;;; This is one of the example programs from the textbook:

;;;

;;; Artificial Intelligence:

;;; Structures and strategies for complex problem solving

;;;

;;; by George F. Luger and William A. Stubblefield

;;;

;;; These programs are copyrighted by Benjamin/Cummings Publishers.

;;;

;;; We offer them for use, free of charge, for educational purposes only.

;;;

;;; Disclaimer: These programs are provided with no warranty whatsoever as to

;;; their correctness, reliability, or any other property. We have written

;;; them for specific educational purposes, and have made no effort

;;; to produce commercial quality computer programs. Please do not expect

;;; more of them then we have intended.

;;;

;;; This file defines the ID3 algorithm presented in chapter 14 of the

;;; text.

;;;

;;; For a set of example data, along with instructions for its use,

;;; see the file credit.lisp

;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Data structure definitions

(defstruct property

 name ; the name of the property

 test ; an evaluable function of 1 argument,

 ; returns a property value

 values) ; a list of all possible values returned by the test

(defstruct example-frame

 instances ; A list of objects of known classification

 properties ; A list of properties of objects in the domain.

 ; These will be used to define the tree

 classifier ; A property that classifies objects in instances.

 ; The values of the classifier will be the eaves of the tree

 size ; The number of objects in instances

 information) ; The information content of instances

(defstruct partition

 test-name ; the name of the property used to partition the examples

 test ; a test function

 components ; an alist of (property-value . example-frame) pairs

 info-gain) ; information gain across all components of the partition

(defstruct decision-tree

 test-name ; the name of the property used to select a branch

 test ; an evaluable function, returns a property value used to select a branch

 branches) ; an a-list of branches, indexed by the values of test

(defstruct leaf

 value)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Functions to construct a decision tree using the ID3 algorithm

(defun build-tree (training-frame)

 (cond

 ; Case 1: Empty example set. Create leaf with no classification

 ((zerop (example-frame-size training-frame))

 (make-leaf :value "unable to classify: no examples"))

 ; Case 2: All properties used. Create leaf with all remaining classes (may be ambiguous)

 ((null (example-frame-properties training-frame))

 (make-leaf :value (list-classes training-frame)))

 ; Case 3: All instances of same class. Create a leaf

 ((zerop (example-frame-information training-frame))

 (make-leaf :value (funcall

 (property-test (example-frame-classifier training-frame))

 (car (example-frame-instances training-frame)))))

 ; Case 4: Choose test for root of tree & recursively build subtrees

 (t (let ((part (choose-partition (gen-partitions training-frame))))

 (make-decision-tree

 :test-name (partition-test-name part)

 :test (partition-test part)

 :branches (mapcar #'(lambda (x)

 (cons (car x) (build-tree (cdr x))))

 (partition-components part)))))))

; Generate all different partitions of an example frame

(defun gen-partitions (training-frame)

 (mapcar #'(lambda (x) (partition training-frame x))

 (example-frame-properties training-frame)))

; Partition takes an example frame and a property;

; It partitions the example frame on that property

; and returns an instance of a partition structure,

; where partition-components is an a-list of (property-value . example-frame) pairs

;

; It also computes the information gain and other statistics

; for each component of the partition

(defun partition (root-frame property)

 ; Initialize parts to to an a-list of empty example frames

 ; indexed by the values of property

 (let ((parts (mapcar #'(lambda (x) (cons x (make-example-frame)))

 (property-values property))))

 ; partition examples on property, placing each example in the appropriate

 ; example frame in parts

 (dolist (instance (example-frame-instances root-frame))

 (push instance (example-frame-instances

 (cdr (assoc (funcall (property-test property) instance)

 parts)))))

 ; complete information in each component of the partition

 (mapcar #'(lambda (x)

 (let ((frame (cdr x)))

 (setf (example-frame-properties frame)

 (remove property (example-frame-properties root-frame)))

 (setf (example-frame-classifier frame)

 (example-frame-classifier root-frame))

 (setf (example-frame-size frame)

 (list-length (example-frame-instances frame)))

 (setf (example-frame-information frame)

 (compute-information

 (example-frame-instances frame)

 (example-frame-classifier root-frame)))))

 parts)

 ; return an instance of a partition

 (make-partition

 :test-name (property-name property)

 :test (property-test property)

 :components parts

 :info-gain (compute-info-gain root-frame parts))))

; Choose partition takes a list of candidate partitions and chooses

; The one with the highest information gain

(defun choose-partition (candidates)

 (cond ((null candidates) nil)

 ((= (list-length candidates) 1)

 (car candidates))

 (t (let ((best (choose-partition (cdr candidates))))

 (if (> (partition-info-gain (car candidates))

 (partition-info-gain best))

 (car candidates)

 best)))))

; Lists all the classes in the instances of a training frame

(defun list-classes (training-frame)

 ; Eliminate those potential classifications not present

 ; in the instances of training frame

 (do

 ((classes (property-values (example-frame-classifier training-frame))

 (cdr classes))

 (classifier (property-test (example-frame-classifier training-frame)))

 classes-present)

 ((null classes) classes-present)

 (if (member (car classes) (example-frame-instances training-frame)

 :test #'(lambda (x y) (equal x (funcall classifier y))))

 (setf classes-present (cons (car classes) classes-present)))))

; compute the information gain of a partition

; by subtracting the weighted average of the information

; in the children from the information in

; the original set of instances.

(defun compute-info-gain (root parts)

 (- (example-frame-information root)

 (sum #'(lambda (x) (\* (example-frame-information (cdr x))

 (/ (example-frame-size (cdr x))

 (example-frame-size root))))

 parts)))

; sum takes the sum of applying f to all numbers in list-of-numbers

(defun sum (f list-of-numbers)

 (apply '+ (mapcar f list-of-numbers)))

; Computes the information content of a list of examples using a classifier.

(defun compute-information (examples classifier)

 (let ((class-count

 (mapcar #'(lambda (x) (cons x 0)) (property-values classifier)))

 (size 0))

 ; count number of instances in each class

 (dolist (instance examples)

 (incf size)

 (incf (cdr (assoc (funcall (property-test classifier) instance)

 class-count))))

 ;compute information content of examples

 (sum #'(lambda (x) (if (= (cdr x) 0) 0

 (\* -1

 (/ (cdr x) size)

 (log (/ (cdr x) size) 2))))

 class-count)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;; Classifies an instance using a decision tree

(defun classify (instance tree)

 (if (leaf-p tree)

 (leaf-value tree)

 (classify instance

 (cdr (assoc (funcall (decision-tree-test tree) instance)

 (decision-tree-branches tree))))))