DS 2500: Intermediate Programming with Data
Prof. Rachlin
Northeastern University

k-Nearest Neighbor classification via Evolutionary Computing

Problem Description:

In the last lab you implemented a single-nearest neighbor algorithm to classify each iris plant in the dataset. Using n-fold cross validation (leave-one-out), you tested each iris record in turn by finding its nearest neighbor, 'predicting' that the species matches that of the nearest other flower in our 4-dimensional attribute space. The result was an overall accuracy of 96% (144 out of 150 irises correctly classified). Can we do better?

In the iris data set, when we wanted to find the nearest neighbor, we used a Euclidean distance measure:

$$\text{diff} = \sqrt{(\Delta Slen)^2 + (\Delta Swid)^2 + (\Delta Plen)^2 + (\Delta Pwid)^2}$$

Note that taking the square root really had no impact. A more general form for our distance measure is:

$$\text{diff} = w_1|\Delta A_1|^r + w_2|\Delta A_2|^r + \ldots + w_n|\Delta A_n|^r$$

where

ΔA_i is the difference with respect to feature i,

w_k is a feature weighting factor (0.0 to 1.0), and

r is an exponent (> 0.0)

We have dropped the square root because it doesn't impact our results.

Part A: k-nearest neighbors

Modify your lab code so that it supports finding k > 1 nearest neighbors instead of just a single nearest neighbor. If there is a discrepancy among the predictions for the k nearest neighbors, we'll assign a prediction based on the class of the *majority* of the neighbors. You may assume that k is odd so that you don't need to deal with ties.

Part B: Generalized distance measure

Instead of using a standard Euclidean distance measure (all weights = 1.0 and r = 2), support the more generalized distance function given above.

Part C: Evolve an optimal nearest-neighbor classifier

Use the evolutionary computing framework given in class (and posted on Canvas) to evolve different values for the number of neighbors (k) the feature weights (w_i values), and the exponent (r). A solution is just a collection of these parameters. Each solution is evaluated by testing it against the iris data set to determine how well the resulting classifier performs. Evaluate your performance according to three objective criteria:

a) Minimize overall error (incorrect classifications / total classifications), and
b) *Minimize* (*1 - F_{score}*) (https://deepai.org/machine-learning-glossary-and-terms/f-score)
c) *Minimize # of weights > 0.0* (We will favor models that eliminate unnecessary features.)

Your agent will tweak the parameters in some way. For example, it might increase or decrease the k value by 2, while altering the r value and one or more weights by some incremental value. Can you discover a better approach to nearest-neighbor classification of the iris dataset?

What to submit:

Submit your code and a short ½-page report describing your optimal classifier(s).

IMPORTANT: Write your code so that it can handle *any* tabular dataset consisting of multiple numeric columns and one non-numeric *class* column. We may want to explore these techniques as a class on datasets other than the relatively easy iris dataset.