University of Wisconsin - Madison CS 540 Lecture Notes C. R. Dyer

Machine Learning (Chapter 18.1 - 18.3)

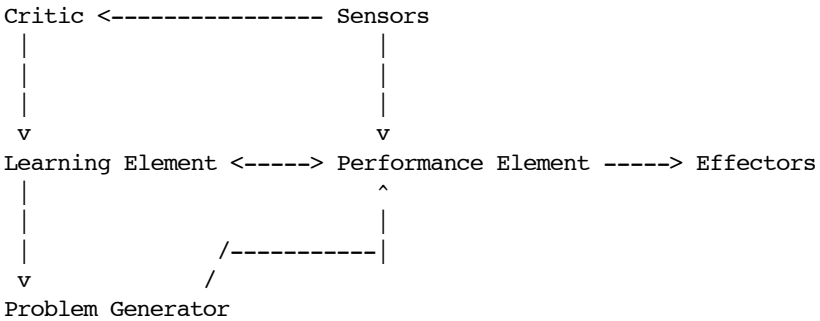
What is Learning?

- "Learning denotes changes in a system that ... enable a system to do the same task more efficiently the next time." --Herbert Simon
- "Learning is constructing or modifying representations of what is being experienced." --Ryszard Michalski
- "Learning is making useful changes in our minds." --Marvin Minsky

Why do Machine Learning?

- Understand and improve efficiency of human learning
  For example, use to improve methods for teaching and tutoring people, as done in CAI -- Computer-aided instruction
- Discover new things or structure that is unknown to humans
  Example: Data mining
- Fill in skeletal or incomplete specifications about a domain
  Large, complex AI systems cannot be completely derived by hand and require dynamic updating to incorporate new information. Learning new characteristics expands the domain or expertise and lessens the "brittleness" of the system

Components of a Learning System



- Learning Element makes changes to the system based on how it's doing
- Performance Element is the agent itself that acts in the world
- Critic tells the Learning Element how it is doing (e.g., success or failure) by comparing with a fixed standard of performance
- Problem Generator suggests "problems" or actions that will generate new examples or experiences that will aid in training the system further

We will concentrate on the Learning Element

Evaluating Performance

Several possible criteria for evaluating a learning algorithm:

- Predictive accuracy of classifier
- Speed of learner
- Speed of classifier
- Space requirements

Most common criterion is **predictive accuracy**

Major Paradigms of Machine Learning

- **Rote Learning**
  One-to-one mapping from inputs to stored representation. "Learning by memorization." Association-based storage and retrieval.
- **Induction**
  Use specific examples to reach general conclusions
- **Clustering**
- **Analogy**
  Determine correspondence between two different representations
- **Discovery**
  Unsupervised, specific goal not given
- **Genetic Algorithms**
- **Reinforcement**
  Only feedback (positive or negative reward) given at end of a sequence of steps. Requires assigning reward to steps by solving the credit assignment problem--which steps should receive credit or blame for a final result?

The Inductive Learning Problem

- Extrapolate from a given set of examples so that we can make accurate predictions about future examples.
- **Supervised versus Unsupervised learning**
  Want to learn an unknown function $f(\mathbf{x}) = \mathbf{y}$, where **x** is an input example and **y** is the desired output. Supervised learning implies we are given a set of (**x, y**) pairs by a "teacher." Unsupervised learning means we are only given the **x**s. In either case, the goal is to estimate *f*.
- **Concept learning**
  Given a set of examples of some concept/class/category, determine if a given example is an instance of the concept or not. If it is an instance, we call it a **positive example**. If it is not, it is called a **negative example**.
- **Problem: Supervised Concept Learning by Induction**
  Given a **training set** of positive and negative examples of a concept, construct a description that will accurately classify whether future examples are positive or negative. That is, learn some good estimate of function *f* given a training set {**(x1, y1), (x2, y2), ..., (xn, yn)**} where each **yi** is either + (positive) or - (negative).

Inductive Bias

- Inductive learning is an inherently conjectural process because any knowledge created by generalization from specific facts cannot be proven true; it can only be proven false. Hence, inductive inference is **falsity preserving**, not truth preserving.
- To generalize beyond the specific training examples, we need constraints or **biases** on what *f* is best. That is, learning can be viewed as searching the **Hypothesis Space** H of possible *f* functions.
- A bias allows us to choose one *f* over another one
- A completely unbiased inductive algorithm could only memorize the training examples and could

not say anything more about other unseen examples.
- Two types of biases are commonly used in machine learning:
  - **Restricted Hypothesis Space Bias**
    Allow only certain types of *f* functions, not arbitrary ones
  - **Preference Bias**
    Define a metric for comparing *f*s so as to determine whether one is better than another

Inductive Learning Framework

- Raw input data from sensors are preprocessed to obtain a **feature vector**, **x**, that adequately describes all of the relevant features for classifying examples.
- Each **x** is a list of (attribute, value) pairs. For example,

  x = `(Person = Sue, Eye-Color = Brown, Age = Young, Sex = Female)`

  The number of attributes (also called features) is fixed (positive, finite). Each attribute has a fixed, finite number of possible values.

- Each example can be interpreted as a *point* in an *n*-dimensional **feature space**, where *n* is the number of attributes.

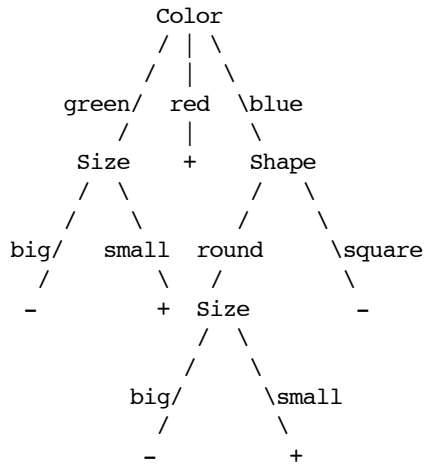
Inductive Learning by Nearest-Neighbor Classification

One simple approach to inductive learning is to save each training example as a point in Feature Space, and then classify a new example by giving it the same classification (+ or -) as its **nearest neighbor in Feature Space**.

The problem with this approach is that it doesn't necessarily generalize well if the examples are not "clustered."

Inductive Concept Learning by Learning Decision Trees

- Goal: Build a decision tree for classifying examples as positive or negative instances of a concept
- Supervised learning, batch processing of training examples, using a preference bias
- A **decision tree** is a tree in which each non-leaf node has associated with it an attribute (feature), each leaf node has associated with it a classification (+ or -), and each arc has associated with it one of the possible values of the attribute at the node where the arc is directed from. For example,



- Preference Bias: **Ockham's Razor**: The simplest explanation that is consistent with all

observations is the best. Here, that means the smallest decision tree that correctly classifies all of the training examples is best.
- Finding the provably smallest decision tree is an NP-Hard problem, so instead of constructing the absolute smallest tree that is consistent with all of the training examples, construct one that is pretty small.
- Decision Tree Construction using a Greedy Algorithm
  - Algorithm called ID3 or C5.0, originally developed by Quinlan (1987)
  - Top-down construction of the decision tree by recursively selecting the "best attribute" to use at the current node in the tree. Once the attribute is selected for the current node, generate children nodes, one for each possible value of the selected attribute. Partition the examples using the possible values of this attribute, and assign these subsets of the examples to the appropriate child node. Repeat for each child node until all examples associated with a node are either all positive or all negative.

Algorithm

```
function decision-tree-learning(examples, attributes, default)
  ;; examples is a list of training examples
  ;; attributes is a list of candidate attributes for the
  ;;    current node
  ;; default is the default value for a leaf node if there
  ;;    are no examples left
  if empty(examples) then return(default)
  if same-classification(examples) then return(class(examples))
  if empty(attributes) then return(majority-classification(examples))
  best = choose-attribute(attributes, examples)
  tree = new node with attribute best
  foreach value v of attribute best do
    v-examples = subset of examples with attribute best = v
    subtree = decision-tree-learning(v-examples, attributes - best,
                  majority-classification(examples))
    add a branch from tree to subtree with arc labeled v
  return(tree)
```

- How to Choose the Best Attribute for a Node?

Some possibilities:

- Random: Select any attribute at random
- Least-Values: Choose the attribute with the smallest number of possible values
- Most-Values: Choose the attribute with the largest number of possible values
- Max-Gain: Choose the attribute that has the largest expected information gain. In other words, try to select the attribute that will result in the smallest expected size of the subtrees rooted at its children.

The C5.0 algorithm uses the Max-Gain method of selecting the best attribute.

Information Gain Method for Selecting the Best Attribute

Use **information theory** to estimate the size of the subtrees rooted at each child, for each possible attribute. That is, try each attribute, evaluate and pick the best one.

- How much (expected) work is required to guess which element I am thinking of in a set S of size |S|?

$\log_2 |S|$

That is, at each step we can ask a yes/no question that eliminates at most 1/2 of the elements remaining. Call this value the *information value* of being told which element it is without having to guess it.

- Given S = P union N, where P and N are two disjoint sets, how hard is it to guess which element I am thinking of in S?

  if x in P, then $\log_2|P| = \log_2 p$ questions needed, where $p = |P|$
  if x in N, then $\log_2|N| = \log_2 n$ questions needed, where $n = |N|$

  So, the expected number of questions that have to be asked is:

  $$(Pr(x \text{ in } P) * \log_2 p) + (Pr(x \text{ in } N) * \log_2 n)$$

  or, equivalently,

  $$(p/(p+n))\ \log_2 p + (n/(p+n))\ \log_2 n$$

- So, how much expected work is required to guess which element I am thinking of in a set S after I am told whether the element is in P or N?

  $$I(P,N) = \log_2|S| - (|P|/|S|\ \log_2|P|) - (|N|/|S|\ \log_2|N|)$$

  or, equivalently,

  $$I(\%P, \%N) = -(\%P\ \log_2 \%P) - (\%N\ \log_2 \%N)$$

  where %P = |P|/|S| = p/(p+n) (fraction of examples in S are positive), and %N = |N|/|S| = n/(p+n) (fraction of examples in S are negative).

  I measures the **information content** or **entropy** in bits (i.e., number of yes/no questions that must be asked) associated with a set S of examples, which consists of the subset P of positive examples and subset N of negative examples.

  Note: 0 <= I(P,N) <= 1, where 0 => no information, and 1 => maximum information.

- Example: Perfect Balance (Maximum Disorder) in S:

  Half the examples in S are positive and half are negative. Hence, %P = %N = 1/2. So,

  $$\begin{aligned} I(1/2, 1/2) &= -1/2\ \log_2 1/2 - 1/2\ \log_2 1/2 \\ &= -1/2\ (\log_2 1 - \log_2 2) - 1/2\ (\log_2 1 - \log_2 2) \\ &= -1/2\ (0 - 1) - 1/2\ (0 - 1) \\ &= 1/2 + 1/2 \\ &= 1 \Rightarrow \text{information content is large} \end{aligned}$$

- Example: Perfect Homogeneity in S:

  Say all of the examples in S are positive and none are negative. Then, %P = 1, and %N = 0. So,

  $$\begin{aligned} I(1,0) &= -1\ \log_2 1 - 0\ \log_2 0 \\ &= -0 - 0 \end{aligned}$$

```
        =  0  => information content is low
```

Low information content is desirable in order to make the smallest tree because low information content means that most of examples are classified the SAME, and therefore we would expect that the rest of the tree rooted at this node will be quite small to differentiate between the two classifications.

Now, measure the **information gained** by using a given attribute. That is, measure the difference in the information content of a node and the information content after a node splits up the examples based on a selected attribute's possible values. To do this, we need a measure of the information content after "splitting" a node's examples into its children based on a hypothesized attribute.

Given a node with a set of examples S = P union N, and an hypothesized attribute A that has m possible values, define **Remainder(A)** as the weighted sum of the information content of each subset of the examples that are associated with each child node as imposed by possible values of the attribute. More specifically, let

S_i = subset of S with value i, i=1,...,m
P_i = subset of S_i that are positive examples
N_i = subset of S_i that are negative examples
q_i = $|S_i|/|S|$ = % of examples on branch i
$\%P_i$ = $|P_i|/|S_i|$ = fraction of positive examples on branch i
$\%N_i$ = $|N_i|/|S_i|$ = fraction of negative examples on branch i

$$\text{Remainder}(A) = \sum_{i=1}^{m} q_i \, I(\%P_i, \%N_i)$$

So, Remainder(A) is a weighted sum of the information content (aka entropy) at each child node generated by that attribute. It measures the total "disorder" or "inhomogeneity" of the children nodes.
0.0 <= Remainder(A) <= 1.0

Now, measure the **gain** from using the attribute test at the current node, defined by:

$$\text{Gain}(A) = I(\%P, \%N) - \text{Remainder}(A)$$

The best attribute at a node is now defined as the attribute A with maximum Gain(A) of all the possible attributes that can be used at the node. Since at a given node I(%P, %N) is constant, this is equivalent to selecting the attribute A with minimum Remainder(A).

Example

Consider the following six training examples, where each example has three attributes: color, shape and size. Color has three possible values: red, green and blue. Shape has two possible values: square and round. Size has two possible values: big and small.

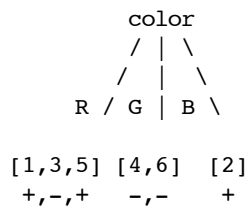
Example	Color	Shape	Size	Class
1	red	square	big	+
2	blue	square	big	+
3	red	round	small	-
4	green	square	small	-
5	red	round	big	+
6	green	square	big	-

- **Which is best attribute for the root node of decision tree?**

```
Remainder(color) = 3/6 I(2/3,1/3) + 1/6 I(1/1,0/1) + 2/6 I(0/2,2/2)
                    |     |   |      |                |
                    |     |   |      1 of 6 is blue   2 of 6 are green
                    |     |   |
                    |     |   1 of the red is negative
                    |     |
                    |     2 of the red are positive
                    |
                    |
                    3 out of 6 are red
                  = 1/2 * (-2/3 log2 2/3  - 1/3 log2 1/3)
                      + 1/6 * (-1 log21 - 0 log20)
                      + 2/6 * (-0 log20 - 1 log21)
                  = 1/2 * (-2/3(log22 - log23) - 1/3(log21 - log23))
                      + 1/6 * 0
                      + 2/6 * 0
                  = 1/2 * (-2/3(1 - 1.58) - 1/3(0 - 1.58))
                  = 1/2 * 0.914
                  = 0.457

Gain(color) = I(3/6, 3/6) - Remainder(color)
            = 1.0 - 0.457
            = 0.543

Remainder(shape) = 4/6 I(2/4, 2/4) + 2/6 I(1/2, 1/2)
                 = 4/6 * 1.0 + 2/6 * 1.0
                 = 1.0

Gain(shape) = I(3/6, 3/6) - Remainder(shape)
            = 1.0 - 1.0
            = 0.0

Remainder(size)  = 4/6 I(3/4, 1/4) + 2/6 I(0/2, 2/2)
                 = 0.541

Gain(size)  = I(3/6, 3/6) - Remainder(size)
```

```
          = 1.0 - 0.541
          = 0.459
```

Max(0.543, 0.0, 0.459) = 0.543, so color is best. Make the root node's attribute color and partition the examples for the resulting children nodes as shown:



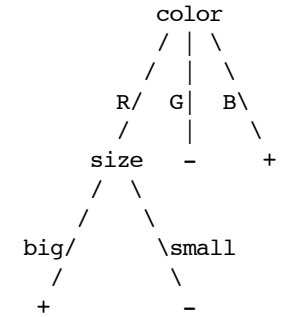
The children associated with values green and blue are uniform, containing only - and + examples, respectively. So make these children leaves with classifications - and +, respectively.

- **What is the best attribute for the red child node?**

  Now recurse on red child node, containing three examples, [1,3,5], and two remaining attributes, [shape, size].

```
Remainder(shape) = 1/3 I(1/1, 0/1) + 2/3 I(1/2, 1/2)
                 = 1/3 * 0 + 2/3 * 1
                 = 0.667

Gain(shape) = I(2/3, 1/3) - .667
            = .914 - .667
            = 0.247

Remainder(size) = 2/3 I(2/2, 0/2) + 1/3 I(0/1, 1/1)
                = 2/3 * 0 + 1/3 * 0
                = 0

Gain(size) = I(2/3, 1/3) - 0
           = 0.914
```

Max(.247, .914) = .914, so make size the attribute at this node. It's children are uniform in their classifications, so the final decision tree is:



Case Studies

Many case studies have shown that decision trees are at least as accurate as human experts. For example, one study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time, and the decision tree classified 72% correct.

British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms. Replaced a rule-based expert system.

Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example.

Extensions of the Decision Tree Learning Algorithm

- **K Class Problems, where k > 2**
  Let k be the number of classes and C1, ..., Ck are the names of the classes. Let Nm be the number of training examples at node m. Let Nm,c be the number of training examples at node m that are in class c. Define Pm,c = Nm,c / Nm which is the fraction of examples at node m that are in class c. Then, the information content (aka entropy), I, at node m is computed by

$$I(Pm,C1, \ldots, Pm,Ck) = -\sum_{c=C1, \ldots, Ck} Pm,c \log_2 Pm,c$$

  Note: When k=2, 0.0 <= I <= 1.0 but, in general, 0.0 <= I <= log2 k. The logarithm is always base 2 because entropy is a measure of the expected encoding length in bits.

  Given an attribute A, let the number of possible values of A be d, with possible values v1, ..., vd. Let Nm,v be the number of examples at node m with attribute A's value = v. Let Nm,v,c be the number of examples at node m with attribute A's value = v and class = c. Define Pm,v = Nm,v / Nm and define Pm,v,c = Nm,v,c / Nm,v Then, we can finally define

$$Remainder(A) = \sum_{v=v1, \ldots, vd} Pm,v \sum_{c=C1, \ldots, Ck} Pm,v,c \log_2 Pm,v,c$$

  Gain(A) = I(Pm,C1, ..., Pm,Ck) - Remainder(A)

- **Real-valued data**
  Select a set of thresholds defining intervals; each interval becomes a discrete value of the attribute

- **Noisy data and Overfitting**
  There are many kinds of "noise" that could occur in the examples:
  - Two examples have the same attribute, value pairs, but different classifications
  - Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  - The classification is wrong (e.g., + instead of -) because of some error
  - Some attributes are irrelevant to the decision-making process. For example, the color of a die is irrelevant to its outcome.

  The last problem, irrelevant attributes, can result in **overfitting** the training example data. For example if the hypothesis space has many dimensions because there are a large number of attributes, then we may find meaningless regularity in the data that is irrelevant to the true, important, distinguishing features. Fix by pruning lower nodes in the decision tree. For example,

if Gain of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes.

One way to address the overfitting problem in decision-tree induction is to use a tuning set in conjunction with a **pruning algorithm**. The following is a greedy algorithm for doing this.

```
Let bestTree = the tree produced by C5.0 on the TRAINING set
Let bestAccuracy = the accuracy of bestTree on the TUNING set
Let progressMade = true

while (progressMade) // Continue as long as improvement on TUNING SET
{
    Set progressMade = false
    Let currentTree = bestTree

    For each interiorNode N (including the root) in currentTree
    {   // Consider various pruned versions of the current tree
        // and see if any are better than the best tree found so far

        Let prunedTree be a copy of currentTree,
        except replace N by a leaf node
        whose label equals the majority class among TRAINING set
        examples that reached node N (break ties in favor of '-')

        Let newAccuracy = accuracy of prunedTree on the TUNING set

        // Is this pruned tree an improvement, based on the TUNE set?
        // When a tie, go with the smaller tree (Occam's Razor).
        If (newAccuracy >= bestAccuracy)
        {
          bestAccuracy = newAccuracy
          bestTree = prunedTree
          progressMade = true
        }
     }
}
return bestTree
```

- **Generation of rules**
  Each path, from the root to a leaf, corresponds to a rule where all of the decisions leading to the leaf define the antecedent to the rule, and the consequent is the classification at the leaf node. For example, from the tree above we could generate the rule:

  ```
  if color = red and size = big then +
  ```

  By constructing a rule for each path to a leaf yields an interpretation of what the tree means.

- **Setting Parameters**
  Some learning algorithms require setting learning parameters. Parameters must be set without looking at the test data! One method: Tuning Sets.

  Using Tuning Sets for Parameter Setting

  1. Partition data in Training set and Test set. Then partition Training set into Train set and Tune set.
  2. For each candidate parameter value, generate decision tree using the Train set
  3. Use Tune set to evaluate error rates and determine which parameter value is best
  4. Compute new decision tree using selected parameter values and entire Training set

- **Cross-Validation for Experimental Validation of Performance**
  1. Divide all examples into N disjoint subsets, $E = E_1, E_2, ..., E_N$
  2. For each i = 1, ..., N do
     - Test set = E_i
     - Training set = $E - E_i$
     - Compute decision tree using Training set
     - Determine performance accuracy P_i using Test set
  3. Compute N-fold cross-validation estimate of performance = $(P_1 + P_2 + ... + P_N)/N$

  One special case of interest called **Leave-1-Out** where N-fold cross-validation uses N = number of examples. Good when the number of examples available is small (less than about 100)

Summary

- One of the most widely used learning methods in practice
- Can out-perform human experts in many problems
- Strengths: Fast; simple to implement; can convert result to a set of easily interpretable rules; empirically valid in many commercial products; handles noisy data
- Weaknesses: "Univariate" splits/partitioning using only one attribute at a time so limits types of possible trees; large decision trees may be hard to understand; requires fixed-length feature vectors; non-incremental (i.e., batch method).
