Decision trees, entropy, information gain, ID3

Relevant Readings: Sections 3.1 through 3.6 in Mitchell

CS495 - Machine Learning, Fall 2009

Decision trees

- *Decision trees* are commonly used in learning algorithms

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.
- Every leaf node has a label "yes" or "no", which is used to classify test instances that end up there

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.
- Every leaf node has a label "yes" or "no", which is used to classify test instances that end up there
- Do example using lakes data

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.
- Every leaf node has a label "yes" or "no", which is used to classify test instances that end up there
- Do example using lakes data
- If someone gave you a decision tree, could you give a boolean function in Disjunctive Normal Form (DNF) that represents it?

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.
- Every leaf node has a label "yes" or "no", which is used to classify test instances that end up there
- Do example using lakes data
- If someone gave you a decision tree, could you give a boolean function in Disjunctive Normal Form (DNF) that represents it?
- If someone gave you a boolean function in DNF, could you give a decision tree that represents it?

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.
- Every leaf node has a label "yes" or "no", which is used to classify test instances that end up there
- Do example using lakes data
- If someone gave you a decision tree, could you give a boolean function in Disjunctive Normal Form (DNF) that represents it?
- If someone gave you a boolean function in DNF, could you give a decision tree that represents it?
- A decision tree can be used to represent any boolean function on discrete attributes

Decision trees

- *Decision trees* are commonly used in learning algorithms
- For simplicity, we will restrict to discrete-valued attributes and boolean target functions in this discussion
- In a decision tree, every internal node v considers some attribute a of the given test instance
  - Each child of v corresponds to some value that a can take on; v just passes the instance to the appropriate child.
- Every leaf node has a label "yes" or "no", which is used to classify test instances that end up there
- Do example using lakes data
- If someone gave you a decision tree, could you give a boolean function in Disjunctive Normal Form (DNF) that represents it?
- If someone gave you a boolean function in DNF, could you give a decision tree that represents it?
- A decision tree can be used to represent any boolean function on discrete attributes

How do we build the tree?

- In the simplest variant, we will start with a root node and recursively add child nodes until we fit the training data perfectly

How do we build the tree?

- In the simplest variant, we will start with a root node and recursively add child nodes until we fit the training data perfectly
- The only question is which attribute to use first

How do we build the tree?

- In the simplest variant, we will start with a root node and recursively add child nodes until we fit the training data perfectly
- The only question is which attribute to use first
- Q: We want to use the attribute that does the "best job" splitting up the training data, but how can this be measured?

How do we build the tree?

- In the simplest variant, we will start with a root node and recursively add child nodes until we fit the training data perfectly
- The only question is which attribute to use first
- Q: We want to use the attribute that does the "best job" splitting up the training data, but how can this be measured?
- A: We will use *entropy* and in particular, *information gain*

How do we build the tree?

- In the simplest variant, we will start with a root node and recursively add child nodes until we fit the training data perfectly
- The only question is which attribute to use first
- Q: We want to use the attribute that does the "best job" splitting up the training data, but how can this be measured?
- A: We will use *entropy* and in particular, *information gain*

Entropy

- *Entropy* is a measure of disorder or impurity

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)
- If the output values were all positive (or negative), the set is quite orderly (pure)

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)
- If the output values were all positive (or negative), the set is quite orderly (pure)
  - The entropy is 0 (average of 0 bits of information to encode an output value)

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)
- If the output values were all positive (or negative), the set is quite orderly (pure)
  - The entropy is 0 (average of 0 bits of information to encode an output value)
- If the output values are split 25%-75%, then the entropy turns out to be about .811

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)
- If the output values were all positive (or negative), the set is quite orderly (pure)
  - The entropy is 0 (average of 0 bits of information to encode an output value)
- If the output values are split 25%-75%, then the entropy turns out to be about .811
  - Let S be a set, let p be the fraction of positive training examples, and q be the fraction of negative training examples

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)
- If the output values were all positive (or negative), the set is quite orderly (pure)
  - The entropy is 0 (average of 0 bits of information to encode an output value)
- If the output values are split 25%-75%, then the entropy turns out to be about .811
  - Let S be a set, let p be the fraction of positive training examples, and q be the fraction of negative training examples
  - *Definition:* $\mathit{Entropy}(S) = -p \log_2(p) - q \log_2(q)$

Entropy

- *Entropy* is a measure of disorder or impurity
- In our case, we will be interested in the entropy of the output values of a set of training instances
- Entropy can be understood as the average number of bits needed to encode an output value
- If the output values are split 50%-50%, then the set is disorderly (impure)
  - The entropy is 1 (average of 1 bit of information to encode an output value)
- If the output values were all positive (or negative), the set is quite orderly (pure)
  - The entropy is 0 (average of 0 bits of information to encode an output value)
- If the output values are split 25%-75%, then the entropy turns out to be about .811
  - Let S be a set, let p be the fraction of positive training examples, and q be the fraction of negative training examples
  - *Definition:* $Entropy(S) = -p \log_2(p) - q \log_2(q)$

Entropy

- Entropy can be generalized from boolean to discrete-valued target functions

Entropy

- Entropy can be generalized from boolean to discrete-valued target functions
  - The idea is the same: entropy is the average number of bits needed to encode an output value

Entropy

- Entropy can be generalized from boolean to discrete-valued target functions
  - The idea is the same: entropy is the average number of bits needed to encode an output value
  - Let S be a set of instances, and let p_i be the fraction of instances in S with output value i

Entropy

- Entropy can be generalized from boolean to discrete-valued target functions
  - The idea is the same: entropy is the average number of bits needed to encode an output value
  - Let S be a set of instances, and let p_i be the fraction of instances in S with output value i
  - *Definition:* $Entropy(S) = -\sum_i p_i \log_2(p_i)$

Entropy

- Entropy can be generalized from boolean to discrete-valued target functions
  - The idea is the same: entropy is the average number of bits needed to encode an output value
  - Let S be a set of instances, and let p_i be the fraction of instances in S with output value i
  - *Definition:* $Entropy(S) = -\sum_i p_i \log_2(p_i)$

Information gain

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets

Information gain

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets
- *Information gain* is a measure of this change in entropy

Information gain

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets
- *Information gain* is a measure of this change in entropy
- *Definition:* Suppose S is a set of instances, A is an attribute, S_v is the subset of S with $A = v$, and $\mathrm{Values}(A)$ is the set of all possible values of A, then
  $Gain(S, A) = Entropy(S) - \sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$
  (recall $|S|$ denotes the size of set S)

Information gain

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets
- *Information gain* is a measure of this change in entropy
- *Definition:* Suppose S is a set of instances, A is an attribute, S_v is the subset of S with $A = v$, and $\mathrm{Values}(A)$ is the set of all possible values of A, then $Gain(S, A) = Entropy(S) - \sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$ (recall $|S|$ denotes the size of set S)
- If you just count the number of bits of information everywhere, you end up with something equivalent:

Information gain

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets
- *Information gain* is a measure of this change in entropy
- *Definition:* Suppose S is a set of instances, A is an attribute, S_v is the subset of S with $A = v$, and $\mathrm{Values}(A)$ is the set of all possible values of A, then $Gain(S, A) = Entropy(S) - \sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$ (recall $|S|$ denotes the size of set S)
- If you just count the number of bits of information everywhere, you end up with something equivalent:
- Total gain in bits $= |S| \cdot Gain(S, A) =$ $|S| \cdot Entropy(S) - \sum_{v \in \mathrm{Values}(A)} |S_v| \cdot Entropy(S_v)$

Information gain

- The entropy typically changes when we use a node in a decision tree to partition the training instances into smaller subsets
- *Information gain* is a measure of this change in entropy
- *Definition:* Suppose S is a set of instances, A is an attribute, S_v is the subset of S with $A = v$, and $\mathrm{Values}(A)$ is the set of all possible values of A, then
  $Gain(S, A) = Entropy(S) - \sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$
  (recall $|S|$ denotes the size of set S)
- If you just count the number of bits of information everywhere, you end up with something equivalent:
- Total gain in bits $= |S| \cdot Gain(S, A) =$
  $|S| \cdot Entropy(S) - \sum_{v \in \mathrm{Values}(A)} |S_v| \cdot Entropy(S_v)$

ID3

- The essentials:

ID3

- The essentials:
  - Start with all training instances associated with the root node

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:
  - If all positive or all negative training instances remain, label that node "yes" or "no" accordingly

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:
  - If all positive or all negative training instances remain, label that node "yes" or "no" accordingly
  - If no attributes remain, label with a majority vote of training instances left at that node

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:
  - If all positive or all negative training instances remain, label that node "yes" or "no" accordingly
  - If no attributes remain, label with a majority vote of training instances left at that node
  - If no instances remain, label with a majority vote of the parent's training instances

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:
  - If all positive or all negative training instances remain, label that node "yes" or "no" accordingly
  - If no attributes remain, label with a majority vote of training instances left at that node
  - If no instances remain, label with a majority vote of the parent's training instances
- The pseudocode:

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:
  - If all positive or all negative training instances remain, label that node "yes" or "no" accordingly
  - If no attributes remain, label with a majority vote of training instances left at that node
  - If no instances remain, label with a majority vote of the parent's training instances
- The pseudocode:
  - Table 3.1 on page 56 in Mitchell

ID3

- The essentials:
  - Start with all training instances associated with the root node
  - Use info gain to choose which attribute to label each node with
    - No root-to-leaf path should contain the same discrete attribute twice
  - Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree
- The border cases:
  - If all positive or all negative training instances remain, label that node "yes" or "no" accordingly
  - If no attributes remain, label with a majority vote of training instances left at that node
  - If no instances remain, label with a majority vote of the parent's training instances
- The pseudocode:
  - Table 3.1 on page 56 in Mitchell

Notes on ID3

- Inductive bias

Notes on ID3

- Inductive bias
    - Shorter trees are preferred over longer trees (but size is not necessarily minimized)

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast
  - Classifying new examples is very fast

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast
  - Classifying new examples is very fast
- Negatives

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast
  - Classifying new examples is very fast
- Negatives
  - Difficult to extend to real-valued target functions

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast
  - Classifying new examples is very fast
- Negatives
  - Difficult to extend to real-valued target functions
  - Need to adapt the algorithm to continuous attributes

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast
  - Classifying new examples is very fast
- Negatives
  - Difficult to extend to real-valued target functions
  - Need to adapt the algorithm to continuous attributes
  - The version of ID3 discussed above can overfit the data

Notes on ID3

- Inductive bias
  - Shorter trees are preferred over longer trees (but size is not necessarily minimized)
    - Note that this is a form of *Ockham's Razor*
  - High information gain attributes are placed nearer to the root of the tree
- Positives
  - Robust to errors in the training data
  - Training is reasonably fast
  - Classifying new examples is very fast
- Negatives
  - Difficult to extend to real-valued target functions
  - Need to adapt the algorithm to continuous attributes
  - The version of ID3 discussed above can overfit the data

Avoiding overfitting

- Some possibilities

Avoiding overfitting

- Some possibilities
  - Use a tuning set to decide when to stop growing the tree

Avoiding overfitting

- Some possibilities
  - Use a tuning set to decide when to stop growing the tree
  - Use a tuning set to prune the tree after it's completely grown

Avoiding overfitting

- Some possibilities
  - Use a tuning set to decide when to stop growing the tree
  - Use a tuning set to prune the tree after it's completely grown
    - Keep removing nodes as long as it doesn't hurt performance on the tuning set

Avoiding overfitting

- Some possibilities
  - Use a tuning set to decide when to stop growing the tree
  - Use a tuning set to prune the tree after it's completely grown
    - Keep removing nodes as long as it doesn't hurt performance on the tuning set
  - Use the *Minimum Description Length* principle

Avoiding overfitting

- Some possibilities
  - Use a tuning set to decide when to stop growing the tree
  - Use a tuning set to prune the tree after it's completely grown
    - Keep removing nodes as long as it doesn't hurt performance on the tuning set
  - Use the *Minimum Description Length* principle
    - Track the number of bits used to describe the tree itself when calculating information gain

Avoiding overfitting

- Some possibilities
  - Use a tuning set to decide when to stop growing the tree
  - Use a tuning set to prune the tree after it's completely grown
    - Keep removing nodes as long as it doesn't hurt performance on the tuning set
  - Use the *Minimum Description Length* principle
    - Track the number of bits used to describe the tree itself when calculating information gain