Supervised Learning: 
*K*-Nearest Neighbors and Decision Trees

Piyush Rai

CS5350/6350: Machine Learning

August 25, 2011
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label
- \( x_i \) is a vector consisting of \( D \) features
  - Also called attributes or dimensions
  - Features can be discrete or continuous
  - \( x_{im} \) denotes the \( m \)-th feature of \( x_i \)
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)

- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label

- \( x_i \) is a vector consisting of \( D \) features
  - Also called attributes or dimensions
  - Features can be discrete or continuous
  - \( x_{im} \) denotes the \( m \)-th feature of \( x_i \)

- Forms of the output:
  - \( y_i \in \{1, \ldots, C\} \) for classification; a discrete variable
  - \( y_i \in \mathbb{R} \) for regression; a continuous (real-valued) variable
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label
- \( x_i \) is a vector consisting of \( D \) features
  - Also called attributes or dimensions
  - Features can be discrete or continuous
  - \( x_{im} \) denotes the \( m \)-th feature of \( x_i \)

- Forms of the output:
  - \( y_i \in \{1, \ldots, C\} \) for classification; a discrete variable
  - \( y_i \in \mathbb{R} \) for regression; a continuous (real-valued) variable

- **Goal:** predict the output \( y \) for an unseen test example \( x \)

- **This lecture:** Two intuitive methods
  - \( K \)-Nearest-Neighbors
  - Decision Trees
**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
**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**
\textit{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

The algorithm requires:
- Parameter $K$: number of nearest neighbors to look for
- \textit{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
**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
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.
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.
**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.
- **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).
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
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

- **One way is:**
  - Replace $x_{im}$ by $z_{im} = \frac{(x_{im} - \bar{x}_m)}{\sigma_m}$ (make them zero mean, unit variance)
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.

- 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: \( d(x_i, x_j) = \sum_{m=1}^{D} I(x_{im} \neq x_{jm}) \)
  - 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: \( d(x_i, x_j) = \sum_{m=1}^{D} w_m d(x_{im}, x_{jm}) \)
Choice of $K$ - Neighborhood Size

- **Small $K$**
  - Creates many small regions for each class
  - May lead to non-smooth decision boundaries and overfit
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
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**)
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)
\textbf{K-Nearest Neighbor: Properties}

\textbf{What's nice}
- Simple and intuitive; easily implementable
- Asymptotically \textit{consistent} (a theoretical property)
  - With infinite training data and large enough \( K \), \( K \)-NN approaches the best possible classifier (Bayes optimal)

\textbf{What's not so nice.}
- Store all the training data \textit{in memory} even at test time
  - Can be memory intensive for large training datasets
  - An example of \textit{non-parametric}, or \textit{memory/instance-based} methods
  - Different from \textit{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!
Computational speed-ups (don’t want to spend $O(ND)$ time)
- Improved data structures for fast nearest neighbor search
- Even if *approximately* nearest neighbors, yet may be good enough

Efficient Storage (don’t want to store all the training data)
- E.g., subsampling the training data to retain “prototypes”
- Leads to computational speed-ups too!

Metric Learning: Learning the “right” distance metric for a given dataset
Decision Tree

- 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

- 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 $x$
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
**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 showing decision tree and data points]

- 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

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
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
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 quantify the informativeness.

**Information content** of a feature decides its position in 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>weak</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>strong</td>
<td>no</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

- 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 $c$. 

K-NN and DT

August 25, 2011 16 / 20
Entropy

- **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 $c$

.. 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 denotes the average number of bits needed to encode $S$
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 \( F \)
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” $\in \{\text{weak, strong}\}$ and its $IG$ w.r.t. the root node.
Computing Information Gain

Let’s begin with the root node of the DT and compute $IG$ of each feature.

Consider feature “wind” $\in \{\text{weak}, \text{strong}\}$ and its $IG$ w.r.t. the root node.

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

Entropy: $H(S) = -\left(\frac{9}{14}\right) \log_2 \left(\frac{9}{14}\right) - \left(\frac{5}{14}\right) \log_2 \left(\frac{5}{14}\right) = 0.94$
Computing Information Gain

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

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

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

$S_{weak} = [6+, 2-] \implies H(S_{weak}) = 0.811$
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: \( H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94 \)

\( S_{weak} = [6+, 2-] \implies H(S_{weak}) = 0.811 \)

\( S_{strong} = [3+, 3-] \implies H(S_{strong}) = 1 \)
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: $H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94$

- $S_{weak} = [6+, 2−] \implies H(S_{weak}) = 0.811$
- $S_{strong} = [3+, 3−] \implies H(S_{strong}) = 1$

$$IG(S, wind) = H(S) - \frac{|S_{weak}|}{|S|} H(S_{weak}) - \frac{|S_{strong}|}{|S|} H(S_{strong})$$

$$= 0.94 - 8/14 \times 0.811 - 6/14 \times 1$$

$$= 0.048$$
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
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
Next Lecture..

- The ID3 Decision Tree Algorithm formally
  - We have already seen the ingredients by now

- Decision Tree Properties
  - E.g., dealing with missing features, overfitting, etc.

- Maths Refresher