Lecture 5
Supervised Learning
RBFNN, SVM, k-nn and DT

Dr. Patrick Chan
patrickchan@ieee.org
South China University of Technology, China

Agenda

- Radial Basis Function Neural Network
- Support Vector Machine
- K-Nearest Neighbor
- Decision Tree
Recall...

Multi-Layer Perception

- Black-Box
  - No idea what happened in side a progress
- Complex parameter space
  - High training complexity
  - Sub-optimal solution

RBF Neural Network

- Radial Basis Function (RBF) Neural Network is special type of ANN
- Only three layers
  - Input Layer
  - Hidden Layer
    - Activation Function: RBF (Gaussian)
  - Output Layer
    - Linearly weighted Sum
- Parameters are meaningful
RBF Neural Network

- Recall, Gaussian function
  - The most common distribution function
  - Univariate function:
    \[ \varphi(x) = \exp \left( -\frac{1}{2\sigma^2} \sum_{i=1}^{d} (x_i - \mu_i)^2 \right) \]
  - Two parameters
    - \( \mu \): mean
    - \( \sigma^2 \): variance

- Neuron (\( \varphi \)) of RBFNN represents Gaussian Distribution with \( \mu \) and \( \sigma^2 \)
  - Measures the effect of the neuron to \( x \)
  - If the output is large, it means \( x \) and the neuron have more similar nature

\[ \varphi_1(x) < \varphi_2(x) \quad x \text{ is more related to } \varphi_2 \]
\[ \varphi_1(x) < \varphi_2(x) \quad x \text{ is more related to } \varphi_2 \]
If the weight is larger, the neuron is more important as it has larger effect to the output.

The output of RBFNN for the class $k$ is:

$$g_k(x) = \sum_{j=1}^{m} w_{kj} \varphi_j(x)$$

- $w$ : the weight (importance of the neuron)
- $m$ : the number of neurons

$$\varphi_j(x) = \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{d} (x_i - \mu_{ji})^2 \right)$$
RBF Neural Network

Parameter Determination

- Parameters ($\mu$, $\sigma^2$ and $w$) are meaningful
  - Can be determined by not only gradient descent but also clustering technique
    - Objective is to seek the “natural” clusters in the data
    - Clustering will be discussed later in detail
RBF Neural Network: Parameter Determination

**Gradient Descent**

- **Weights** \( w \)
  \[ w_{i+1} = w_i - \eta_w \frac{\partial J}{\partial w_i(x)} \]

- **Centers** \( \mu \)
  \[ \mu_{i+1} = \mu_i - \eta_\mu \frac{\partial J}{\partial \mu_i(x)} \]

- **Width** \( \sigma^2 \)
  \[ \sigma^2_{i+1} = \sigma^2_i - \eta_{\sigma^2} \frac{\partial J}{\partial \sigma^2_i(x)} \]

\[ g_k(x) = \sum_{j=1}^{m} w_{kj} \exp \left( -\frac{1}{2 \sigma_j^2} \sum_{i=1}^{d} (x_i - \mu_{ji})^2 \right) \]

Learning rate of parameters can be different

**Clustering Technique**

- **Given a dataset**
- **Separate samples into groups for each class by clustering technique**
  - E.g. \( K \)-mean
  - May determine the number of clusters for each class in advance

- **Center** \( \mu \)
  - Mean of samples belonging to the same cluster

- **Width** \( \sigma^2 \)
  - Variance of samples belonging to the same cluster
- $\varphi$ is fixed when $\mu$ and $\sigma^2$ are determined
- RBFNN:
  \[ g_k(x) = \sum_{j=1}^{m} w_{kj} \varphi_j(x) \]
  \[ \hat{Y} = \Phi W \]
- As $\Phi$ and $\hat{Y}$ are linear relation
- Weight ($w$) can be calculated by Pseudoinverse (refer to the previous lecture)

---

**2-Phase Learning**

- Initialization of gradient descent affects the performance significantly
- **2-Phase Learning** combines clustering and gradient descent
  1. Cluster technique initials values of $\mu$, $\sigma^2$ and $w$
  2. Gradient descent fine-tunes the values
RBF Network

- How to determine the number of neurons?
  - Less neurons
    - Low complexity
    - Low training cost
  - More neurons
    - Represents more complicated distribution
  - Special Case
    Maximum $m = \text{training sample } \# (n)$
    - Mean: a training sample
    - Covariance: influence of a sample to unseen samples

RBF Neural Network

XOR Example

- Input space:

- Output space:
RBF Neural Network

**XOR Example**

- The feature space:

\[
\phi_1(x_1, x_2) = e^{-\frac{|x-u_1|^2}{2v_1^2}}
\]

\[
u_1 = 1/\sqrt{2}
\]

\[
\phi_2(x_1, x_2) = e^{-\frac{|x-u_2|^2}{2v_2^2}}
\]

\[
u_2 = 1/\sqrt{2}
\]

<table>
<thead>
<tr>
<th>x</th>
<th>(\phi_1)</th>
<th>(\phi_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1)</td>
<td>1</td>
<td>0.1353</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>0.3678</td>
<td>0.3678</td>
</tr>
<tr>
<td>(0, 0)</td>
<td>0.1353</td>
<td>1</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>0.3678</td>
<td>0.3678</td>
</tr>
</tbody>
</table>

- Weight Determination:

\[
\Phi W = d
\]

\[
\Phi = \begin{pmatrix}
1 & 0.1353 & 1 \\
0.3678 & 0.3678 & 1 \\
0.1353 & 1 & 1 \\
0.3678 & 0.3678 & 1 \\
\phi_1(x_1, x_2) & \phi_2(x_1, x_2) & w_3
\end{pmatrix}
\]

\[
W = \begin{pmatrix}
w_1 \\
w_2 \\
w_3
\end{pmatrix}
\]

\[
d = \begin{pmatrix}
1 \\
0 \\
1 \\
0
\end{pmatrix}
\]

Class outputs of samples

Decision boundary
RBF Network: Characteristic

◆ **Advantage**
  - RBF Network trains faster than MLP
  - The hidden layer is easier to interpret than MLP

◆ **Disadvantage**
  - During the test phase, the calculation speed of a neuron in RBF is slower than MLP
  - Architecture is fixed

Support Vector Machine (SVM)

◆ Which one is the **best** linear separator?
A clever sheep dog who was herding his sheep...

It runs between the sheep and tries to separate the black sheep and white sheep.

The sheep dog keeps running...

The sheep start to grow wools...

The dog feel the gap between black sheep and white sheep is narrower...
The wools become bigger and bigger...

Finally, only one path is left..

From: Learning with Kernels, Schölkopf & Smola

The sheep found out that the single path relies only on the some sheep.

These sheep are “sheep vectors”

* “Support vector” in SVM
Support Vector Machine (SVM)

- Similar to the sheep, if a sample is growing...
  
This should be the best classifier

- A classifier with the largest margin
- The concept is the same
Support Vector Machine (SVM)

- Maximum Margin Classifier ONLY depends on few samples, called **Support Vectors**

![Diagram of Support Vectors]

Support Vector Machine
Statistical Learning Theory

- Recall, a **classifier** is trained by minimizing the training error ($R_{emp}$, empirical risk)
- Our **ultimate objective** is to minimize the error on unseen sample ($R$, true risk)
- Training error can be reduced easily by using a more complicated classifier
  - Overfit easily too
  - A simpler classifier is preferred (assume empirical risk is the same)
**Support Vector Machine**

**Statistical Learning Theory**

- **SLT** is proposed by Prof. Vladimir Vapnik

- Mention the relation between $R$, $R_{\text{emp}}$ and classifier complexity

- With 1-$\eta$ probability, the bound on true risk ($R$) is:

  $$R \leq R_{\text{emp}} + \sqrt{\frac{h(\log(2n/h) + 1) - \log(\eta/4)}{n}}$$

  - $n$ = number of training samples
  - $h$ = VC (Vapnik-Chervonenkis) dimension
    - Complexity of the classifier

- **Observation:**
  - When $l \to \infty$, $R_{\text{emp}} \to R$
  - More sample, less complexity
  - This bound is independent of the sample distribution

Lecture 05: SL - RBFNN, SVM, k-nn and DT

Dr. Patrick Chan @ SCUT
VC Dimension for a set of functions \{ f \} is \( h \) if there exists a set of \( h \) points that can be shattered by \{ f \} but no set of \( h+1 \) points.

- Shattered means that any labeling of the points can be classified correctly by a function from \{ f \}.

- Larger \( h \) = more complicated classifier.

**Linear Classifier Example**

- 2 features
- Linear Classifier is a straight line
- \( h \) is sample #
**Linear Classifier Example**

- Maximum value of $h$ is 3
- Therefore, the VC-Dimension of Linear classifier is 3

**Support Vector Machine (SVM)**

- SVM with a larger margin has a smaller VC dimension
- Not only $R_{emp}$ but also the margin is optimized in SVM training
  - Implicitly optimize $R$
- How to find the boundary?
  - Separable case
  - Non-separable case
    - Slack variables
    - Kernel method
SVM: Linearly Separable

- The hyperplane is:
  \[ wx + b = 0 \]

- Suppose the dotted lines are:
  \[ wx + b = \pm \sigma \]

- If change:
  \( w = \lambda w, \ b = \lambda b \) and \( \sigma = \lambda \sigma \)
  the equations are still valid

- We choose \( \sigma = 1 \)

Distance from origin to hyperplane is

\[ \frac{||b||}{w} \]

Similar, the distance to dotted line is

\[ \frac{(||b|| \pm 1)}{w} \]

Therefore, the margin is:

\[
\frac{b + 1}{||w||} - \frac{b - 1}{||w||} = \frac{2}{||w||}
\]
SVM: Linearly Separable

- Problem can be formulated as Quadratic Optimization Problem and solve for $w$ and $b$

$$\begin{align*}
\text{minimize} &\quad \frac{1}{2} \|w\|^2 \\
\text{subject to} &\quad y_i(w^T x_i + b) \geq 1
\end{align*}$$

where $i = 1 \ldots n$ and $y = \{1, -1\}$

This optimization problem can be formulated as **Dual Problem** using Lagrangian method:

$\begin{align*}
\text{minimize} &\quad \frac{1}{2} \|w\|^2 \\
\text{subject to} &\quad y_i(w^T x_i + b) \geq 1
\end{align*}$

- Weight is determined by: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
SVM: Linearly Separable

- Many $\alpha_i$ are zero
- $x_i$ with non-zero $\alpha_i$ are support vectors (SV)
  - The decision boundary is determined only by the SV

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

- For testing with a new data $x$
  $$f(z) = w^T z + b$$
  where
  $$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

  $$f(z) = \sum_{j=1}^{l} \alpha_t y_t \left( x_t^T z \right) + b$$

- If $f(z) > 0$, class 1
- If $f(z) < 0$, class 2
- Note: $w$ need not be formed explicitly
SVM: Non-Linearly Separable

- **How about Non-Linearly Separable Case?**
  - The margin cannot be defined anymore

- **Two approaches:**
  - Add a slack variables
  - Use a kernel (Non-Linear SVM)

**Slack Variable**

- \( \xi_1 > 1 \) Error
- \( \xi_2 > 1 \) Error
- \( \xi_3 < 1 \) Correct
- \( \xi_4 > 1 \) Error
- \( \xi_5 < 1 \) Correct

- **Slack Variable (\( \xi \))** is added as a punishment to allow a sample in far away from the margin

- **Optimization:**

  Minimize \( \frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \xi_i \)

  subject to \( y_i(w^T x_i + b) \geq 1 - \xi_i \quad i = 1...N \)

  where \( \xi_i \geq 0 \)

  \( C \) : tradeoff parameter between error and margin
SVM: Non-Linearly Separable

**Slack Variable**

- The **dual** of this new constrained optimization problem is:
  
  \[
  \text{Maximum}_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j 
  \]

  subject to \( C \geq \alpha_i \geq 0 \)
  
  \[ i = 1 \ldots n \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \]

- It is as the same as the **linearly separable case**, except that there is an upper bound \( C \) on \( \alpha_i \)

- \( w \) is calculated by:
  
  \[
  w = \sum_{i=1}^{n} \alpha_i y_i x_i 
  \]

- **Kernel Method**
  
  - Map samples to higher dimensional space
    - **Input space**: the original space of \( x_i \)
    - **Feature space**: \( \phi(x_i) \)
  
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  
  - Classification can become easier with a proper transformation

---

**Support Vector** \((C \geq \alpha_i \geq 0)\)

- Non-SV \( \alpha_i = 0 \)

- Samples on the margin
- Samples in / far away from margin (\( \xi > 0 \))
Kernel is a function which maps input into high dimensional feature space.

Construct linear SVM in feature space.

Recall, the linearly separable case:

Maximum \( \alpha \)
\[
\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j
\]
subject to \( \alpha_i \geq 0 \) and \( \alpha_i y_i = 0 \) for \( i = 1 \ldots n \)

The data points only appear as inner product.
As long as the inner product is calculated, the mapping is not required explicitly.
Define the kernel function \( K \) by
\[
K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)
\]
Suppose $\phi()$ is given as follows

$$\varphi([x_1, x_2]^T) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

An inner product in the feature space is

$$\langle \varphi([x_1, x_2]^T), \varphi([y_1, y_2]^T) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

The kernel function is defined as follows

$$K(x, y) = (1 + x_1y_1 + x_2y_2)^2$$

No need to define $\phi(.)$ explicitly

It is known as the kernel trick

---

**Polynomial** kernel with degree $d$

$$K(x, y) = (x^T y + 1)^d$$

**Radial basis function** kernel with width $\sigma$

$$K(x, y) = \exp\left(\frac{-\|x - y\|^2}{(2\sigma)^2}\right)$$

- Closely related to radial basis function neural networks
- The feature space is infinite-dimensional

**Sigmoid** with parameter $\kappa$ and $\theta$

$$K(x, y) = \tanh(\kappa x^T y + \theta)$$

- It does not satisfy the Mercer condition on all $\kappa$ and $\theta$
SVM: Non-Linearily Separable

Kernel Method: Example

- Similar to the linearly separable case but change all inner products to kernel functions

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad y_i (w^T \varphi(x_i) + b) \geq 1 \quad i = 1 \ldots N
\end{align*}
\]

\[
\begin{align*}
\text{maximize} & \quad \alpha \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\
\text{subject to} & \quad \alpha_i \geq 0 \quad i = 1 \ldots n \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0
\end{align*}
\]

- For testing with a new data \( z \)

\[
f(z) = w^T z + b \quad \text{where} \quad w = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i)
\]

\[
f(z) = \sum_{j=1}^{l} \alpha_{t_j} y_{t_j} K(x_{t_j}, z) + b
\]

- If \( f(z) > 0 \), class 1
- If \( f(z) < 0 \), class 2
- Note: \( \varphi \) needs not be formed explicitly
SVM: Example

Given:
- 5 one-dimensional training samples:
- Polynomial kernel of degree 2 is used:
  \[ K(x,y) = (xy+1)^2 \]
- \( C \) is set to 100

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

By
Maximum
\[
\sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2
\]
subject to \( 100 \geq \alpha_i \geq 0 \) for \( i = 1 \ldots 5 \) and \( \sum_{i=1}^{5} \alpha_i y_i = 0 \)

Result:
- \( \alpha_1 = 0, \alpha_2 = 2.5, \alpha_3 = 0, \alpha_4 = 7.333, \alpha_5 = 4.833 \)
- The support vectors are \( \{ x_2 = 2, x_4 = 5, x_5 = 6 \} \)

The discriminate function is
\[
f(z) = \sum_{j=1}^{3} \alpha_{t_j} y_{t_j} \left( x_{t_j}^T z + 1 \right)^2 + b
\]
\[
= 2.5(1)(2z + 1)^2 + 7.333(-1)(5z + 1)^2 + 4.833(1)(6z + 1)^2 + b
\]
\[
= 0.6667z^2 - 5.333z + b
\]
\[ f(z) = 0.6667z^2 - 5.333z + b \]

- \( b \) can be determined by using the points on the margin, \( \varphi(w)^T\varphi(x) + b = \pm 1 \)
  - \( f(2) = 1 \), \( f(6) = 1 \) and \( f(5) = -1 \)
  - Therefore, \( b \) is -9

\[ f(z) = 0.6667z^2 - 5.333z - 9 \]
SVM: Multi-Class Problem

- SVM only can handle 2-class problem
- How to handle multi-class problem?
  - $g$ in LDF can be formulated as the estimation on posterior probability to a class
  - However, SVM must considers two classes
    - Do not estimate the probability of a class
    - Max method cannot be applied to SVM

How to handle multi-class problem?
- 1-against-All
- 1-against-1
- Discussed in previous lecture
**SVM: Characteristic**

- **Advantages**
  - Training is relatively easy
  - No local optimal
  - It scales relatively well to high dimensional data (inner product)
  - Tradeoff between classifier complexity and error can be controlled explicitly

- **Disadvantage**
  - Slow when the number of samples is large
  - Need to choose a “good” kernel function

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

- A new pattern is classified by a majority vote of its $k$ nearest neighbors (training samples)
- $n$ distances are calculated for each new sample
  - $n$: the number of training samples
K-Nearest Neighbor (K-NN)

- Target function for the entire space may be described as a combination of less complex local approximations

How to determine $k$?

- Small $k$
  - Noise Sensitive
- Large $k$
  - Neighbours may be too far away from the unseen sample
  - Less representative
K-NN: Characteristic

- **Advantages:**
  - Very **simple**
  - No training is needed
    - All computations deferred until classification

- **Disadvantages:**
  - Difficult to determine $k$
  - Affected by **noisy** training data
  - Classification is **time consuming**
    - Need to calculate the distance between the unseen sample and each training sample

Decision Tree (DT)

- Most classifiers are black-box
- DT provides **explanation** on decisions
- One of the most widely used and practical methods for inductive inference
- Approximates discrete-valued functions (including disjunctions)
DT: Example

- Do we go to play tennis today?

<table>
<thead>
<tr>
<th>Condition</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook is Sunny AND Humidity is Normal</td>
<td>Yes</td>
</tr>
<tr>
<td>Outlook is Overcast</td>
<td>Yes</td>
</tr>
<tr>
<td>Outlook is Rain AND Wind is Weak</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Other situation? No

DT: Classification

- Decision Region:
  - Internal nodes can be univariate
  - (Only one feature is used)
DT: Classification

- **Internal nodes** can be multivariate
  - More than one features are used
  - Shape of Decision Region is irregular

\[ ax_1 + bx_2 + c > 0 \]

DT: Learning Algorithm

- **LOOP:**
  1. Select the **best feature** (A)
  2. For each value of A, create new descendant of node
  3. Sort training samples to leaf nodes
- **STOP** when training samples perfectly classified
DT: Learning Algorithm

- **Observation**
  - Many trees may code a training set without any error
  - Finding the smallest tree is a NP-hard problem

- **Local search algorithm** to find reasonable solutions
  - What is the best feature?

DT: Feature Measurement

- **Entropy** is used to evaluate features
  - Measure of uncertainty
  - Range: 0 - 1
  - Smaller value, less uncertainty

\[
H(X) = - \sum_{i=1}^{n} p(x_i) \log_2 p(x_i)
\]

where

- \(X\): a random variable with \(n\) outcomes, \(X = \{x_i | i = 1, 2, \ldots, n\}\)
- \(p(x)\): the probability mass function of outcome \(x\).

- If all samples belongs to \(x_i\), then \(p(x_i) = 1\), and other \(p(x_j) = 0, i \neq j\)
  - Thus, \(H(X) = 0\) (no uncertainty)
**DT: Feature Measurement**

- **Information Gain**
  - Reduction in entropy (reduce uncertainty) due to sorting on a feature $A$

\[
Gain(X, A) = H(X) - H(X|A)
\]

**Current entropy**

**Entropy after using feature $A$**

**DT: Example**

- Which feature is the best?

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

\[
H(X) = - \sum_{i=1}^{n} p(x_i) \log_2 p(x_i)
\]

$x_1 = \text{yes}$  \hspace{1cm} $x_2 = \text{No}$

\[
H(X) = - \sum_{i=1}^{2} p(x_i) \log_2 p(x_i)
\]

\[
= - \frac{9}{14} \log_2 \left( \frac{9}{14} \right) - \frac{5}{14} \log_2 \left( \frac{5}{14} \right)
\]

\[
= 0.410 + 0.531
\]

\[
= 0.941
\]

**Current:** $H(X) = 0.941$

Uncertainty is high w/o any sorting by feature
Let A = Outlook

\[
\begin{align*}
H(X | A) &= H(X | A = \text{sunny})P(A = \text{sunny}) + \\
&+ H(X | A = \text{Rain})P(A = \text{Rain}) + \\
&+ H(X | A = \text{overcast})P(A = \text{overcast})
\end{align*}
\]

\[
H(X | \text{sunny}) = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971
\]

\[
H(X | \text{Rain}) = -\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) = 0.971
\]

\[
H(X | \text{overcast}) = -\frac{0}{4}\log_2\left(\frac{0}{4}\right) - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0
\]

\[
H(X | A) = 0.971 \times \left(\frac{5}{14}\right) + 0.971 \times \left(\frac{5}{14}\right) + 0 \times \left(\frac{4}{14}\right)
\]

\[
= 0.694
\]
Dr. Patrick Chan @ SCUT

DT: Example

Outlook is the best feature and Should be used as the first node

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

Current: \[ H(X) = 0.941 \]

Similarly, for each feature

\[ H(X \mid \text{Outlook}) = 0.694 \]
\[ H(X \mid \text{Temperature}) = 0.911 \]
\[ H(X \mid \text{Humidity}) = 0.789 \]
\[ H(X \mid \text{Wind}) = 0.892 \]

Information Gain is:

\[ \text{Gain}(X, \text{Outlook}) = 0.247 \]
\[ \text{Gain}(X, \text{Temperature}) = 0.030 \]
\[ \text{Gain}(X, \text{Humidity}) = 0.152 \]
\[ \text{Gain}(X, \text{Wind}) = 0.049 \]

Recall:

\[ \text{Gain}(X, A) = H(X) - H(X \mid A) \]

Next Step

- Repeat the steps for each sub-branch
- Until there is no ambiguity
  (all samples are of the same class)

Outlook

- Sunny
  - No: 3
  - Yes: 2
- Rain
  - No: 2
  - Yes: 3
- Overcast
  - No: 0
  - Yes: 4

Continues to select next features

Done
So far, we handle features with categorical values

How to build a decision tree whose features are numerical?

Accomplished by partitioning the continuous attribute value into a discrete set of intervals

In particular, a new Boolean feature $A_c (A < c)$ can be created

How to select the best value for $c$
Objective is to minimize the entropy (or maximize the information gain)

Entropy only needs to be evaluated between points of different classes

Best $c$ is: $c^* = \arg \max_c Gain(X, A_c)$