take values from a set $V$

The KNN algorithm is

---

**Training algorithm:**
- For each training example $(x, f(x))$, add the example to the list `training_examples`

**Classification algorithm:**
- Given a query instance $x_q$ to be classified,
  - Let $x_1 \ldots x_k$ denote the $k$ instances from `training_examples` that are nearest to $x_q$
  - Return

\[
\hat{f}(x_q) \leftarrow \arg \max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))
\]

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.
Choice of $K$ - Neighborhood Size

- Small $K$
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfitting

![Diagrams showing decision boundaries for different $K$ values](image-url)
**Choice of $K$ - Neighborhood Size**

- **Small $K$**
  - Creates many small regions for each class
  - May lead to non-smooth decision boundaries and overfit

- **Large $K$**
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
Choice of $K$ - Neighborhood Size

- **Small $K$**
  - Creates many small regions for each class
  - May lead to non-smooth decision boundaries and overfit

- **Large $K$**
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)

- **Choosing $K$**
  - Often data dependent and heuristic based
  - Or using **cross-validation** (using some **held-out data**)
  - In general, a $K$ too small or too big is bad!
K-Nearest Neighbor: Properties

- **What’s nice**
  - Simple and intuitive; easily implementable

- Asymptotically consistent (a theoretical property)
  - With infinite training data and large enough \( K \), \( K \)-NN approaches the best possible classifier (Bayes optimal)

- What’s not so nice...
  - Store all the training data in memory even at test time
  - Can be memory intensive for large training datasets

- Example of non-parametric, or memory/instance-based methods
  - Different from parametric, model-based learning models

- Expensive at test time: \( O(ND) \) computations for each test point
  - Have to search through all training data to find nearest neighbors

- Distance computations with \( N \) training points (\( D \) features each)

- Sensitive to noisy features
  - May perform badly in high dimensions (curse of dimensionality)

- In high dimensions, distance notions can be counter-intuitive!
K-Nearest Neighbor: Properties

What’s nice

- Simple and intuitive; easily implementable
- Asymptotically consistent (a theoretical property)
  - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (Bayes optimal)

What’s not so nice...
- Store all the training data in memory even at test time
  - Can be memory intensive for large training datasets
- Example of non-parametric, or memory/instance-based methods
  - Different from parametric, model-based learning models
  - Expensive at test time: $O(ND)$ computations for each test point
  - Have to search through all training data to find nearest neighbors
  - Distance computations with $N$ training points ($D$ features each)
- Sensitive to noisy features
- May perform badly in high dimensions (curse of dimensionality)
  - In high dimensions, distance notions can be counter-intuitive!
K-Nearest Neighbor: Properties

- **What’s nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)

- **What’s not so nice..**
  - Store all the training data *in memory* even at test time
    - Can be memory intensive for large training datasets
    - An example of **non-parametric**, or **memory/instance-based** methods
    - Different from **parametric**, **model-based** learning models
K-Nearest Neighbor: Properties

- **What’s nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)

- **What’s not so nice..**
  - Store all the training data **in memory** even at test time
    - Can be memory intensive for large training datasets
    - An example of **non-parametric**, or **memory/instance-based** methods
    - Different from **parametric**, **model-based** learning models
  
  - Expensive at test time: $O(ND)$ computations for each test point
    - Have to **search through all training data** to find nearest neighbors
    - Distance computations with $N$ training points ($D$ features each)

- Sensitive to noisy features
- May perform badly in high dimensions (**curse of dimensionality**)
K-Nearest Neighbor: Properties

- **What’s nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)

- **What’s not so nice..**
  - Store all the training data *in memory* even at test time
    - Can be memory intensive for large training datasets
    - An example of **non-parametric**, or **memory/instance-based** methods
    - Different from **parametric**, **model-based** learning models
  
  Expensive at test time: $O(ND)$ computations for each test point
    - Have to search through all training data to find nearest neighbors
    - Distance computations with $N$ training points ($D$ features each)
  
  Sensitive to noisy features
**K-Nearest Neighbor: Properties**

- **What’s nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)

- **What’s not so nice..**
  - Store all the training data *in memory* even at test time
    - Can be memory intensive for large training datasets
    - An example of **non-parametric**, or **memory/instance-based** methods
    - Different from **parametric, model-based** learning models
  - Expensive at test time: $O(ND)$ computations for each test point
    - Have to search through all training data to find nearest neighbors
    - Distance computations with $N$ training points ($D$ features each)
  - Sensitive to noisy features
  - May perform badly in high dimensions (**curse of dimensionality**)
    - In high dimensions, distance notions can be counter-intuitive!
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set \( S = \{x_1, \ldots, x_n\} \) of \( n \) points and a query point \( x_q \), find the \( k \) points from \( S \) that are the \( k \) closest point to \( x_q \) according to a pre-specified distance function.
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set $S = \{x_1, \ldots, x_n\}$ of $n$ points and a query point $x_q$, find the $k$ points from $S$ that are the $k$ closest point to $x_q$ according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification

When $n$ is large and data dimensionality is very high, a linear scan $O(nD)$ may not be ideal. To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice. Approximate nearest neighbor search ($k=1$) gives any query point $x_q$, suppose $x \in S$ is the closest point to $x_q$ for $\epsilon > 0$, approximate nearest neighbor search returns a point $x \in S$ such that $\|x_q - x\| \leq (1 + \epsilon)\|x_q - x_0\|$. Depending on which method is used, running time is sub-linear $O(N(1+\epsilon)^2)$ in number of examples, or even logarithmic in $N$. K-NN and DT
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set \( S = \{x_1, \ldots, x_n\} \) of \( n \) points and a query point \( x_q \), find the \( k \) points from \( S \) that are the \( k \) closest point to \( x_q \) according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification
- When \( n \) is large and data dimensionality is very high a linear scan \( O(ND) \) may not be ideal
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set $S = \{x_1, \ldots, x_n\}$ of $n$ points and a query point $x_q$, find the $k$ points from $S$ that are the $k$ closest point to $x_q$ according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification

- When $n$ is large and data dimensionality is very high a linear scan $O(ND)$ may not be ideal

- To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set $S = \{x_1, \ldots, x_n\}$ of $n$ points and a query point $x_q$, find the $k$ points from $S$ that are the $k$ closest point to $x_q$ according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification

- When $n$ is large and data dimensionality is very high a linear scan $O(ND)$ may not be ideal

- To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice

- Approximate nearest neighbor search ($k=1$)
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set \( S = \{x_1, \ldots, x_n\} \) of \( n \) points and a query point \( x_q \), find the \( k \) points from \( S \) that are the \( k \) closest point to \( x_q \) according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification

- When \( n \) is large and data dimensionality is very high a linear scan \( O(ND) \) may not be ideal

- To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice

- Approximate nearest neighbor search (\( k=1 \))
  - given any query point \( x_q \), suppose \( x^* \in S \) is the closest point to \( x_q \)
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set \( S = \{x_1, \ldots, x_n\} \) of \( n \) points and a query point \( x_q \), find the \( k \) points from \( S \) that are the \( k \) closest point to \( x_q \) according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification

- When \( n \) is large and data dimensionality is very high a linear scan \( O(ND) \) may not be ideal

- To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice

- Approximate nearest neighbor search (\( k=1 \))
  - given any query point \( x_q \), suppose \( x^* \in S \) is the closest point to \( x_q \)
  - for \( \epsilon > 0 \), approximate nearest neighbor search returns a point \( x \in S \) such that
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set $S = \{x_1, \ldots, x_n\}$ of $n$ points and a query point $x_q$, find the $k$ points from $S$ that are the $k$ closest point to $x_q$ according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification
- When $n$ is large and data dimensionality is very high a linear scan $O(ND)$ may not be ideal
- To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice
- Approximate nearest neighbor search ($k=1$)
  - given any query point $x_q$, suppose $x^* \in S$ is the closest point to $x_q$
  - for $\epsilon > 0$, approximate nearest neighbor search returns a point $x \in S$ such that
    $$||x_q - x|| \leq (1 + \epsilon)||x_q - x^*||$$
Unsupervised Version (Nearest Neighbor Search)

- Unsupervised version of this problem is as follows
  - Given a set $S = \{x_1, \ldots, x_n\}$ of $n$ points and a query point $x_q$, find the $k$ points from $S$ that are the $k$ closest point to $x_q$ according to a pre-specified distance function
  - Nearest neighbors needs to be identified before taking the majority vote for supervised classification

- When $n$ is large and data dimensionality is very high a linear scan $O(ND)$ may not be ideal

- To get a fast answer, instead of exact nearest neighbor search, approximate nearest neighbor search are very popular in practice

- Approximate nearest neighbor search ($k=1$)
  - given any query point $x_q$, suppose $x^* \in S$ is the closest point to $x_q$
  - for $\epsilon > 0$, approximate nearest neighbor search returns a point $x \in S$ such that
    \[
    ||x_q - x|| \leq (1 + \epsilon)||x_q - x^*||
    \]
  - depending on which method is used, running time is sub-linear $O \left( \frac{1}{N(1+\epsilon)^2} \right)$ in number of examples, or even logarithmic in $N$
Decision Tree

- Decision tree is a classification technique defined by a *hierarchy* of rules (in form of a tree)
Decision tree is a classification technique defined by a **hierarchy** of rules (in form of a tree).
Decision Tree

- Decision tree is a classification technique defined by a hierarchy of rules (in form of a tree)

- Rules form the internal nodes of the tree (topmost internal node = root)
Decision Tree

- Decision tree is a classification technique defined by a hierarchy of rules (in form of a tree).

- Rules form the **internal nodes** of the tree (topmost internal node = root).

- Each rule (internal node) tests the value of some property the data
Decision tree is a classification technique defined by a hierarchy of rules (in form of a tree)

- Rules form the **internal nodes** of the tree (topmost internal node = root)
- Each rule (internal node) tests the value of some property the data
- **Decision Tree Learning**
Decision Tree

- Decision tree is a classification technique defined by a hierarchy of rules (in form of a tree)

- Rules form the internal nodes of the tree (topmost internal node = root)

- Each rule (internal node) tests the value of some property the data

Decision Tree Learning
- Training data is used to construct the Decision Tree (DT)
Decision Tree

- Decision tree is a classification technique defined by a hierarchy of rules (in form of a tree)

![Decision Tree Diagram]

- Rules form the **internal nodes** of the tree (topmost internal node = root)
- Each rule (internal node) tests the value of some property the data

**Decision Tree Learning**
- Training data is used to construct the Decision Tree (DT)
- The DT is used to predict label $y$ for test input $x$
Decision Tree Learning: Example 1

- Identifying the region (blue or green) a point lies in
  - A classification problem (blue vs green)
  - Each input has 2 features: co-ordinates \( \{x_1, x_2\} \) in the 2D plane
  - Left: Training data, Right: A decision tree constructed using this data

![Diagram of decision tree]

K-NN and DT
Identifying the region (blue or green) a point lies in
- A classification problem (blue vs green)
- Each input has 2 features: co-ordinates \( \{x_1, x_2\} \) in the 2D plane
- Left: Training data, Right: A decision tree constructed using this data

The DT can be used to predict the region (blue/green) of a new test point
- By testing the features of the test point
- In the order defined by the DT (first \( x_2 \) and then \( x_1 \))
Deciding whether to play or not to play Tennis on a Saturday

- A classification problem (play vs no-play)
- Each input has 4 features: Outlook, Temperature, Humidity, Wind
- Left: Training data, Right: A decision tree constructed using this data
Deciding whether to play or not to play Tennis on a Saturday

- A classification problem (play vs no-play)
- Each input has 4 features: Outlook, Temperature, Humidity, Wind
- Left: Training data, Right: A decision tree constructed using this data

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Decision Tree Learning: Example 2

- Deciding whether to play or not to play Tennis on a Saturday
  - A classification problem (play vs no-play)
  - Each input has 4 features: Outlook, Temperature, Humidity, Wind
  - Left: Training data, Right: A decision tree constructed using this data

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Decision Tree Learning: Example 2

- Deciding whether to play or not to play Tennis on a Saturday
  - A classification problem (play vs no-play)
  - Each input has 4 features: Outlook, Temperature, Humidity, Wind
  - Left: Training data, Right: A decision tree constructed using this data

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- The DT can be used to predict play vs no-play for a new Saturday
  - By testing the features of that day
  - In the order defined by the DT
Now let’s look at the Tennis playing example

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Decision Tree Construction

- Now let’s look at the Tennis playing example

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- **Question:** Why does it make more sense to test the feature “outlook” first?
Decision Tree Construction

Now let’s look at the Tennis playing example

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

Question: Why does it make more sense to test the feature “outlook” first?

Answer: Of all the 4 features, it’s most informative
Now let’s look at the Tennis playing example

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

**Question:** Why does it make more sense to test the feature “outlook” first?

**Answer:** Of all the 4 features, it’s most informative

We will see shortly how to quantity the informativeness
Decision Tree Construction

- Now let’s look at the Tennis playing example

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

**Question:** Why does it make more sense to test the feature “outlook” first?

**Answer:** Of all the 4 features, it’s most informative

- We will see shortly how to quantity the informativeness
- **Information content** of a feature decides its position in the DT
Decision Tree Construction

Now let’s look at the Tennis playing example

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

**Question:** Why does it make more sense to test the feature “outlook” first?

**Answer:** Of all the 4 features, it’s most informative

We will see shortly how to quantity the informativeness

**Information content** of a feature decides its position in the DT

**Analogy:** Playing the game 20 Questions (the most useful questions first)
Decision Tree Construction

Summarizing:

- The training data is used to construct the DT
- Each internal node is a rule (testing the value of some feature)
- Highly informative features are placed higher up in the tree
- We need a way to rank features according to their information content
- We will use Entropy and Information Gain as the criteria
- Note: There are several specific versions of the Decision Tree
  - ID3, C4.5, Classification and Regression Trees (CART), etc.
  - We will be looking at the ID3 algorithm
Entropy measures the randomness/uncertainty in the data.
- **Entropy** measures the randomness/uncertainty in the data

- Let’s consider a set $S$ of examples with $C$ many classes. Entropy of this set:

  $$H(S) = - \sum_{c \in C} p_c \log_2 p_c$$

- $p_c$ is the probability that an element of $S$ belongs to class $c$
  - .. basically, the fraction of elements of $S$ belonging to class $c$
Entropy measures the randomness/uncertainty in the data.

Let’s consider a set $S$ of examples with $C$ many classes. Entropy of this set:

$$H(S) = - \sum_{c \in C} p_c \log_2 p_c$$

$p_c$ is the probability that an element of $S$ belongs to class $c$.
.. basically, the fraction of elements of $S$ belonging to class $c$

Intuition: Entropy is a measure of the “degree of surprise”

- Some dominant classes $\implies$ small entropy (less uncertainty)
- Equiprobable classes $\implies$ high entropy (more uncertainty)
Entropy measures the randomness/uncertainty in the data.

Let’s consider a set $S$ of examples with $C$ many classes. Entropy of this set:

$$H(S) = - \sum_{c \in C} p_c \log_2 p_c$$

$p_c$ is the probability that an element of $S$ belongs to class $c$.

Intuition: Entropy is a measure of the “degree of surprise”

- Some dominant classes $\implies$ small entropy (less uncertainty)
- Equiprobable classes $\implies$ high entropy (more uncertainty)

Entropy denotes the average number of bits needed to encode $S$. 
Let’s assume each element of \( S \) consists of a set of features.

**Information Gain** (IG) on a feature \( F \)

\[
IG(S, F) = H(S) - \sum_{f \in F} \frac{|S_f|}{|S|} H(S_f)
\]

- \( S_f \) number of elements of \( S \) with feature \( F \) having value \( f \)

- \( IG(S, F) \) measures the increase in our certainty about \( S \) once we know the value of \( F \)
Information Gain

Let’s assume each element of \( S \) consists of a set of features

**Information Gain (IG) on a feature \( F \)**

\[
IG(S, F) = H(S) - \sum_{f \in F} \frac{|S_f|}{|S|} H(S_f)
\]

- \( S_f \) number of elements of \( S \) with feature \( F \) having value \( f \)

- \( IG(S, F) \) measures the increase in our certainty about \( S \) once we know the value of \( F \)

- \( IG(S, F) \) denotes the number of bits saved while encoding \( S \) once we know the value of the feature \( F \)
Computing Information Gain

- Let's begin with the root node of the DT and compute IG of each feature.
- Consider feature “wind” ∈ {weak, strong} and its IG w.r.t. the root node.

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Computing Information Gain

- Let's begin with the root node of the DT and compute $IG$ of each feature
- Consider feature "wind" $\in \{\text{weak, strong}\}$ and its $IG$ w.r.t. the root node
- Root node: $S = [9+, 5-]$ (all training data: 9 play, 5 no-play)
- Entropy: $H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94$
Let’s begin with the root node of the DT and compute $IG$ of each feature. Consider feature “wind” $\in \{\text{weak, strong}\}$ and its $IG$ w.r.t. the root node.

**Root node:** $S = [9+, 5-]$ (all training data: 9 play, 5 no-play)

**Entropy:** $H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94$

**$S_{\text{weak}} = [6+, 2-] \implies H(S_{\text{weak}}) = 0.811$**
Computing Information Gain

- Let's begin with the root node of the DT and compute $IG$ of each feature.
- Consider feature "wind" $\in \{\text{weak, strong}\}$ and its $IG$ w.r.t. the root node.
- Root node: $S = [9+, 5-]$ (all training data: 9 play, 5 no-play).
- Entropy: $H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94$
- $S_{weak} = [6+, 2-] \implies H(S_{weak}) = 0.811$
- $S_{strong} = [3+, 3-] \implies H(S_{strong}) = 1$
Computing Information Gain

- Let’s begin with the root node of the DT and compute $IG$ of each feature.
- Consider feature “wind” ∈ \{weak, strong\} and its $IG$ w.r.t. the root node.

Root node: $S = [9+, 5–]$ (all training data: 9 play, 5 no-play)
- Entropy: $H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94$
- $S_{weak} = [6+, 2–] \implies H(S_{weak}) = 0.811$
- $S_{strong} = [3+, 3–] \implies H(S_{strong}) = 1$

$$\begin{align*}
IG(S, \text{wind}) &= H(S) - \frac{|S_{weak}|}{|S|} H(S_{weak}) - \frac{|S_{strong}|}{|S|} H(S_{strong}) \\
&= 0.94 - \frac{8}{14} \times 0.811 - \frac{6}{14} \times 1 \\
&= 0.048
\end{align*}$$
Choosing the most informative feature

- At the root node, the information gains are:
  - $IG(S, \text{wind}) = 0.048$ (we already saw)
  - $IG(S, \text{outlook}) = 0.246$
  - $IG(S, \text{humidity}) = 0.151$
  - $IG(S, \text{temperature}) = 0.029$

- “outlook” has the maximum $IG$ $\implies$ chosen as the root node
Choosing the most informative feature

At the root node, the information gains are:

- $IG(S, \text{wind}) = 0.048$ (we already saw)
- $IG(S, \text{outlook}) = 0.246$
- $IG(S, \text{humidity}) = 0.151$
- $IG(S, \text{temperature}) = 0.029$

“outlook” has the maximum $IG \implies$ chosen as the root node

Growing the tree:

- Iteratively select the feature with the highest information gain for each child of the previous node
Choosing the most informative feature

- At the root node, the information gains are:
  - $IG(S, \text{wind}) = 0.048$ (we already saw)
  - $IG(S, \text{outlook}) = 0.246$
  - $IG(S, \text{humidity}) = 0.151$
  - $IG(S, \text{temperature}) = 0.029$

- “outlook” has the maximum $IG \Rightarrow$ chosen as the root node

Growing the tree:
- Iteratively select the feature with the highest information gain for each child of the previous node
Growing The Tree

- How to decide which feature to choose as we descend the tree?

Rule:
Iterate - for each child node, select the feature with the highest IG.

For level-2, left node: 
$S = \{2, 3\}$ (days 1, 2, 8, 9, 11)

Let's compute the Information Gain for each feature (except outlook).
The feature with the highest Information Gain should be chosen for this node.
Growing The Tree

How to decide which feature to choose as we descend the tree?

**Rule:** Iterate - for each child node, select the feature with the highest IG

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Growing The Tree

- How to decide which feature to choose as we descend the tree?
- **Rule:** Iterate - for each child node, select the feature with the highest IG

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

Which attribute should be tested here?
Growing The Tree

- How to decide which feature to choose as we descend the tree?
- **Rule:** Iterate - for each child node, select the feature with the highest IG

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- For level-2, left node: \( S = [2+, 3-] \) (days 1,2,8,9,11)
- Let’s compute the Information Gain for each feature (except outlook)
Growing The Tree

- How to decide which feature to choose as we descend the tree?
- **Rule:** Iterate - for each child node, select the feature with the highest IG

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- For level-2, left node: $S = [2+, 3−]$ (days 1, 2, 8, 9, 11)
- Let’s compute the Information Gain for each feature (except outlook)
- The feature with the highest Information Gain should be chosen for this node
Growing The Tree

For this node \( S = [2+; 3] \), the IG for the feature temperature:

\[
IG(S; \text{temperature}) = H(S) \sum_v \frac{S_v}{S} \sum_j f_{\text{hot, mild, cool}} g_j S_j H(S_v) = 0:
\]

\[
S_{\text{hot}} = [0+; 2] = \log_2(2) = 0
\]

\[
S_{\text{mild}} = [1+; 1] = \log_2(1) = 1
\]

\[
S_{\text{cool}} = [1+; 0] = \log_2(0) = 0
\]

\[
IG(S; \text{temperature}) = 0:
\]

Likewise we can compute:

\[
IG(S; \text{humidity}) = 0:
\]

\[
IG(S; \text{wind}) = 0:
\]

Therefore, we choose \( \text{humidity} \) (with highest \( IG = 0:970 \)) for the level-2 left node.

K-NN and DT
For this node \((S = [2+, 3-])\), the \(IG\) for the feature \textit{temperature}:

\[
IG(S, \text{temperature}) = H(S) - \sum_{v \in \{\text{hot, mild, cool}\}} \frac{|S_v|}{|S|} H(S_v)
\]
For this node \((S = [2+, 3–])\), the IG for the feature \(\text{temperature}\):

\[
IG(S, \text{temperature}) = \sum_{S_v \in \{\text{hot, mild, cool}\}} \frac{|S_v|}{|S|} H(S_v)
\]

\(S = [2+, 3–] \implies H(S) = -(2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.971\)
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

For this node \( S = [2+, 3-] \), the IG for the feature \textit{temperature}:

\[
IG(S, \text{temperature}) = H(S) - \sum_{v \in \{\text{hot, mild, cool}\}} \frac{|S_v|}{|S|} H(S_v)
\]

- \( S = [2+, 3-] \Rightarrow H(S) = -(2/5) \cdot \log_2(2/5) - (3/5) \cdot \log_2(3/5) = 0.971 \)
- \( S_{\text{hot}} = [0+, 2-] \Rightarrow H(S_{\text{hot}}) = -0 \cdot \log_2(0) - (2/2) \cdot \log_2(2/2) = 0 \)
Growing The Tree

For this node \((S = [2+, 3-])\), the \(IG\) for the feature \textit{temperature}:

\[
IG(S, \text{temperature}) = H(S) - \sum_{v \in \{\text{hot, mild, cool}\}} \frac{|S_v|}{|S|} H(S_v)
\]

- \(S = [2+, 3-] \implies H(S) = -(2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.971\)
- \(S_{\text{hot}} = [0+, 2-] \implies H(S_{\text{hot}}) = -0 \times \log_2(0) - (2/2) \times \log_2(2/2) = 0\)
- \(S_{\text{mild}} = [1+, 1-] \implies H(S_{\text{mild}}) = -(1/2) \times \log_2(1/2) - (1/2) \times \log_2(1/2) = 1\)
Growing The Tree

For this node \((S = [2+, 3-])\), the \(\text{IG}\) for the feature \(\text{temperature}\):

\[
\text{IG}(S, \text{temperature}) = H(S) - \sum_{v \in \{\text{hot}, \text{mild}, \text{cool}\}} \frac{|S_v|}{|S|} H(S_v)
\]

- \(S = [2+, 3-] \implies H(S) = -(2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.971\)
- \(S_{\text{hot}} = [0+, 2-] \implies H(S_{\text{hot}}) = -0 \times \log_2(0) - (2/2) \times \log_2(2/2) = 0\)
- \(S_{\text{mild}} = [1+, 1-] \implies H(S_{\text{mild}}) = -(1/2) \times \log_2(1/2) - (1/2) \times \log_2(1/2) = 1\)
- \(S_{\text{cool}} = [1+, 0-] \implies H(S_{\text{cool}}) = -(1/1) \times \log_2(1/1) - (0/1) \times \log_2(0/1) = 0\)
Growing The Tree

For this node \((S = [2+, 3-])\), the \(IG\) for the feature \(temperature\):

\[
IG(S, \text{temperature}) = H(S) - \sum_{v \in \{\text{hot, mild, cool}\}} \frac{|S_v|}{|S|} H(S_v)
\]

- \(S = [2+, 3-] \implies H(S) = -(2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5) = 0.971\)
- \(S_{\text{hot}} = [0+, 2-] \implies H(S_{\text{hot}}) = -0 \times \log_2(0) - (2/2) \times \log_2(2/2) = 0\)
- \(S_{\text{mild}} = [1+, 1-] \implies H(S_{\text{mild}}) = -(1/2) \times \log_2(1/2) - (1/2) \times \log_2(1/2) = 1\)
- \(S_{\text{cool}} = [1+, 0-] \implies H(S_{\text{cool}}) = -(1/1) \times \log_2(1/1) - (0/1) \times \log_2(0/1) = 0\)
- \(IG(S, \text{temperature}) = 0.971 - 2/5 \times 0 - 2/5 \times 1 - 1/5 \times 0 = 0.570\)
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- For this node \((S = [2+, 3-])\), the \(IG\) for the feature \(temperature\):
  \[
  IG(S, temperature) = H(S) - \sum_{v \in \{hot, mild, cool\}} \frac{|S_v|}{|S|} H(S_v)
  \]

- \(S = [2+, 3-] \implies H(S) = -(2/5) \cdot \log_2(2/5) - (3/5) \cdot \log_2(3/5) = 0.971\)
- \(S_{hot} = [0+, 2-] \implies H(S_{hot}) = -0 \cdot \log_2(0) - (2/2) \cdot \log_2(2/2) = 0\)
- \(S_{mild} = [1+, 1-] \implies H(S_{mild}) = -(1/2) \cdot \log_2(1/2) - (1/2) \cdot \log_2(1/2) = 1\)
- \(S_{cool} = [1+, 0-] \implies H(S_{cool}) = -(1/1) \cdot \log_2(1/1) - (0/1) \cdot \log_2(0/1) = 0\)
- \(IG(S, temperature) = 0.971 - 2/5 \cdot 0 - 2/5 \cdot 1 - 1/5 \cdot 0 = 0.570\)
- Likewise we can compute: \(IG(S, humidity) = 0.970\), \(IG(S, wind) = 0.019\)

Which attribute should be tested here?
For this node \((S = [2+, 3-])\), the \(IG\) for the feature \(temperature\):

\[
IG(S, temperature) = H(S) - \sum_{v \in \{hot, mild, cool\}} \frac{|S_v|}{|S|} H(S_v)
\]

- \(S = [2+, 3-] \implies H(S) = -(2/5) \cdot \log_2(2/5) - (3/5) \cdot \log_2(3/5) = 0.971\)
- \(S_{hot} = [0+, 2-] \implies H(S_{hot}) = -0 \cdot \log_2(0) - (2/2) \cdot \log_2(2/2) = 0\)
- \(S_{mild} = [1+, 1-] \implies H(S_{mild}) = -(1/2) \cdot \log_2(1/2) - (1/2) \cdot \log_2(1/2) = 1\)
- \(S_{cool} = [1+, 0-] \implies H(S_{cool}) = -(1/1) \cdot \log_2(1/1) - (0/1) \cdot \log_2(0/1) = 0\)
- \(IG(S, temperature) = 0.971 - 2/5 \times 0 - 2/5 \times 1 - 1/5 \times 0 = 0.570\)
- Likewise we can compute: \(IG(S, humidity) = 0.970\), \(IG(S, wind) = 0.019\)
- Therefore, we choose “humidity” (with highest \(IG = 0.970\)) for the level-2 left node
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

Level-2, middle node: no need to grow (already a leaf)

Level-2, right node: repeat the same exercise!

Compute IG for each feature (except outlook)

Exercise: Verify that wind has the highest IG for this node

Level-2 expansion gives us the following tree:

K-NN and DT
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- Level-2, middle node: no need to grow (already a leaf)

Compute IG for each feature (except outlook)

Exercise: Verify that wind has the highest IG for this node

Level-2 expansion gives us the following tree:

- Outlook
  - Sunny
  - Overcast
  - Rain

- Humidity
  - Yes
  - ?

Which attribute should be tested here?
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- Level-2, middle node: no need to grow (already a leaf)
- Level-2, right node: repeat the same exercise!
Growing The Tree

- Level-2, middle node: no need to grow (already a leaf)
- Level-2, right node: repeat the same exercise!
  - Compute $IG$ for each feature (except outlook)
Growing The Tree

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>

- **Level-2, middle node:** no need to grow (already a leaf)
- **Level-2, right node:** repeat the same exercise!
  - Compute $IG$ for each feature (except outlook)
  - **Exercise:** Verify that wind has the highest $IG$ for this node

K-NN and DT
Growing The Tree

- Level-2, middle node: no need to grow (already a leaf)
- Level-2, right node: repeat the same exercise!
  - Compute $IG$ for each feature (except outlook)
  - Exercise: Verify that wind has the highest $IG$ for this node
- Level-2 expansion gives us the following tree:
### Growing The Tree: Stopping Criteria

When expanding a node further, stop if:

- The examples at the node have the same label.
- There are no more features to test.

---

<table>
<thead>
<tr>
<th>day</th>
<th>outlook</th>
<th>temperature</th>
<th>humidity</th>
<th>wind</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>weak</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>weak</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
<td>high</td>
<td>strong</td>
<td>no</td>
</tr>
</tbody>
</table>
Growing The Tree: Stopping Criteria

Stop expanding a node further when:
- It consists of examples all having the same label
- We run out of features to test!
Growing The Tree: Stopping Criteria

- Stop expanding a node further when:
  - It consists of examples all having the same label
  - We run out of features to test!
Growing The Tree: Stopping Criteria

Stop expanding a node further when:

- It consist of examples all having the same label
Growing The Tree: Stopping Criteria

- Stop expanding a node further when:
  - It consist of examples all having the same label
  - Or we run out of features to test!
A recursive algorithm:

\[ DT(Examples, Labels, Features) : \]

- If all examples are positive, return a single node tree `Root` with label `+`.
- If all examples are negative, return a single node tree `Root` with label `-`.
- If all features exhausted, return a single node tree `Root` with majority label.
- Otherwise, let `F` be the feature having the highest information gain.
- For each possible value `f` of `F`:
  - Add a tree branch below `Root` corresponding to the test `F = f`.
  - Let `Examples_f` be the set of examples with feature `F` having value `f`.
  - Let `Labels_f` be the corresponding labels.
  - If `Examples_f` is empty, add a leaf node below this branch with label the most common label in `Examples`.
  - Otherwise, add the following subtree below this branch:
    \[ DT(Examples_f, Labels_f, Features - \{F\}) \]

Note: `Features - \{F\}` removes feature `F` from the feature set.
A recursive algorithm:

\[
\text{DT}(\text{Examples}, \text{Labels}, \text{Features}):\n\]

- If all examples are positive, return a single node tree \textit{Root} with label \( = + \)
A recursive algorithm:
\[ \text{DT}(\text{Examples, Labels, Features}): \]
- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
A recursive algorithm:

\[ \text{DT}(\text{Examples}, \text{Labels}, \text{Features}) : \]

- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
Decision Tree Algorithm

A recursive algorithm:

\( \text{DT}(\text{Examples, Labels, Features}): \)

- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
- Otherwise, let \( F \) be the feature having the highest information gain
- \( \text{Root} \leftarrow F \)
A recursive algorithm:

\[ \text{DT}(\text{Examples}, \text{Labels}, \text{Features}): \]
- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
- Otherwise, let \( F \) be the feature having the highest information gain
  \( \text{Root} \leftarrow F \)
  For each possible value \( f \) of \( F \)
Decision Tree Algorithm

**A recursive algorithm:**

\[ \text{DT} (\text{Examples, Labels, Features}) : \]

- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
- Otherwise, let \( F \) be the feature having the highest information gain
- \( \text{Root} \leftarrow F \)
- For each possible value \( f \) of \( F \)
  - Add a tree branch below \( \text{Root} \) corresponding to the test \( F = f \)
Decision Tree Algorithm

A recursive algorithm:

\[ \text{DT}(\text{Examples}, \text{Labels}, \text{Features}): \]

- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
- Otherwise, let \( F \) be the feature having the highest information gain
- \( \text{Root} \leftarrow F \)
- For each possible value \( f \) of \( F \)
  - Add a tree branch below \( \text{Root} \) corresponding to the test \( F = f \)
  - Let \( \text{Examples}_f \) be the set of examples with feature \( F \) having value \( f \)
  - Let \( \text{Labels}_f \) be the corresponding labels
A recursive algorithm:

\( \text{DT}(\text{Examples, Labels, Features}) : \)

- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
- Otherwise, let \( F \) be the feature having the highest information gain
- \( \text{Root} \leftarrow F \)
- For each possible value \( f \) of \( F \)
  - Add a tree branch below \( \text{Root} \) corresponding to the test \( F = f \)
  - Let \( \text{Examples}_f \) be the set of examples with feature \( F \) having value \( f \)
  - Let \( \text{Labels}_f \) be the corresponding labels
  - If \( \text{Examples}_f \) is empty, add a leaf node below this branch with label \( = \) most common label in \( \text{Examples} \)
Decision Tree Algorithm

A recursive algorithm:

\[ \text{DT}(\text{Examples, Labels, Features}): \]

- If all examples are positive, return a single node tree \( \text{Root} \) with label \( = + \)
- If all examples are negative, return a single node tree \( \text{Root} \) with label \( = - \)
- If all features exhausted, return a single node tree \( \text{Root} \) with majority label
- Otherwise, let \( F \) be the feature having the highest information gain
- \( \text{Root} \leftarrow F \)
- For each possible value \( f \) of \( F \)
  - Add a tree branch below \( \text{Root} \) corresponding to the test \( F = f \)
  - Let \( \text{Examples}_f \) be the set of examples with feature \( F \) having value \( f \)
  - Let \( \text{Labels}_f \) be the corresponding labels
  - If \( \text{Examples}_f \) is empty, add a leaf node below this branch with label \( = \) most common label in \( \text{Examples} \)
  - Otherwise, add the following subtree below this branch:
    \[ \text{DT}(\text{Examples}_f, \text{Labels}_f, \text{Features} - \{F\}) \]
  - Note: \( \text{Features} - \{F\} \) removes feature \( F \) from the feature set \( \text{Features} \)
Overfitting in Decision Trees

- What if we added a noisy example in our Tennis Playing dataset?
- Outlook=Sunny, Temperature=Hot, Humidity=Normal, Wind=Strong, Play=No
What if we added a noisy example in our Tennis Playing dataset?

Outlook=Sunny, Temperature=Hot, Humidity=Normal, Wind=Strong, \text{Play}=\text{No}

This \text{Play}=\text{No} example would be grouped with the node D9, D11 (both \text{Play}=\text{Yes})
What if we added a noisy example in our Tennis Playing dataset?

Outlook=Sunny, Temperature=Hot, Humidity=Normal, Wind=Strong, **Play=No**

This **Play=No** example would be grouped with the node D9, D11 (both **Play=Yes**)

This node will need to be expanded by testing some other feature
Overfitting in Decision Trees

- What if we added a noisy example in our Tennis Playing dataset?

  Outlook=Sunny, Temperature=Hot, Humidity=Normal, Wind=Strong, Play=No

  This Play=No example would be grouped with the node D9, D11 (both Play=Yes)

  ![Decision Tree Diagram]

  This node will need to be expanded by testing some other feature.

  The new tree would be more complex than the earlier one (trying to fit noise).

  The extra complexity may not be worth it ⇒ may lead to overfitting if the test data follows the same pattern as our normal training data.

Note: Overfitting may also occur if the training data is not sufficient.
Overfitting in Decision Trees

- What if we added a noisy example in our Tennis Playing dataset?
- Outlook=Sunny, Temperature=Hot, Humidity=Normal, Wind=Strong, Play=No
- This Play=No example would be grouped with the node D9, D11 (both Play=Yes)

This node will need to be expanded by testing some other feature
The new tree would be more complex than the earlier one (trying to fit noise)
The extra complexity may not be worth it ⇒ may lead to overfitting if the test data follows the same pattern as our normal training data

Note: Overfitting may also occur if the training data is not sufficient
Overfitting in Decision Trees

- **Overfitting Illustration**

The graph shows the accuracy of a decision tree model as a function of the size of the tree (number of nodes). The solid line represents the accuracy on the training data, while the dashed line represents the accuracy on the test data. High training data accuracy doesn't necessarily imply high test data accuracy.
Overfitting in Decision Trees

- **Overfitting Illustration**

- High training data accuracy doesn’t necessarily imply high test data accuracy
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn’t hurt the accuracy on the validation set

- Statistical tests such as the $\chi^2$ test (Quinlan, 1986)
- Minimum Description Length (MDL): more details when we cover Model Selection

K-NN and DT
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn’t hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most

Statistical tests such as the $\chi^2$ test (Quinlan, 1986)
Minimum Description Length (MDL): more details when we cover Model Selection

K-NN and DT
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn’t hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
  - Prune each possible node that doesn’t hurt the accuracy on the validation set
  - Greedily remove the node that improves the validation accuracy the most
  - Stop when the validation set accuracy starts worsening
  - Statistical tests such as the $\chi^2$ test (Quinlan, 1986)
Avoiding Overfitting: Decision Tree Pruning

- Desired: a DT that is not too big in size, yet fits the training data reasonably

- Mainly two approaches
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn’t hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening
  - Statistical tests such as the $\chi^2$ test (Quinlan, 1986)
  - Minimum Description Length (MDL): more details when we cover Model Selection
Avoiding Overfitting: Decision Tree Pruning

![Graph showing the accuracy of decision trees as a function of tree size. The graph compares the accuracy on training data, test data, and during pruning.](image)
Dealing with Missing Features

- Want to compute $IG(S, F)$ for feature $F$ on a (sub)set of training data $S$

- What if a training example in $x$ in $S$ has feature $F$ missing?
Dealing with Missing Features

- Want to compute $IG(S, F)$ for feature $F$ on a (sub)set of training data $S$

- What if a training example in $x$ in $S$ has feature $F$ missing?

- We will need some way to *approximate* the value of this feature for $x$
Dealing with Missing Features

Want to compute $IG(S, F)$ for feature $F$ on a (sub)set of training data $S$

What if a training example in $x$ in $S$ has feature $F$ missing?

We will need some way to *approximate* the value of this feature for $x$

**One way:** Assign the value of $F$ which a majority of elements in $S$ have
Dealing with Missing Features

- Want to compute $IG(S, F)$ for feature $F$ on a (sub)set of training data $S$

- What if a training example in $x$ in $S$ has feature $F$ missing?

- We will need some way to *approximate* the value of this feature for $x$

- **One way:** Assign the value of $F$ which a majority of elements in $S$ have

- **Another (maybe better?) way:** Assign the value of $F$ which a majority of elements in $S$ with the same label as $x$ have
Real-valued features can be dealt with using thresholding.
• Real-valued features can be dealt with using thresholding

• Real-valued labels (Regression Trees) by re-defining entropy or using other criteria (how similar to each other are the y’s at any node)
Decision Tree Extensions

- Real-valued features can be dealt with using thresholding
- Real-valued labels (Regression Trees) by re-defining entropy or using other criteria (how similar to each other are the \( y \)'s at any node)
- Other criteria for judging feature informativeness
  - Gini-index, misclassification rate
Decision Tree Extensions

- Real-valued features can be dealt with using thresholding

- Real-valued labels (Regression Trees) by re-defining entropy or using other criteria (how similar to each other are the y’s at any node)

- Other criteria for judging feature informativeness
  - Gini-index, misclassification rate

- Handling features with differing costs (see the DT handout, section 3.7.5)
Decision Tree Extensions

- Real-valued features can be dealt with using thresholding
- Real-valued labels (Regression Trees) by re-defining entropy or using other criteria (how similar to each other are the y’s at any node)
- Other criteria for judging feature informativeness
  - Gini-index, misclassification rate
- Handling features with differing costs (see the DT handout, section 3.7.5)
- Approaches other than greedy tree building
Data Representation

Data Representation (we briefly talked about it in the first lecture)
Most learning algorithms require the data in some numeric representation (e.g., each input pattern is a vector)
Most learning algorithms require the data in some numeric representation (e.g., each input pattern is a vector).

If the data naturally has numeric (real-valued) features, one way is to just represent it as a vector of real numbers.

E.g., a $28 \times 28$ image by a $784 \times 1$ vector of its pixel intensities.
Data to Features

Most learning algorithms require the data in some numeric representation (e.g., each input pattern is a vector).

If the data naturally has numeric (real-valued) features, one way is to just represent it as a vector of real numbers.

- E.g., a $28 \times 28$ image by a $784 \times 1$ vector of its pixel intensities.

What if the data has a non-numeric representation?

- An email (a text document)
Data to Features

- Most learning algorithms require the data in some numeric representation (e.g., each input pattern is a vector).

- If the data naturally has numeric (real-valued) features, one way is to just represent it as a vector of real numbers.
  - E.g., a $28 \times 28$ image by a $784 \times 1$ vector of its pixel intensities.

- What if the data has a non-numeric representation?
  - An email (a text document).

- Let’s look at some examples..
A possible feature vector representation for a text document

- **Dear Sir.**
  
  First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidential and top secret. ...

- **Feature vector representation**:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>W=dear</td>
<td>1</td>
</tr>
<tr>
<td>W=sir</td>
<td>1</td>
</tr>
<tr>
<td>W=his</td>
<td>2</td>
</tr>
<tr>
<td>W=wish</td>
<td>0</td>
</tr>
<tr>
<td>MISSPELLED</td>
<td>2</td>
</tr>
<tr>
<td>NAMELESS</td>
<td>1</td>
</tr>
<tr>
<td>ALL_CAPS</td>
<td>0</td>
</tr>
<tr>
<td>NUM_URLS</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Let's consider a dataset similar to the Tennis Playing example.

Features are nominal (Low/High, Yes/No, Overcast/Rainy/Sunny, etc.)
Let's consider a dataset similar to the Tennis Playing example
- Features are nominal (Low/High, Yes/No, Overcast/Rainy/Sunny, etc.)
- Features with only 2 possible values can be represented as 0/1
Data to Features: Symbolic/Categorical/Nominal Features

Let's consider a dataset similar to the Tennis Playing example.

Features are nominal (Low/High, Yes/No, Overcast/Rainy/Sunny, etc.)

Features with only 2 possible values can be represented as 0/1.

What about features having more than 2 possible values?
### Data to Features: Symbolic/Categorical/Nominal Features

Let's consider a dataset similar to the Tennis Playing example.

Features are nominal (Low/High, Yes/No, Overcast/Rainy/Sunny, etc.)

Features with only 2 possible values can be represented as 0/1

What about features having more than 2 possible values?
Can't we just map Sunny to 0, Overcast to 1, Rainy to 2?

<table>
<thead>
<tr>
<th>Y</th>
<th>Out</th>
<th>T</th>
<th>R</th>
<th>Y</th>
<th>(Out, T, R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Sunny</td>
<td>Low</td>
<td>Yes</td>
<td>1</td>
<td>(?, 0, 1)</td>
</tr>
<tr>
<td>N</td>
<td>Sunny</td>
<td>High</td>
<td>Yes</td>
<td>0</td>
<td>(?, 1, 1)</td>
</tr>
<tr>
<td>N</td>
<td>Sunny</td>
<td>High</td>
<td>No</td>
<td>0</td>
<td>(?, 1, 0)</td>
</tr>
<tr>
<td>P</td>
<td>Overcast</td>
<td>Low</td>
<td>Yes</td>
<td>1</td>
<td>(?, 0, 1)</td>
</tr>
<tr>
<td>P</td>
<td>Overcast</td>
<td>High</td>
<td>No</td>
<td>1</td>
<td>(?, 1, 0)</td>
</tr>
<tr>
<td>P</td>
<td>Overcast</td>
<td>Low</td>
<td>No</td>
<td>1</td>
<td>(?, 0, 0)</td>
</tr>
<tr>
<td>N</td>
<td>Rainy</td>
<td>Low</td>
<td>Yes</td>
<td>0</td>
<td>(?, 0, 1)</td>
</tr>
<tr>
<td>P</td>
<td>Rainy</td>
<td>Low</td>
<td>No</td>
<td>1</td>
<td>(?, 0, 0)</td>
</tr>
</tbody>
</table>
Data to Features

- Well, we could map Sunny to 0, Overcast to 1, Rainy to 2..
- But such a mapping may not always be appropriate
  - Imagine color being a feature in some data
  - Let’s code 3 possible colors as Red=0, Blue=1, Green=2
Data to Features

- Well, we could map Sunny to 0, Overcast to 1, Rainy to 2.
- But such a mapping may not always be appropriate.
  - Imagine color being a feature in some data.
  - Let’s code 3 possible colors as Red=0, Blue=1, Green=2.
  - This implies Red is more similar to Blue than to Green!
Well, we could map Sunny to 0, Overcast to 1, Rainy to 2.
But such a mapping may not always be appropriate
- Imagine color being a feature in some data
- Let’s code 3 possible colors as Red=0, Blue=1, Green=2
- This implies Red is more similar to Blue than to Green!

**Solution:** For a feature with $K > 2$ possible values, we usually create $K$ binary features, one for each possible value

<table>
<thead>
<tr>
<th>Y</th>
<th>Out</th>
<th>T</th>
<th>R</th>
<th>Y</th>
<th>(S?, O?, R?, T, R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Sunny</td>
<td>Low</td>
<td>Yes</td>
<td>1</td>
<td>(1, 0, 0, 0, 1)</td>
</tr>
<tr>
<td>N</td>
<td>Sunny</td>
<td>High</td>
<td>Yes</td>
<td>0</td>
<td>(1, 0, 0, 1, 1)</td>
</tr>
<tr>
<td>N</td>
<td>Sunny</td>
<td>High</td>
<td>No</td>
<td>0</td>
<td>(1, 0, 0, 1, 0)</td>
</tr>
<tr>
<td>P</td>
<td>Overcast</td>
<td>Low</td>
<td>Yes</td>
<td>1</td>
<td>(0, 1, 0, 0, 1)</td>
</tr>
<tr>
<td>P</td>
<td>Overcast</td>
<td>High</td>
<td>No</td>
<td>1</td>
<td>(0, 1, 0, 1, 0)</td>
</tr>
<tr>
<td>P</td>
<td>Overcast</td>
<td>Low</td>
<td>No</td>
<td>1</td>
<td>(0, 1, 0, 0, 0)</td>
</tr>
<tr>
<td>N</td>
<td>Rainy</td>
<td>Low</td>
<td>Yes</td>
<td>0</td>
<td>(0, 0, 1, 0, 1)</td>
</tr>
<tr>
<td>P</td>
<td>Rainy</td>
<td>Low</td>
<td>No</td>
<td>1</td>
<td>(0, 0, 1, 0, 0)</td>
</tr>
</tbody>
</table>