Topics in Machine Learning: I[1]

Instructor: Dr. B. John Oommen

Chancellor's Professor
Fellow: IEEE; Fellow: IAPR
School of Computer Science, Carleton University, Canada.

[1] The primary source of these slides are the notes of Dr. Stan Matwin, from the University of Ottawa. I sincerely thank him for this. The content is essentially from the book by Tom M.Mitchell, *Machine Learning*, McGraw Hill 1997.

Content of These Lectures

- Machine Learning / Data Mining: Basic Terminology
- Decision trees
- Pruning
- Testing
- Confusion Matrix
- ROC Curves
- Cost Matrix

Machine Learning / Data Mining: Basic Terminology

- Machine Learning:
  - Given a certain task
  - And a data set that constitutes the task
  - ML provides algorithms that resolve the task based on the data, and the solution improves with time
- Examples:
  - Predicting lottery numbers next Saturday
  - Detecting oil spills on sea surface
  - Assigning documents to a folder
  - Identifying likely people for a new credit card (cross selling)

Machine Learning / Data Mining: Basic Terminology

- Machine Learning:
  - Given a certain task
  - And a data set that constitutes the task
  - ML provides algorithms that resolve the task based on the data, and the solution improves with time
- Examples:
  - Predicting lottery numbers next Saturday
  - Detecting oil spills on sea surface
  - Assigning documents to a folder
  - Identifying likely people for a new credit card (cross selling)

Data Mining

- Data Mining:
  - Extracting regularities from a VERY LARGE dataset/database
  - As part of a business/application cycle
- Examples:
  - Cell phone fraud detection
  - Customer churn (loss of customers)
  - Direct mail targeting/ cross selling
  - Prediction of aircraft component failures

Data Mining

- Data Mining:
  - Extracting regularities from a VERY LARGE dataset/database
  - As part of a business/application cycle
- Examples:
  - Cell phone fraud detection
  - Customer churn (loss of customers)
  - Direct mail targeting/ cross selling
  - Prediction of aircraft component failures

Basic ML tasks

- Supervised learning:
  - Classification/Concept learning
  - Estimation: Essentially, extrapolation
- Unsupervised learning:
  - Clustering: Finding groups of “similar” objects
  - Associations: Determining, in a database, that some values of attributes go with some others

Basic ML tasks

- Supervised learning:
  - Classification/Concept learning
  - Estimation: Essentially, extrapolation
- Unsupervised learning:
  - Clustering: Finding groups of “similar” objects
  - Associations: Determining, in a database, that some values of attributes go with some others

Concept Learning: Definitions

- The concept learning problem:

Given:

A set $\mathbf{E} = \{e_1, e_2, ..., e_n\}$ of training instances of "Concepts". Each is labeled with the name of a concept $C_1, ..., C_k$ to which it belongs.

Determine:

Definitions of each of $C_1, ..., C_k$ which correctly cover $\mathbf{E}$. Each definition is a *Concept Description*.

Concept Learning: Definitions

- The concept learning problem:

Given:

A set **E** = $\{e_1, e_2, ..., e_n\}$ of training instances of "Concepts". Each is labeled with the name of a concept $C_1, ..., C_k$ to which it belongs.

Determine:

Definitions of each of $C_1, ..., C_k$ which correctly cover **E**. Each definition is a *Concept Description*.

Concept Learning: Definitions

- The concept learning problem:

Given:

A set **E** = $\{e_1, e_2, ..., e_n\}$ of training instances of "Concepts". Each is labeled with the name of a concept $C_1, ..., C_k$ to which it belongs.

Determine:

Definitions of each of $C_1, ..., C_k$ which correctly cover **E**. Each definition is a *Concept Description*.

Dimensions of concept learning

- Representation:
  - Data
    - Symbolic
    - Numeric
  - Concept Description
    - Attribute-value (propositional logic)
    - Relational (first order logic)
- Level of Learning:
  - Symbolic: Knowledge relatively easy to comprehend and relate to human knowledge
  - Sub-symbolic: Not possible to understand the knowledge (Neural networks)
- Method of Learning
  - Bottom-up (covering)
  - Top-down
  - Different search algorithms

Dimensions of concept learning

- Representation:
  - Data
    - Symbolic
    - Numeric
  - Concept Description
    - Attribute-value (propositional logic)
    - Relational (first order logic)
- Level of Learning:
  - Symbolic: Knowledge relatively easy to comprehend and relate to human knowledge
  - Sub-symbolic: Not possible to understand the knowledge (Neural networks)
- Method of Learning
  - Bottom-up (covering)
  - Top-down
  - Different search algorithms

Dimensions of concept learning

- Representation:
  - Data
    - Symbolic
    - Numeric
  - Concept Description
    - Attribute-value (propositional logic)
    - Relational (first order logic)
- Level of Learning:
  - Symbolic: Knowledge relatively easy to comprehend and relate to human knowledge
  - Sub-symbolic: Not possible to understand the knowledge (Neural networks)
- Method of Learning
  - Bottom-up (covering)
  - Top-down
  - Different search algorithms

Dimensions of concept learning

- Representation:
  - Data
    - Symbolic
    - Numeric
  - Concept Description
    - Attribute-value (propositional logic)
    - Relational (first order logic)
- Level of Learning:
  - Symbolic: Knowledge relatively easy to comprehend and relate to human knowledge
  - Sub-symbolic: Not possible to understand the knowledge (Neural networks)
- Method of Learning
  - Bottom-up (covering)
  - Top-down
  - Different search algorithms

Decision Trees: Introduction

- DT learning is a method for approximating discrete-valued target functions
- A tree can be re-presented as sets of “If-Then” rules
- Among the most popular of inductive inference algorithms

Decision Tree Representation

- Decision trees classify *instances* by sorting them down the tree from the root to some leaf nodes
- Each node in the tree specifies a test of some attribute of the instance
- Each branch descending from that node corresponds to one of the possible values for this attribute

Decision Tree Representation

- An instance is classified by:
  - Starting at the root node of the tree
  - Testing the attribute specified by this node
  - Then moving down the tree branch corresponding to the value of the attribute
  - The process is then repeated for the subtree rooted at the new node
- Decision trees represent a disjunction of conjunction of constraints on the attribute values of instances
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests
- The tree itself to a disjunction of these conjunctions

Decision Tree Representation

- An instance is classified by:
  - Starting at the root node of the tree
  - Testing the attribute specified by this node
  - Then moving down the tree branch corresponding to the value of the attribute
  - The process is then repeated for the subtree rooted at the new node
- Decision trees represent a disjunction of conjunction of constraints on the attribute values of instances
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests
- The tree itself to a disjunction of these conjunctions

Decision Trees: An Example

A DT as a “Concept Representation” for deciding to *Play Tennis*

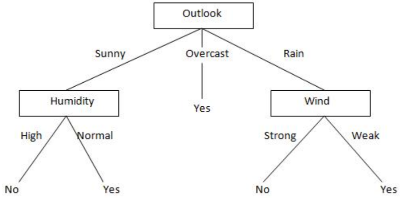


Figure: A DT for deciding when to play tennis.

- Classified example by sorting it through the tree to the appropriate leaf
- Return the classification associated with this leaf (Here: *Yes* or *No*)
- This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis

Decision Trees: An Example

A DT as a "Concept Representation" for deciding to *Play Tennis*

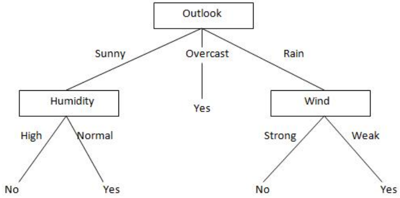


Figure: A DT for deciding when to play tennis.

- Classified example by sorting it through the tree to the appropriate leaf
- Return the classification associated with this leaf (Here: *Yes* or *No*)
- This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis

Decision Trees: An Example

A DT as a "Concept Representation" for deciding to *Play Tennis*

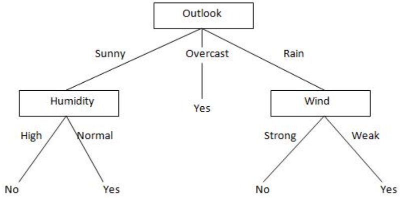
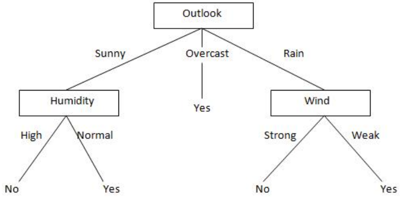


Figure: A DT for deciding when to play tennis.

- Classified example by sorting it through the tree to the appropriate leaf
- Return the classification associated with this leaf (Here: *Yes* or *No*)
- This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis

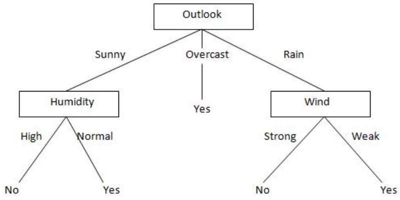
Decision Trees: Example Contd.



For example, the instance:
$\langle Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong \rangle$

- Would be sorted down the leftmost branch of this decision tree
- Therefore it will be classified as a negative instance
- Tree predicts that decision to *Play_Tennis = No*

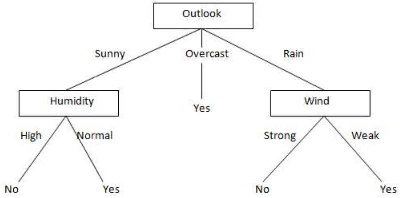
Decision Trees: Example Contd.



For example, the instance:
$\langle Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong \rangle$

- Would be sorted down the leftmost branch of this decision tree
- Therefore it will be classified as a negative instance
- Tree predicts that decision to *Play_Tennis* = *No*

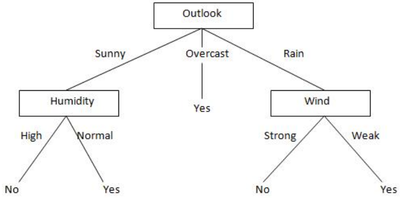
Decision Trees: Example Contd.



For example, to obtain all the *Yes* cases using disjunctions and conjunctions:

- $(Outlook = Sunny \wedge Humidity = Normal)$
- $\vee\ (Outlook = Overcast)$
- $\vee\ (Outlook = Rain \wedge Wind = Weak)$

Decision Trees: Example Contd.



For example, to obtain all the *Yes* cases using disjunctions and conjunctions:

- $(Outlook = Sunny \wedge Humidity = Normal)$
- $\vee\ (Outlook = Overcast)$
- $\vee\ (Outlook = Rain \wedge Wind = Weak)$

The Basic DT Learning Algorithm

- Usually, these algorithms are variations on a core algorithm
- Latter employs a top-down, greedy search through the space of possible DTs
- Initial query: “Which attribute should be tested at the DTs root?”
- Each attribute is evaluated using a statistical test
- This test determine how well it *alone* classifies the training examples
- The “best” attribute is selected and used as the test at the DTs root
- A descendant of the root is created for each possible value of this attribute
- Training examples are sorted to the appropriate descendant node (i.e. down the branch corresponding to the example’s value for this attribute).

The Basic DT Learning Algorithm

- Usually, these algorithms are variations on a core algorithm
- Latter employs a top-down, greedy search through the space of possible DTs
- Initial query: “Which attribute should be tested at the DTs root?”
- Each attribute is evaluated using a statistical test
- This test determine how well it *alone* classifies the training examples
- The “best” attribute is selected and used as the test at the DTs root
- A descendant of the root is created for each possible value of this attribute
- Training examples are sorted to the appropriate descendant node (i.e. down the branch corresponding to the example’s value for this attribute).

The Basic DT Learning Algorithm

- Usually, these algorithms are variations on a core algorithm
- Latter employs a top-down, greedy search through the space of possible DTs
- Initial query: “Which attribute should be tested at the DTs root?”
- Each attribute is evaluated using a statistical test
- This test determine how well it *alone* classifies the training examples
- The “best” attribute is selected and used as the test at the DTs root
- A descendant of the root is created for each possible value of this attribute
- Training examples are sorted to the appropriate descendant node (i.e. down the branch corresponding to the example’s value for this attribute).

The Basic DT Learning Algorithm

- The entire process is then repeated
- Using the training examples associated with each descendant node
- Thus select the best attribute to test at that point in the tree
- This forms a greedy search for an acceptable decision tree
- The algorithm never backtracks to reconsider earlier choices.

The Basic DT Learning Algorithm

- The entire process is then repeated
- Using the training examples associated with each descendant node
- Thus select the best attribute to test at that point in the tree
- This forms a greedy search for an acceptable decision tree
- The algorithm never backtracks to reconsider earlier choices.

Which attribute is the best classifier?

- The central issue: Which attribute to test at each node in the tree?
- Goal: Select the attribute that is “most useful” for classifying examples
- What is a good quantitative measure of the “worth” of an attribute?
- We define a statistical property, *Information Gain*
- Measures how well a given attribute separates the training examples according to their target classification

Which attribute is the best classifier?

- The central issue: Which attribute to test at each node in the tree?
- Goal: Select the attribute that is “most useful” for classifying examples
- What is a good quantitative measure of the “worth” of an attribute?
- We define a statistical property, *Information Gain*
- Measures how well a given attribute separates the training examples according to their target classification

Entropy measures: Homogeneity of examples

- Before defining information gain precisely, we introduce the concept of *Entropy*
- This characterizes the (im)purity of an arbitrary collection of examples
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this classification is: $Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$
  - where $p_{\oplus}$ is the proportion of positive examples in S, and
  - $p_{\ominus}$ is the proportion of negative examples in S
- In all calculations involving entropy, $0 \log 0$ is considered 0.

Entropy measures: Homogeneity of examples

- Before defining information gain precisely, we introduce the concept of *Entropy*
- This characterizes the (im)purity of an arbitrary collection of examples
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this classification is: $Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$
  - where $p_{\oplus}$ is the proportion of positive examples in S, and
  - $p_{\ominus}$ is the proportion of negative examples in S
- In all calculations involving entropy, $0 \log 0$ is considered 0.

Entropy measures: Homogeneity of examples

- Before defining information gain precisely, we introduce the concept of *Entropy*
- This characterizes the (im)purity of an arbitrary collection of examples
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this classification is: $Entropy(S) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$
  - where $p_{\oplus}$ is the proportion of positive examples in S, and
  - $p_{\ominus}$ is the proportion of negative examples in S
- In all calculations involving entropy, $0 \log 0$ is considered 0.

Entropy measures: Homogeneity of examples

- Suppose S a collection of 14 examples of some boolean concept
- It has 9 positive examples and 5 negative examples
- We adopt the notation [9+,5-] to summarize such a sample of data
- The entropy of S relative to this classification is : $Entropy([9+,5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940$.
- Note that the entropy is 0 if all members of S belong to the same class
- For example, if all members are positive ($p_\oplus = 1$), then $p_\ominus = 0$ and $Entropy(S) = -1\log_2 1 - 0\log_2 0 = 0$
- Note that the entropy is 1 when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Entropy measures: Homogeneity of examples

- Suppose S a collection of 14 examples of some boolean concept
- It has 9 positive examples and 5 negative examples
- We adopt the notation [9+,5-] to summarize such a sample of data
- The entropy of S relative to this classification is : $Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$ $= 0.940.$
- Note that the entropy is 0 if all members of S belong to the same class
- For example, if all members are positive ($p_\oplus = 1$), then $p_\ominus = 0$ and $Entropy(S) = -1\log_2 1 - 0\log_2 0 = 0$
- Note that the entropy is 1 when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Entropy measures: Homogeneity of examples

- Suppose S a collection of 14 examples of some boolean concept
- It has 9 positive examples and 5 negative examples
- We adopt the notation [9+,5-] to summarize such a sample of data
- The entropy of S relative to this classification is :
  $Entropy([9+,5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$
  $= 0.940.$
- Note that the entropy is 0 if all members of S belong to the same class
- For example, if all members are positive ($p_{\oplus} = 1$), then $p_{\ominus} = 0$ and
  $Entropy(S) = -1\log_2 1 - 0\log_2 0 = 0$
- Note that the entropy is 1 when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Entropy measures: Homogeneity of examples

- Suppose S a collection of 14 examples of some boolean concept
- It has 9 positive examples and 5 negative examples
- We adopt the notation [9+,5-] to summarize such a sample of data
- The entropy of S relative to this classification is :
  $Entropy([9+,5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$
  $= 0.940$.
- Note that the entropy is 0 if all members of S belong to the same class
- For example, if all members are positive ($p_\oplus = 1$), then $p_\ominus = 0$ and $Entropy(S) = -1\log_2 1 - 0\log_2 0 = 0$
- Note that the entropy is 1 when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Entropy measures: Homogeneity of examples

- Entropy has an information theoretic interpretation
- It specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S
- This is if a member of S drawn at random with uniform probability
- If $p_{\oplus} = 1$, the receiver knows the drawn example will be positive
- So no message need be sent, and the entropy is zero
- If $p_{\oplus} = 0.5$, one bit is required to indicate whether the drawn example is positive or negative
- If $p_{\oplus} = 0.8$, then a collection of messages can be encoded using on average less than 1 bit per message
- Assign shorter codes to collections of positive examples and longer codes to less likely negative examples

Entropy measures: Homogeneity of examples

- Entropy has an information theoretic interpretation
- It specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S
- This is if a member of S drawn at random with uniform probability
- If $p_{\oplus} = 1$, the receiver knows the drawn example will be positive
- So no message need be sent, and the entropy is zero
- If $p_{\oplus} = 0.5$, one bit is required to indicate whether the drawn example is positive or negative
- If $p_{\oplus} = 0.8$, then a collection of messages can be encoded using on average less than 1 bit per message
- Assign shorter codes to collections of positive examples and longer codes to less likely negative examples

Entropy measures: Homogeneity of examples

- Entropy has an information theoretic interpretation
- It specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S
- This is if a member of S drawn at random with uniform probability
- If $p_{\oplus} = 1$, the receiver knows the drawn example will be positive
- So no message need be sent, and the entropy is zero
- If $p_{\oplus} = 0.5$, one bit is required to indicate whether the drawn example is positive or negative
- If $p_{\oplus} = 0.8$, then a collection of messages can be encoded using on average less than 1 bit per message
- Assign shorter codes to collections of positive examples and longer codes to less likely negative examples

Entropy measures: Homogeneity of examples

- Entropy has an information theoretic interpretation
- It specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S
- This is if a member of S drawn at random with uniform probability
- If $p_{\oplus} = 1$, the receiver knows the drawn example will be positive
- So no message need be sent, and the entropy is zero
- If $p_{\oplus} = 0.5$, one bit is required to indicate whether the drawn example is positive or negative
- If $p_{\oplus} = 0.8$, then a collection of messages can be encoded using on average less than 1 bit per message
- Assign shorter codes to collections of positive examples and longer codes to less likely negative examples

Entropy measures: Homogeneity of examples

- More generally: if the target attribute can take on c different values
  - Entropy of S relative to this c-wise classification is defined as: $Entropy(\mathrm{S}) = \sum_{i=1}^{c} -p_i \log_2 p_i$, where p_i is the proportion of S belonging to class i
- The logarithm is still base 2
- Because entropy is a measure of the expected encoding length measured in *bits*
- If the target attribute can take on c possible values, the entropy can be as large as $\log_2 c$

Entropy measures: Homogeneity of examples

- More generally: if the target attribute can take on *c* different values
  - Entropy of *S* relative to this *c*-wise classification is defined as: $Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$, where p_i is the proportion of *S* belonging to class *i*
- The logarithm is still base 2
- Because entropy is a measure of the expected encoding length measured in *bits*
- If the target attribute can take on *c* possible values, the entropy can be as large as $\log_2 c$

Information Gain: Expected reduction in entropy

- Entropy is a measure of the impurity in a collection of training examples
- We can now define a measure of the effectiveness of an attribute in classifying the training data:
- **Information Gain**
  - Expected reduction in entropy caused by partitioning the examples according to this attribute
  - The Information Gain of an attribute A:
    $Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$
    - where $Values(A)$ is the set of all possible values for attribute A
    - S_v is the subset of S for which attribute A has value v
    - That is: $S_v = \{s \in S | A(s) = v\}$
- The first term is just the entropy of the original collection S
- The second term is the expected value of the entropy after S is partitioned using attribute A

Information Gain: Expected reduction in entropy

- Entropy is a measure of the impurity in a collection of training examples
- We can now define a measure of the effectiveness of an attribute in classifying the training data:
- **Information Gain**
  - Expected reduction in entropy caused by partitioning the examples according to this attribute
  - The Information Gain of an attribute A:
    $Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$
    - where $Values(A)$ is the set of all possible values for attribute A
    - S_v is the subset of S for which attribute A has value v
    - That is: $S_v = \{s \in S | A(s) = v\}$
- The first term is just the entropy of the original collection S
- The second term is the expected value of the entropy after S is partitioned using attribute A

Information Gain: Expected reduction in entropy

- Entropy is a measure of the impurity in a collection of training examples
- We can now define a measure of the effectiveness of an attribute in classifying the training data:
- **Information Gain**
  - Expected reduction in entropy caused by partitioning the examples according to this attribute
  - The Information Gain of an attribute A:
    $Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$
    - where $Values(A)$ is the set of all possible values for attribute A
    - S_v is the subset of S for which attribute A has value v
    - That is: $S_v = \{s \in S | A(s) = v\}$
- The first term is just the entropy of the original collection S
- The second term is the expected value of the entropy after S is partitioned using attribute A

Information Gain: Expected reduction in entropy

- Entropy is a measure of the impurity in a collection of training examples
- We can now define a measure of the effectiveness of an attribute in classifying the training data:
- **Information Gain**
  - Expected reduction in entropy caused by partitioning the examples according to this attribute
  - The Information Gain of an attribute A:
    $Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$
    - where $Values(A)$ is the set of all possible values for attribute A
    - S_v is the subset of S for which attribute A has value v
    - That is: $S_v = \{s \in S | A(s) = v\}$
- The first term is just the entropy of the original collection S
- The second term is the expected value of the entropy after S is partitioned using attribute A

Information Gain: Expected reduction in entropy

- $Gain(S, A)$ is the expected reduction in entropy by knowing A's value
- $Gain(S, A)$ is the information provided about the target function value, given the value of A
- The value of $Gain(S, A)$ is the number of bits saved
  - For encoding the target value of an arbitrary member of S
  - By knowing the value of A

Information Gain: Expected reduction in entropy

- $Gain(S, A)$ is the expected reduction in entropy by knowing A's value
- $Gain(S, A)$ is the information provided about the target function value, given the value of A
- The value of $Gain(S, A)$ is the number of bits saved
    - For encoding the target value of an arbitrary member of S
    - By knowing the value of A

Information Gain: An Example

Data for inferring to *Play* or *Not_Play* tennis depending on the *Weather*

Day	Outlook	Temp	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Ovcst	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Col	Normal	Weak	yes
6	Rain	Cool	Normal	Strong	No
7	Ovcst	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Ovcst	Mild	High	Strong	Yes
13	Ovcst	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

The Info(S) = 0.940

Information Gain: An Example (Contd.)

- Suppose *S* is a collection of 14 training-example days described by:
  - Attributes, for example, *Wind*
  - *Wind* can take the values *Weak* or *Strong*
  - On 9 days one can *Play Tennis* (*Yes*)
  - On 5 days one cannot *Play Tennis* (*No*)
  - Record this as: [9+,5-]

Information Gain: An Example (Contd.)

- Of these 14 examples:
  - Suppose 6 of the positive and 2 of the negative examples have *Wind=Weak*
  - The remainder have *Wind=Strong*
- The information gain due to sorting the original 14 examples by the attribute *Wind*:

$$Values(Wind) = Weak, Strong$$
S=[9+,5-]
$$S_{Weak} \leftarrow [6+,2-]$$
$$S_{Strong} \leftarrow [3+,3-]$$

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

Information Gain: An Example (Contd.)

- Of these 14 examples:
    - Suppose 6 of the positive and 2 of the negative examples have *Wind*=*Weak*
    - The remainder have *Wind*=*Strong*
- The information gain due to sorting the original 14 examples by the attribute *Wind*:

$$Values(Wind) = Weak, Strong$$

S=[9+,5-]

$S_{Weak} \leftarrow$ [6+,2-]

$S_{Strong} \leftarrow$ [3+,3-]

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

Information Gain: An Example (Contd.)

- Of these 14 examples:
  - Suppose 6 of the positive and 2 of the negative examples have *Wind=Weak*
  - The remainder have *Wind=Strong*
- The information gain due to sorting the original 14 examples by the attribute *Wind*:

  $Values(Wind) = Weak, Strong$
  S=[9+,5-]
  $S_{Weak} \leftarrow$ [6+,2-]
  $S_{Strong} \leftarrow$ [3+,3-]

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

Information Gain: An Example (Contd.)

- $Gain(S, Wind)$

$$= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S)$$

$$-(8/14)Entropy(S_{Weak}) - (6/14)Entropy(S_{Strong})$$

$$= 0.940 - (8/14)0.811 + (6/14)1.00 = 0.048$$

Information Gain: An Example (Contd.)

- $Gain(S, Wind)$

$$= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

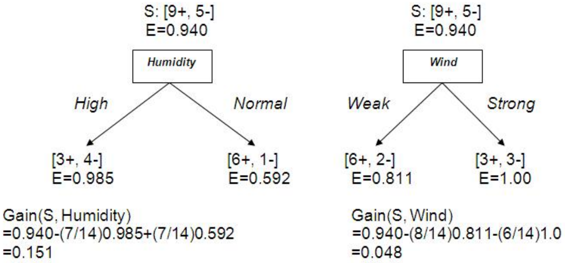
$$= Entropy(S)$$

$$-(8/14) Entropy(S_{Weak}) - (6/14) Entropy(S_{Strong})$$

$$= 0.940 - (8/14)0.811 + (6/14)1.00 = 0.048$$

Information gain: Two Attributes

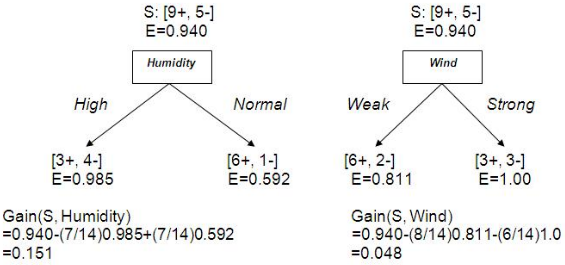
- We introduce a second attribute: *Humidity*
- The indices are:
  - Values of [3+,4-] (*Humidity*=*High*)
  - Values of [6+,1-] (*Humidity*=*Normal*)



- The information gained by this partitioning is 0.151
- Greater than 0.048 obtain for the attribute *Wind*.

Information gain: Two Attributes

- We introduce a second attribute: *Humidity*
- The indices are:
  - Values of [3+,4-] (*Humidity*=*High*)
  - Values of [6+,1-] (*Humidity*=*Normal*)



- The information gained by this partitioning is 0.151
- Greater than 0.048 obtain for the attribute *Wind*.

Selecting the Attribute

Day	Outlook	Temp	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Ovcst	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Col	Normal	Weak	yes
6	Rain	Cool	Normal	Strong	No
7	Ovcst	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Ovcst	Mild	High	Strong	Yes
13	Ovcst	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

- Gain(S, Outlook) = 0.246;
- Gain(S, Humidity) = 0.151;
- Gain(S, Wind) = 0.048;
- Gain(S, Temp) = 0.029

- Gain(S, Outlook) = 0.246;
- Choose **Outlook** as the top test - the best predictor
- Branches are created below the root for each possible values
- (i.e., *Sunny*, *Overcast*, and *Rain*)

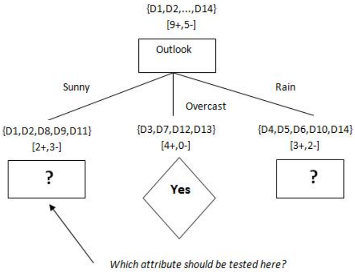
Selecting the Attribute

Day	Outlook	Temp	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Ovcst	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Col	Normal	Weak	yes
6	Rain	Cool	Normal	Strong	No
7	Ovcst	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Ovcst	Mild	High	Strong	Yes
13	Ovcst	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

- Gain(S, Outlook) = 0.246;
- Gain(S, Humidity) = 0.151;
- Gain(S, Wind) = 0.048;
- Gain(S, Temp) = 0.029

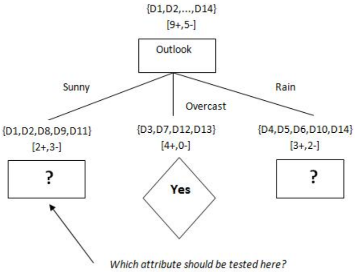
- Gain(S, Outlook) = 0.246;
- Choose **Outlook** as the top test - the best predictor
- Branches are created below the root for each possible values
- (i.e., *Sunny*, *Overcast*, and *Rain*)

Information Gain: Next Steps



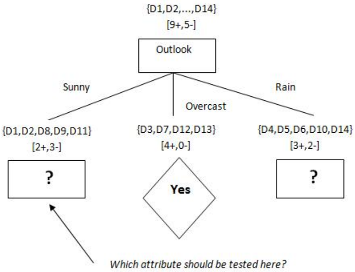
- The **Overcast** descendant has only positive examples (entropy zero)
- Therefore becomes a leaf node with classification *Yes*
- The other nodes will be further expanded
- Select the attribute with highest information gain
- Relative to the new subset of examples

Information Gain: Next Steps



- The **Overcast** descendant has only positive examples (entropy zero)
- Therefore becomes a leaf node with classification *Yes*
- The other nodes will be further expanded
- Select the attribute with highest information gain
- Relative to the new subset of examples

Information Gain: Next Steps



- The **Overcast** descendant has only positive examples (entropy zero)
- Therefore becomes a leaf node with classification *Yes*
- The other nodes will be further expanded
- Select the attribute with highest information gain
- Relative to the new subset of examples

Information Gain: Next Steps

- Repeat for each nonterminal descendant node this process
- Select a new attribute and partition the training examples
- Each time use only examples associated with that node
- Attributes incorporated higher in the tree are excluded
- Any given attribute can appear at most once along any path in the tree
- Process continues for each new leaf node until:
  - Either every attribute has already been included along this path through the tree
  - Or the training examples associated with this leaf node all have the same target attribute value (i.e. entropy zero)

Information Gain: Next Steps

- Repeat for each nonterminal descendant node this process
- Select a new attribute and partition the training examples
- Each time use only examples associated with that node
- Attributes incorporated higher in the tree are excluded
- Any given attribute can appear at most once along any path in the tree
- Process continues for each new leaf node until:
  - Either every attribute has already been included along this path through the tree
  - Or the training examples associated with this leaf node all have the same target attribute value (i.e. entropy zero)

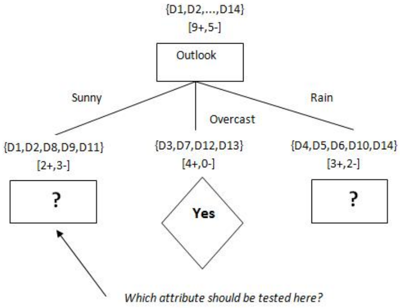
Information Gain: Next Steps

- Repeat for each nonterminal descendant node this process
- Select a new attribute and partition the training examples
- Each time use only examples associated with that node
- Attributes incorporated higher in the tree are excluded
- Any given attribute can appear at most once along any path in the tree
- Process continues for each new leaf node until:
  - Either every attribute has already been included along this path through the tree
  - Or the training examples associated with this leaf node all have the same target attribute value (i.e. entropy zero)

Information Gain: Next Steps

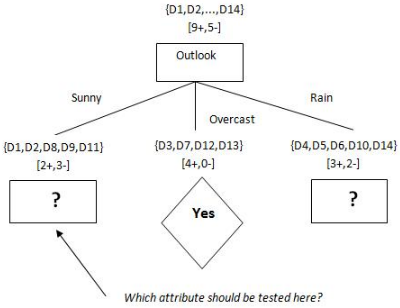
- Repeat for each nonterminal descendant node this process
- Select a new attribute and partition the training examples
- Each time use only examples associated with that node
- Attributes incorporated higher in the tree are excluded
- Any given attribute can appear at most once along any path in the tree
- Process continues for each new leaf node until:
    - Either every attribute has already been included along this path through the tree
    - Or the training examples associated with this leaf node all have the same target attribute value (i.e. entropy zero)

Information Gain: Second Iteration



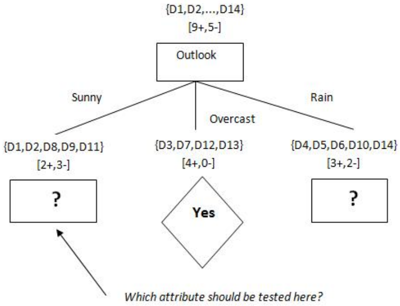
- S_{Sunny} = {D1, D2, D8, D9, D11}
- Gain(S_{Sunny}, Humidity) = 0.970-(3/5)*0.0 -(2/5)*0.0=**0.970**
- Gain(S_{Sunny}, Temp) = 0.970-(2/5)*0.0 -(2/5)*1.0-(1/5)*0.0=0.570
- Gain(S_{Sunny}, Wind) = 0.970-(2/5)*1.0 -(3/5)*0.918=0.019

Information Gain: Second Iteration



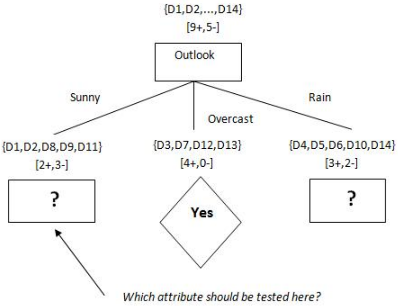
- S_{Sunny} = {D1, D2, D8, D9, D11}
- Gain(S_{Sunny}, Humidity) = 0.970-(3/5)*0.0 -(2/5)*0.0=**0.970**
- Gain(S_{Sunny}, Temp) = 0.970-(2/5)*0.0 -(2/5)*1.0-(1/5)*0.0=0.570
- Gain(S_{Sunny}, Wind) = 0.970-(2/5)*1.0 -(3/5)*0.918=0.019

Information Gain: Second Iteration



- S_{Sunny} = {D1, D2, D8, D9, D11}
- Gain(S_{Sunny}, Humidity) = 0.970-(3/5)*0.0 -(2/5)*0.0=**0.970**
- Gain(S_{Sunny}, Temp) = 0.970-(2/5)*0.0 -(2/5)*1.0-(1/5)*0.0=0.570
- Gain(S_{Sunny}, Wind) = 0.970-(2/5)*1.0 -(3/5)*0.918=0.019

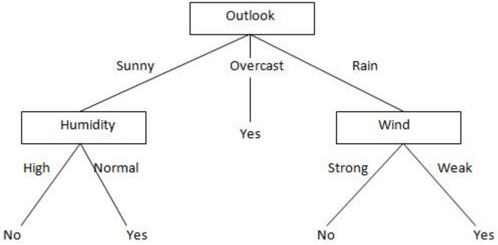
Information Gain: Second Iteration



- S_{Sunny} = {D1, D2, D8, D9, D11}
- Gain(S_{Sunny}, Humidity) = 0.970-(3/5)*0.0 -(2/5)*0.0=**0.970**
- Gain(S_{Sunny}, Temp) = 0.970-(2/5)*0.0 -(2/5)*1.0-(1/5)*0.0=0.570
- Gain(S_{Sunny}, Wind) = 0.970-(2/5)*1.0 -(3/5)*0.918=0.019

Decision Tree: Final Tree

The final decision tree for our example:



Partition of Cases and the DT

Partition of cases

Outlook = Sunny:

Humidity ≤ 75

Outlook	Temp (°F)	Humidity (%)	Windy ?	Decision
Sunny	75	70	true	Play
Sunny	69	70	false	Play

Humidity > 75

Outlook	Temp (°F)	Humidity (%)	Windy ?	Decision
Sunny	80	90	true	Don't Play
Sunny	85	85	false	Don't Play
Sunny	72	95	false	Don't Play

Outlook = Overcast:

Outlook	Temp (°F)	Humidity (%)	Windy ?	Decision
Overcast	72	90	true	Play
Overcast	83	78	false	Play
Overcast	64	65	true	Play
Overcast	81	75	false	Play

Outlook = Rain:

Humidity = true

Outlook	Temp (°F)	Humidity (%)	Windy ?	Decision
Rain	71	80	true	Don't Play
Rain	65	70	true	Don't Play

Windy=false

Outlook	Temp (°F)	Humidity (%)	Windy ?	Decision
Rain	75	80	false	Play
Rain	68	80	false	Play
Rain	70	80	false	Play

Corresponding decision tree:

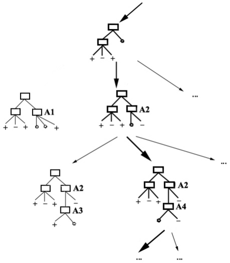
Outlook=sunny:
- Humidity ≤ 75: **Play**
- Humidity > 75 : **Don't Play**

Outlook=overcast : **Play**

Outlook=rain:
- Windy=true: **Don't Play**
- Windy = false: **Play**

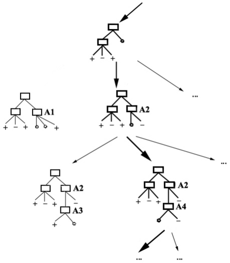
Partition of cases and the DT

- Learning DTs with the gain ratio heuristic
- Simple-to-complex, hill-climbing search through this hypothesis space
- Beginning with the empty tree
- Proceed progressively to more elaborate hypothesis in DT space
- Goal: Correctly classify the training data.
- Evaluation Fn. that guides this hill-climbing search: Information gain



Partition of cases and the DT

- Learning DTs with the gain ratio heuristic
- Simple-to-complex, hill-climbing search through this hypothesis space
- Beginning with the empty tree
- Proceed progressively to more elaborate hypothesis in DT space
- Goal: Correctly classify the training data.
- Evaluation Fn. that guides this hill-climbing search: Information gain



Continuous Attributes

- A simple trick:
  - Sort the values of each continuous attribute
  - Choose the midpoint between each two consecutive values
  - For m values, there are $m - 1$ possible splits
  - Examine them linearly
- What is the Cost of this “trick”?

Overfitting and Pruning

- What is over overfitting?
- Why prefer shorter hypothesis?
- Occam's Razor (1930): Prefer the simplest hypothesis that fits the data
- Many complex hypothesis that fit the current training data
- But fail to generalize correctly to subsequent data
- Algorithm will try to "learn the noise" (there is noise in data).
- Overfitting: Hypothesis overfits the training examples if:
  - Some other hypothesis that fits the training examples "worse"
  - BUT aactually performs better over the entire distribution of instances
  - That is: including instances beyond the training set

Overfitting and Pruning

- What is over overfitting?
- Why prefer shorter hypothesis?
- Occam's Razor (1930): Prefer the simplest hypothesis that fits the data
- Many complex hypothesis that fit the current training data
- But fail to generalize correctly to subsequent data
- Algorithm will try to "learn the noise" (there is noise in data).
- Overfitting: Hypothesis overfits the training examples if:
  - Some other hypothesis that fits the training examples "worse"
  - BUT aactually performs better over the entire distribution of instances
  - That is: including instances beyond the training set

Overfitting and Pruning

- What is over overfitting?
- Why prefer shorter hypothesis?
- Occam's Razor (1930): Prefer the simplest hypothesis that fits the data
- Many complex hypothesis that fit the current training data
- But fail to generalize correctly to subsequent data
- Algorithm will try to "learn the noise" (there is noise in data).
- Overfitting: Hypothesis overfits the training examples if:
  - Some other hypothesis that fits the training examples "worse"
  - BUT aactually performs better over the entire distribution of instances
  - That is: including instances beyond the training set

Pruning : Avoiding overfitting

- Two classes of approaches to avoid overfitting in DT learning:
  - Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
  - Allow the tree to overfit the data, and then post-prune the tree

Pruning : Avoiding overfitting

- Two classes of approaches to avoid overfitting in DT learning:
  - Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
  - Allow the tree to overfit the data, and then post-prune the tree

Pruning : Avoiding overfitting

- Two classes of approaches to avoid overfitting in DT learning:
  - Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
  - Allow the tree to overfit the data, and then post-prune the tree

Pruning

Reduced-error pruning: Consider each decision node in the tree to be candidate for pruning.

- Pruning a decision node
- Removing the subtree rooted at that node: Make it a leaf node
- Assign it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set
- Effect: Leaf node added due to coincidental training-set regularities - likely to be pruned
- Because same coincidences: Unlikely to occur in the validation set
- Nodes are pruned iteratively
- Choose node whose removal most increases accuracy over the validation set
- Pruning of nodes continues until further pruning is harmful
- That is: Decreases accuracy of the tree over the validation set

Pruning

Reduced-error pruning: Consider each decision node in the tree to be candidate for pruning.

- Pruning a decision node
- Removing the subtree rooted at that node: Make it a leaf node
- Assign it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set
- Effect: Leaf node added due to coincidental training-set regularities - likely to be pruned
- Because same coincidences: Unlikely to occur in the validation set
- Nodes are pruned iteratively
- Choose node whose removal most increases accuracy over the validation set
- Pruning of nodes continues until further pruning is harmful
- That is: Decreases accuracy of the tree over the validation set

Pruning

Reduced-error pruning: Consider each decision node in the tree to be candidate for pruning.

- Pruning a decision node
- Removing the subtree rooted at that node: Make it a leaf node
- Assign it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set
- Effect: Leaf node added due to coincidental training-set regularities - likely to be pruned
- Because same coincidences: Unlikely to occur in the validation set
- Nodes are pruned iteratively
- Choose node whose removal most increases accuracy over the validation set
- Pruning of nodes continues until further pruning is harmful
- That is: Decreases accuracy of the tree over the validation set

Pruning

Reduced-error pruning: Consider each decision node in the tree to be candidate for pruning.

- Pruning a decision node
- Removing the subtree rooted at that node: Make it a leaf node
- Assign it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set
- Effect: Leaf node added due to coincidental training-set regularities - likely to be pruned
- Because same coincidences: Unlikely to occur in the validation set
- Nodes are pruned iteratively
- Choose node whose removal most increases accuracy over the validation set
- Pruning of nodes continues until further pruning is harmful
- That is: Decreases accuracy of the tree over the validation set

Pruning

Reduced-error pruning: Consider each decision node in the tree to be candidate for pruning.

- Pruning a decision node
- Removing the subtree rooted at that node: Make it a leaf node
- Assign it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set
- Effect: Leaf node added due to coincidental training-set regularities - likely to be pruned
- Because same coincidences: Unlikely to occur in the validation set
- Nodes are pruned iteratively
- Choose node whose removal most increases accuracy over the validation set
- Pruning of nodes continues until further pruning is harmful
- That is: Decreases accuracy of the tree over the validation set

Pruning - Contd.

- How can we predict the err rate?
- Either put aside part of the training set for that purpose,
- Or apply "Crossvalidation":
  - Divide the training data into C equal-sized blocks
  - For each block: Construct a tree from testing example's in $C - 1$ remaining blocks
  - Tested on the "reserved" block

Pruning - Contd.

- How can we predict the err rate?
- Either put aside part of the training set for that purpose,
- Or apply “Crossvalidation”:
  - Divide the training data into C equal-sized blocks
  - For each block: Construct a tree from testing example’s in $C - 1$ remaining blocks
  - Tested on the “reserved” block

From Trees to Rules

- One of the most expressive and human readable representations for learned hypotheses is *set of if-then rules*.
- To translate a DT into set of rules
- Traverse the DT from root to leaf to get a rule
- With the path conditions as the antecedent and the leaf as the class
- Rule sets for a whole class
  - Simplified by removing rules
  - Those that do not contribute to the accuracy of the whole set

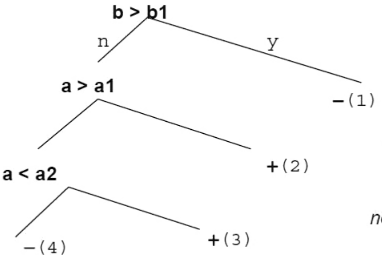
From Trees to Rules

- One of the most expressive and human readable representations for learned hypotheses is *set of if-then rules*.
- To translate a DT into set of rules
- Traverse the DT from root to leaf to get a rule
- With the path conditions as the antecedent and the leaf as the class
- Rule sets for a whole class
  - Simplified by removing rules
  - Those that do not contribute to the accuracy of the whole set

From Trees to Rules

- One of the most expressive and human readable representations for learned hypotheses is *set of if-then rules*.
- To translate a DT into set of rules
- Traverse the DT from root to leaf to get a rule
- With the path conditions as the antecedent and the leaf as the class
- Rule sets for a whole class
    - Simplified by removing rules
    - Those that do not contribute to the accuracy of the whole set

Decision Rules: From DTs



(1) **if b>b1 then class is -**

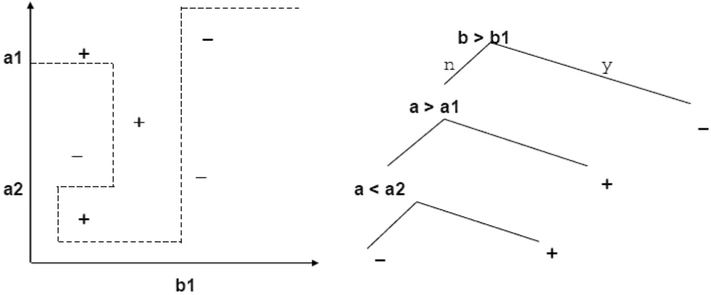
(2) **if b <= b1 and a > a1 then class is +**

(3) **if b <= b1 a < a2 then class is +**

(4) **if b <= b1 and a2 <= a <= a1 then class is -**

notice the inference involved in rule (3)

Geometric interpretation of DTs: Axis-parallel area



Empirical Evaluation of Accuracy

The usual approach:

- Partition the set E of all labeled examples
- Into a training set and a testing set.
- Use the training set for learning: Obtain a hypothesis H
  - Set $acc := 0$.
  - For each element t of the testing set, apply H on t
  - If $H(t) = label(t)$ then $acc := acc + 1$
  - $acc := acc/|testing\ set|$

Empirical Evaluation of Accuracy

The usual approach:

- Partition the set E of all labeled examples
- Into a training set and a testing set.
- Use the training set for learning: Obtain a hypothesis H
    - Set $acc := 0$.
    - For each element t of the testing set, apply H on t
    - If $H(t) = label(t)$ then $acc := acc + 1$
    - $acc := acc/|testing\ set|$

Empirical Evaluation of Accuracy

The usual approach:

- Partition the set E of all labeled examples
- Into a training set and a testing set.
- Use the training set for learning: Obtain a hypothesis H
    - Set $acc := 0$.
    - For each element t of the testing set, apply H on t
    - If $H(t) = label(t)$ then $acc := acc + 1$
    - $acc := acc/|testing\ set|$

Empirical Evaluation of Accuracy

The usual approach:

- Partition the set E of all labeled examples
- Into a training set and a testing set.
- Use the training set for learning: Obtain a hypothesis H
  - Set $acc := 0$.
  - For each element t of the testing set, apply H on t
  - If $H(t) = label(t)$ then $acc := acc + 1$
  - $acc := acc/|testing\ set|$

Testing

The usual approach:

- Given a dataset, how to split to Training/Testing sets?
- Cross-validation (n-fold)
  - Partition E into n (usually, $n = 3, 5, 10$) groups
  - Choose $n - 1$ groups from n
  - Perform learning on their union
  - Repeat the choice n times
  - Average the n results;
- Another approach: "Leave One Out"
  - Learn on all but one example
  - Test that example

Testing

The usual approach:

- Given a dataset, how to split to Training/Testing sets?
- Cross-validation (n-fold)
  - Partition E into n (usually, $n = 3, 5, 10$) groups
  - Choose $n - 1$ groups from n
  - Perform learning on their union
  - Repeat the choice n times
  - Average the n results;
- Another approach: "Leave One Out"
  - Learn on all but one example
  - Test that example

Testing

The usual approach:

- Given a dataset, how to split to Training/Testing sets?
- Cross-validation (n-fold)
  - Partition E into n (usually, $n = 3, 5, 10$) groups
  - Choose $n - 1$ groups from n
  - Perform learning on their union
  - Repeat the choice n times
  - Average the n results;
- Another approach: "Leave One Out"
  - Learn on all but one example
  - Test that example

Testing

The usual approach:

- Given a dataset, how to split to Training/Testing sets?
- Cross-validation (n-fold)
  - Partition E into n (usually, $n = 3, 5, 10$) groups
  - Choose $n - 1$ groups from n
  - Perform learning on their union
  - Repeat the choice n times
  - Average the n results;
- Another approach: “Leave One Out”
  - Learn on all but one example
  - Test that example

Testing

The usual approach:

- Given a dataset, how to split to Training/Testing sets?
- Cross-validation (n-fold)
  - Partition E into n (usually, $n = 3, 5, 10$) groups
  - Choose $n - 1$ groups from n
  - Perform learning on their union
  - Repeat the choice n times
  - Average the n results;
- Another approach: “Leave One Out”
  - Learn on all but one example
  - Test that example

Confusion Matrix

Confusion Matrix

	Classifier-Decision Positive Label	Classifier-Decision Negative Label
True Positive label	a	b
True Negative label	c	d

- a = True Positives
- b = False Negatives
- c = False Positives
- d = True Negatives
- Accuracy $= \frac{a+d}{a+b+c+d}$

Confusion Matrix

Confusion Matrix

	Classifier-Decision Positive Label	Classifier-Decision Negative Label
True Positive label	a	b
True Negative label	c	d

- a = True Positives
- b = False Negatives
- c = False Positives
- d = True Negatives
- Accuracy $= \frac{a+d}{a+b+c+d}$

Confusion Matrix - Related measures

- Precision = $\frac{a}{a+c}$;
- Recall = $\frac{a}{a+b}$;
- F-measure combines Recall and Precision: $F_\beta = \frac{(\beta^2+1)*P*R}{\beta^2 P+R}$
- Reflects importance of Recall versus Precision(e.g., $F_0 = P$)

Confusion Matrix - Related measures

- Precision = $\frac{a}{a+c}$;
- Recall = $\frac{a}{a+b}$;
- F-measure combines Recall and Precision: $F_\beta = \frac{(\beta^2+1)*P*R}{\beta^2 P+R}$
- Reflects importance of Recall versus Precision(e.g., $F_0 = P$)

Confusion Matrix - Related measures

- Precision = $\frac{a}{a+c}$;
- Recall = $\frac{a}{a+b}$;
- F-measure combines Recall and Precision:
  $F_\beta = \frac{(\beta^2+1)*P*R}{\beta^2 P+R}$
- Reflects importance of Recall versus Precision(e.g., $F_0 = P$)

Confusion Matrix - Related measures

- Precision = $\frac{a}{a+c}$;
- Recall = $\frac{a}{a+b}$;
- F-measure combines Recall and Precision:
  $F_\beta = \frac{(\beta^2+1)*P*R}{\beta^2 P+R}$
- Reflects importance of Recall versus Precision(e.g., $F_0 = P$)

ROC Curves

- ROC = Receiver Operating Characteristics
- Not as sensitive to imbalanced classes as accuracy
- Allows a choice of a classifier with desired characteristics
- Uses False Positive (FP), True Positive (TP) rate:
  - $TP = \frac{a}{a+b}$ [$p(Y|p)$ Proportion of +ve examples correctly identified]
  - $FP = \frac{c}{c+d}$ [$p(Y|n)$ Proportion of -ve examples incorrectly classified as +ve]

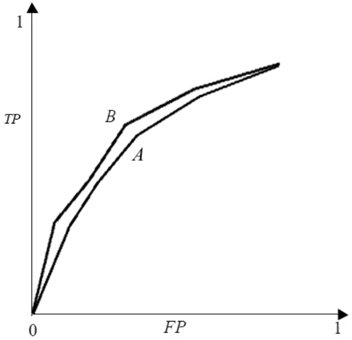
ROC Curves

- ROC = Receiver Operating Characteristics
- Not as sensitive to imbalanced classes as accuracy
- Allows a choice of a classifier with desired characteristics
- Uses False Positive (*FP*), True Positive (*TP*) rate:
  - $TP = \frac{a}{a+b}$ [$p(Y|p)$ Proportion of +ve examples correctly identified]
  - $FP = \frac{c}{c+d}$ [$p(Y|n)$ Proportion of -ve examples incorrectly classified as +ve]

ROC Curves

- ROC = Receiver Operating Characteristics
- Not as sensitive to imbalanced classes as accuracy
- Allows a choice of a classifier with desired characteristics
- Uses False Positive (*FP*), True Positive (*TP*) rate:
  - $TP = \frac{a}{a+b}$ [$p(Y|p)$ Proportion of +ve examples correctly identified]
  - $FP = \frac{c}{c+d}$ [$p(Y|n)$ Proportion of -ve examples incorrectly classified as +ve]

ROC Curves: Contd.



B is better than A

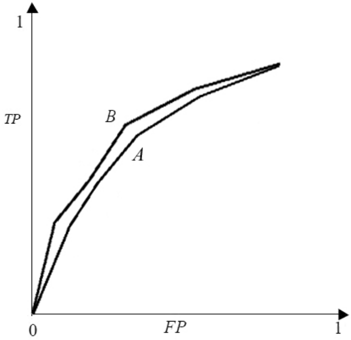
(1,0) misclassifies everything

(1,1) classifies all as +

(0,1) is the best

- ROC represents info. from the Confusion Matrix
- ROC is obtained by parameterizing a classifier (e.g. with a threshold)
- Plotting a point on the *TP*, *FP* axes for that point

ROC Curves: Contd.



B is better than A

(1,0) misclassifies everything

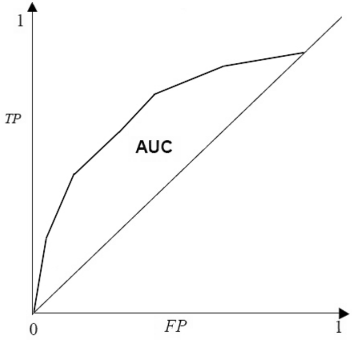
(1,1) classifies all as +

(0,1) is the best

- ROC represents info. from the Confusion Matrix
- ROC is obtained by parameterizing a classifier (e.g. with a threshold)
- Plotting a point on the *TP*, *FP* axes for that point

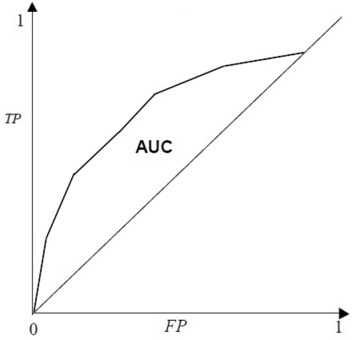
ROC Curves: Contd.

- How does a random “classifier” (default) look?
- Often we characterize a classifier with the area under curve (AUC)



ROC Curves: Contd.

- How does a random “classifier” (default) look?
- Often we characterize a classifier with the area under curve (AUC)



Cost Matrix

- Like Confusion Matrix
- Except costs of errors are assigned to the off-diagonal elements (i.e., the misclassifications)
- This may be important in applications, e.g. a diagnosis rule.
- For a survey of learning with misclassification costs see http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html

Cost Matrix

- Like Confusion Matrix
- Except costs of errors are assigned to the off-diagonal elements (i.e., the misclassifications)
- This may be important in applications, e.g. a diagnosis rule.
- For a survey of learning with misclassification costs see http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html

Cost Matrix

- Like Confusion Matrix
- Except costs of errors are assigned to the off-diagonal elements (i.e., the misclassifications)
- This may be important in applications, e.g. a diagnosis rule.
- For a survey of learning with misclassification costs see http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html