Artificial Intelligence (CS 4253/6613)

Project on Supervised Learning: Decision Trees & Feedforward, Multilayer Neural Networks

In this project you will write Python code to implement and evaluate two supervised learning schemes. You will work individually and turn in commented code, runs on some test data to show your implementations work as expected, and a report with analysis of your results when you vary algorithm options or paramters.

Part A. Decision Tree Supervised Learner (40 points): Write a program that performs *supervised classification* using the algorithm presented in Figure 19.5 in the Russell & Norvig textbook (Fourth Edition). Implement two variants, one using the entropy measure (page 662) and the other using Gini index (see slide 16 of the file *SupervisedLearningDecisionTrees.pptx*), for deciding the next attribute to branch on.

Part B. Artificial Neural Network (ANN) Supervised Learner (40 points): A multilayer feedforward neural network for supervised learning, with a backpropagation learning scheme to learn appropriate weights to map inputs to target outputs (implement algorithm in Figure 1).

Evaluation & Report (mandatory, 20 points): Evaluate the decision tree and ANN learners on several data sets (use the data set provided for the kNN project and at least two other data sets from the UC Irvine ML repository No grade will be assigned without the Evaluation and Report.

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example (**x**, **y**) **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j} a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

Figure 1: Backpropagation algorithm for learning weights in layered, feedforward ANNs.