CSE/ECE 848 Introduction to Evolutionary Computation

Module 2, Lecture 7, Part 1
An Example and Typical Behavior of EAs

Erik D. Goodman, Executive Director
BEACON Center for the Study of Evolution in Action
Professor, ECE, ME, and CSE

A Real-World Example of an EA in Action: GA for Feature Selection and Extraction in Pattern Classification

This example is from:

Dimensionality Reduction Using Genetic Algorithms
Michael L. Raymer, William F. Punch, Erik D. Goodman, Leslie A. Kuhn, and Anil K. Jain
IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 4, NO. 2, JULY 2000

and has been cited over 750 times.

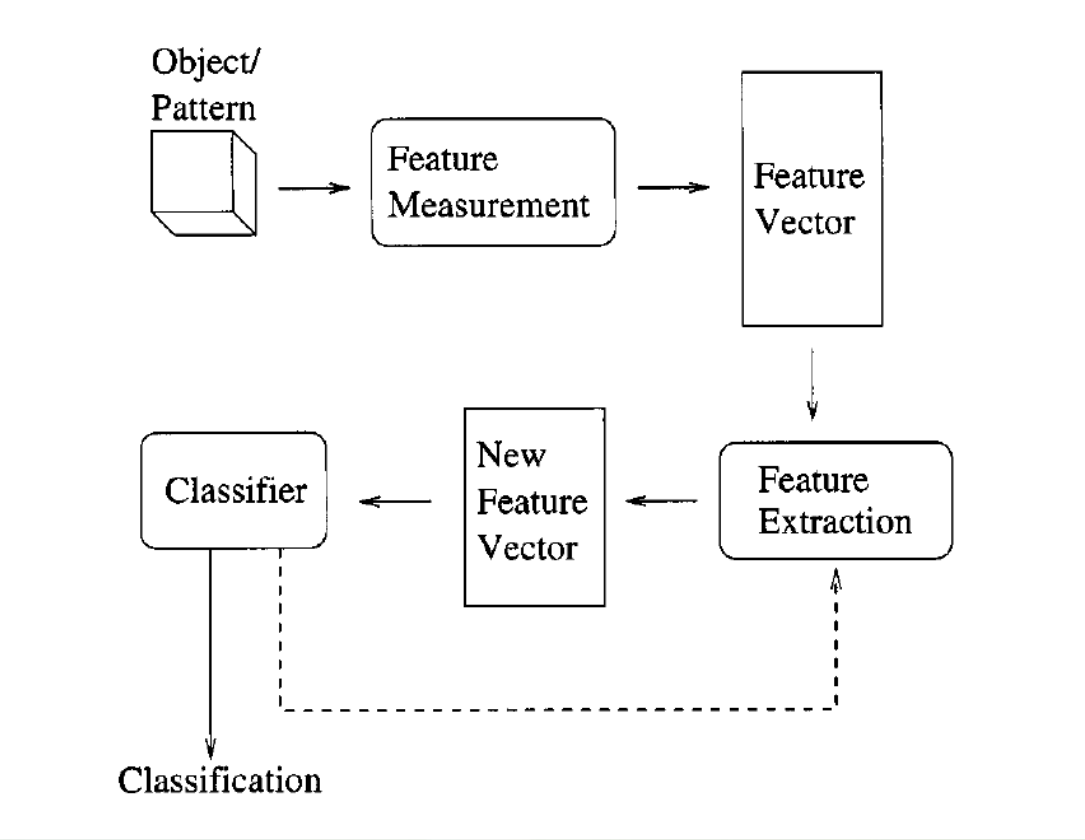
It will illustrate how GAs can be used with ML tools to tackle difficult problems

A Real-World Example of an EA in Action: GA for Feature Selection and Extraction in Pattern Classification

- Pattern classification is a common problem in computer/data science/ML/AI
- Typical problem: Given results of *m* tests (i.e., *m features*) for each of *N* patients, maximize the classification accuracy as to which patients have/do not have a given condition
- Useful both for diagnosing future patients and for deciding which tests are most important to run!
- If we can get high accuracy without running some of the tests, they can be eliminated, and this is OFTEN the case
- Beware: in real-world problems, more time is often spent on "cleaning" the data than on any other task!

Classification Approach with Feature Extraction

- Divide dataset (feature measurements) into three disjoint subsets:
  - Training
  - Tuning
  - Testing
- Select and extract a subset of the features (i.e., *weight* a subset)
- Embed a k-nearest neighbor (knn) classifier



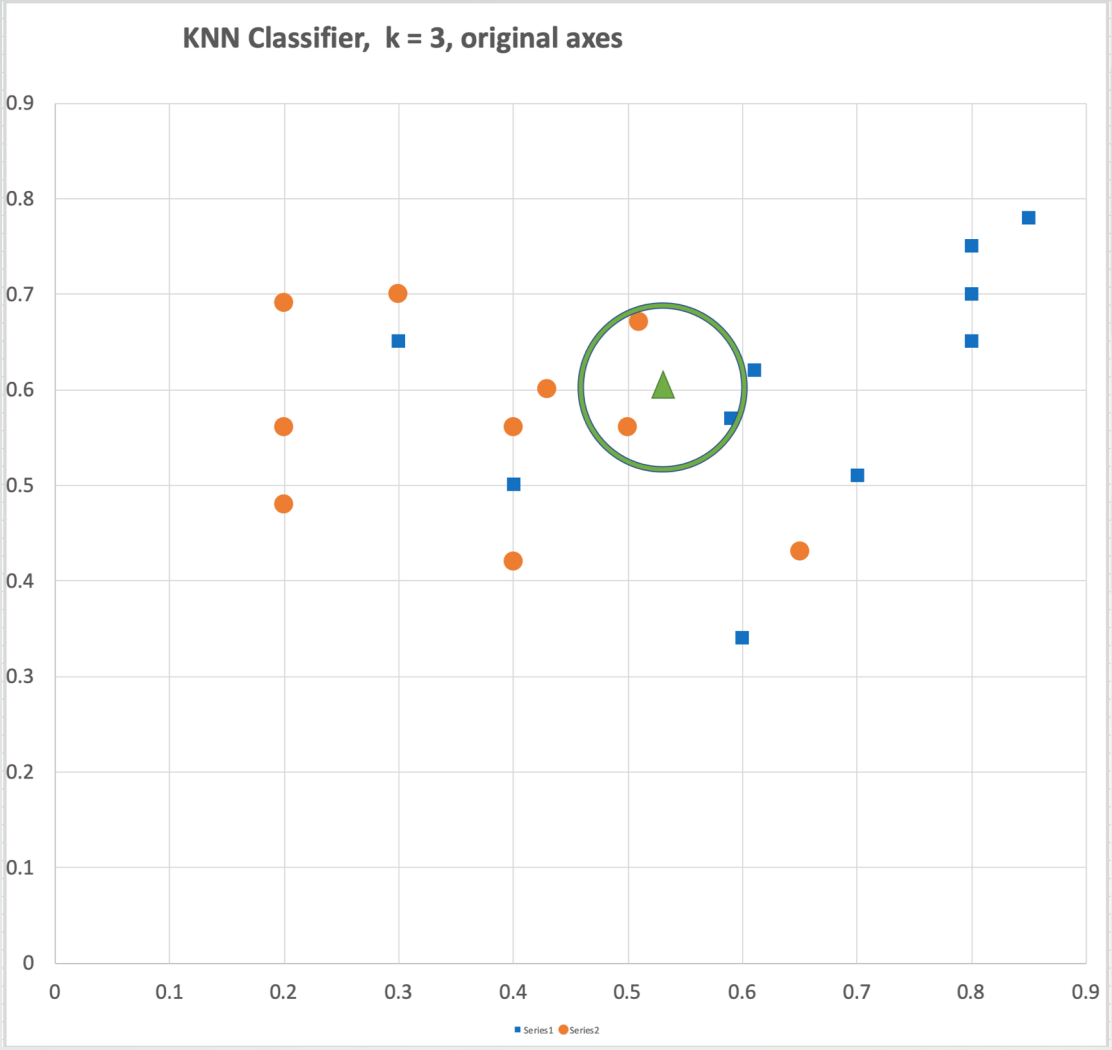
We used a GA to do the feature extraction!

Normal knn Classifier— A "Workhorse" of Classification!

- Fix k as some odd number
- To classify any instance:
  - Determine the k nearest instances in the dataset, using a Euclidean distance metric (usually, but could be another)
  - Classify the instance as whatever is the majority class among those neighbors (won't be ties as long as k odd and we are doing only binary classification: ***is*** or ***isn't***)

Let's first look at an example, then see how we can improve on this using evolutionary computation

Illustration of knn for classifying a new triangle (in unknown class), as red or blue

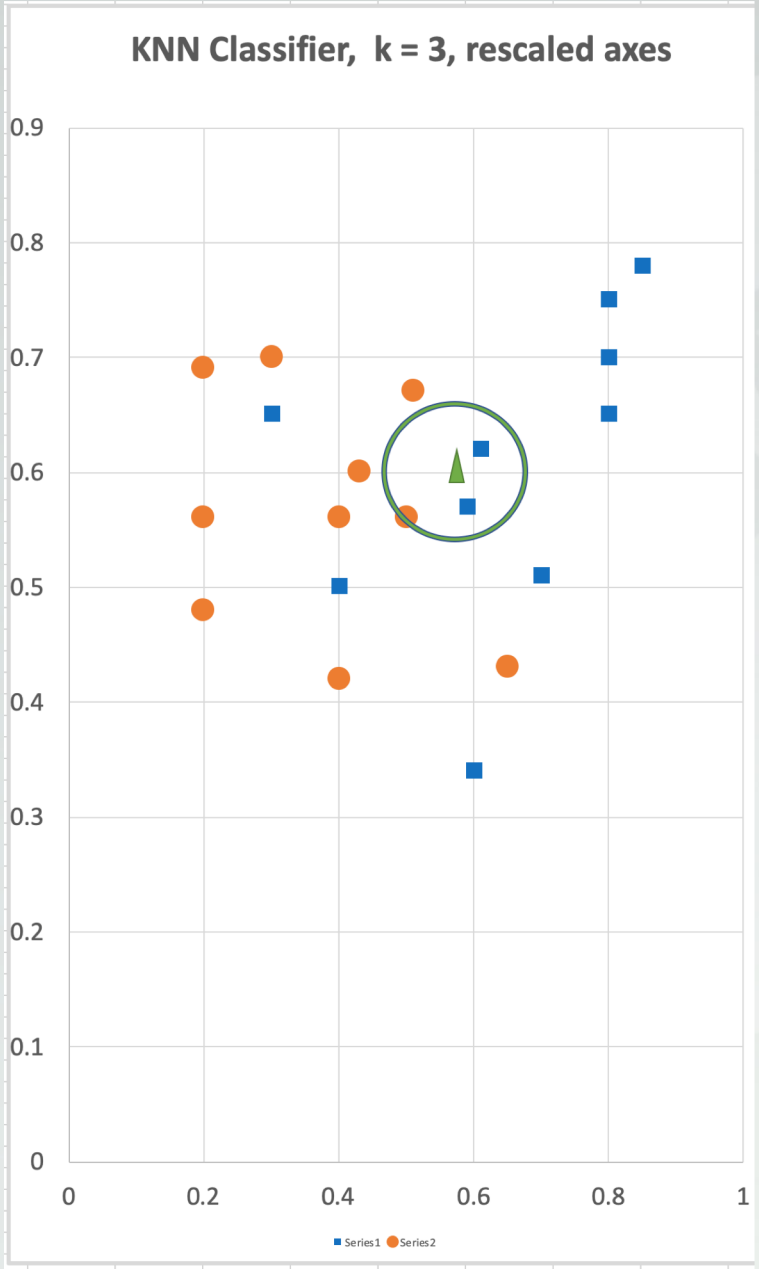


If only 2 features, and if k = 3, we find the tightest circle around the triangle (point to be classified) that contains 3 points, and label the unknown as the majority color of those 3 points. So the new point is labeled red, in this case.

Weighting the tests (rescaling the axes)

The knn *might* achieve higher accuracy on our training set (and later on other data) if we assign different weights to the m test measures.

- First we normalize each measure to [1., 10.]
- We will use a GA to weight them [0., 100.], assigning a real "weight" (or scaling factor) to each axis (test result)
- What does that do (example)?



Example knn after Re-weighting:

We have now "squished" the x-axis (first test result), making it more important

- Notice that the new (unknown) point originally labeled as red would now be labeled as blue (2 of the 3 now closest points are blue)
- We can do this to a different extent on each axis, "tuning" the knn to work better for THIS problem
- How? Let the GA do it!

Improving KNN Classification with GA

- Would be good if we could use fewer than all m tests to make the decision, but without losing (much) accuracy
- In basic knn, distances along different test axes may be misleading—we can use a GA to scale the axes to improve knn results—need to evolve real numbers as weighting (scaling) factors

We will use a GA to decide THREE things:

1) What should k be?
2) Which subset of the m tests should we use?
3) How should we weight the tests (i.e., rescale the axes)

How should we REPRESENT the problem?

- Let's make the first locus be an int, representing k, in range [0,100]
- Let's let the next m (# tests or features) loci be reals, in range [1.0, 10.0]
- Let's let the next m loci be *masks*, saying whether feature i is IN or OUT

This isn't the simple GA, is it…

Fields are different types, different lengths, so the fitness function and operators must take those differences into account. So somebody will be writing some code, somewhere…

But it's relatively easy… You can write it yourself, or **many** frameworks let you specify the range and type of each locus, and how to mutate them, and where you can cross them over.

To consider:

- We could have left off the binary fields—zeroing a weight has the same effect. But the evolution proceeds much more rapidly leaving weights at evolved good values and just switching them on and off, rather than leaving it to them to evolve down to 0 and then back.
- When you change k, many classifications will change, so you might want to change k only infrequently, to let evolution ***explore*** any given k (or set of a few) relatively well before moving on…
- We eventually made *multi-bit* masks for each feature… a feature is ON only if the majority of its mask bits are 1's. This makes mutations less disruptive to the classifier, allowing for more gradual change (more neutral fitness landscape)

How did we evaluate fitness of any classifier?

- Used a weighted sum of four objectives, minimizing:
  - C_{pred} = # incorrect predictions
  - C_{mask} = # unmasked features
  - C_{vote} = # incorrect votes
  - C_{bal} = diff. in accuracy between classes

$$\text{Fitness} = 20 * C_{pred} + 1 * C_{mask} + 2 * C_{vote} + 5 * C_{bal}$$

(Today, we would have used *evolutionary multi-objective optimization*, but we didn't back then… we just forced the tradeoff among objectives)

Other GA parameters

- Crossover rate 0.8
- Mutation rate 0.001
- Population size 200
- Generations 200
- Gaussian mutation on reals, binary mutation on bits, discretized Gaussian mutation on k
- Two-point crossover at field boundaries

We tested our GA/knn classifier on the "standard" Appendicitis Dataset

TABLE V
RESULTS REPORTED BY WEISS, AND THOSE OF THE GA FEATURE EXTRACTOR, ON APPENDICITIS DATA

Method	Accuracy (training)	Accuracy (testing)
GA Feature Extractor	90.4%	90.6%
Linear Discriminant	88.7%	86.8%
Quadratic Discriminant	79.3%	73.6%
Nearest Neighbor	100%	82.1%
Bayes (independent)	88.7%	83.0%
Bayes (2nd order)	95.3%	81.1%
Neural Net (Back prop)	90.0%	85.8%
Predictive Value Max.	91.5%	89.6%
CART Tree	90.0%	84.9%

Results on our MSU dataset

- Our colleague, Dr. Leslie Kuhn, provided data on water binding sites on proteins, and eight features associated with those sites
- Need a binary classification of each site: when the protein docks, does the water molecule ***stay*** or ***go***… that is important in evaluating potential drug candidates and related protein chemistry
- This is a very hard classification task, as the data overlap to a high degree in the 8-dimensional space, and we KNEW the feature set was inadequate to capture the ideal classification. It is why we undertook this research!

Results on Protein Docking Site Water Retention

FEATURE WEIGHTS, k VALUE, AND MEAN BOOTSTRAP ACCURACY (ACC) FOR THE BEST TWO WEIGHT SETS AND FEATURE SUBSETS FOUND USING THE GA FEATURE EXTRACTOR AND SFFS FEATURE SELECTION IN COMBINATION WITH A k NEAREST NEIGHBOR CLASSIFIER

Method	k	Acc	ADN	AHP	BVAL	HBDP
GA/knn	65	64.20	0.00	0.00	0.413	0.135
GA/knn	29	63.62	0.00	0.00	0.667	0.00
SFFS/knn	65	63.21	0.00	1.00	1.00	0.00
Method	k		HBDW	MOB	ABVAL	NBVAL
GA/knn	65		0.137	0.315	0.00	0.00
GA/knn	29		0.00	0.333	0.00	0.00
SFFS/knn	65		0.00	1.00	0.00	0.00

Conclusions about this example

- This example is NOT atypical of the kinds of modifications needed to a “simple GA” in order to address real-world problems
- Hard problems are just hard. Inadequate information can make accurate classification difficult, even with the best tools! As the understanding of protein chemistry/physics deepens, so will the ability to make better classification decisions. We are called on to wring all we can from the *available* data.