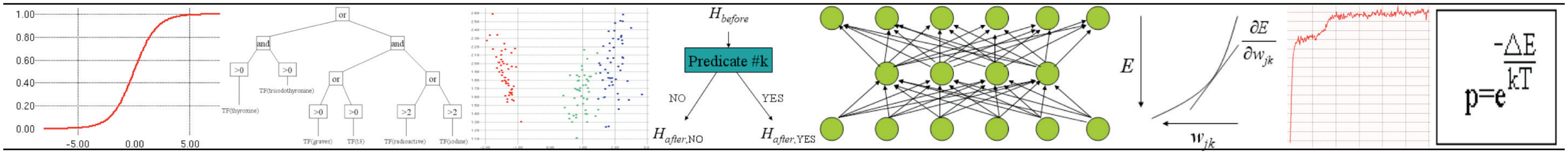


BioMaLL

Bioinformatics Machine Learning Library

B. Majoros

Part I

Overview

Current Contents

Classification methods

K-Nearest Neighbors
w/Mahalanobis Distance
Naive Bayes
Linear Discriminant Analysis
Entropy-based Decision Trees
Feedforward Neural Networks
Multivariate Regression
Genetic Programming
Bayesian Networks
Logistic Regression
Simulated Annealing

Feature selection methods

F-ratio
PCA
LDA

Sequence parsing methods

Hidden Markov Models

Phylogenetic Inference

UPGMA
Neighbor-Joining
Maximum Parsimony
Felsenstein's Algorithm

(grey = coming soon)

Compiling and Installing BioMaLL

BioMaLL can be downloaded on the internet at:

http://www.geneprediction.org/biomall/index.html

Unpack the “tarball” via the commands:

```
gunzip biomall.tar.gz

tar xvf biomall.tar
```

In the BioMaLL directory, enter the command

```
make biomall
```

to compile the library.

Running BioMaLL

All BioMaLL programs are executed via the UNIX command-line.

The correct usage of each program can be determined by running the program with no parameters. The program will print out a *usage statement*:

```
[bmajoros $] apply-bayes-net
apply-bayes-net <*.model> <*.names> <*.data> <outfile>
```

i.e., this program requires four parameters: a model file, a names file, a data file, and the name of a file where the output should be stored.

Directory Structure

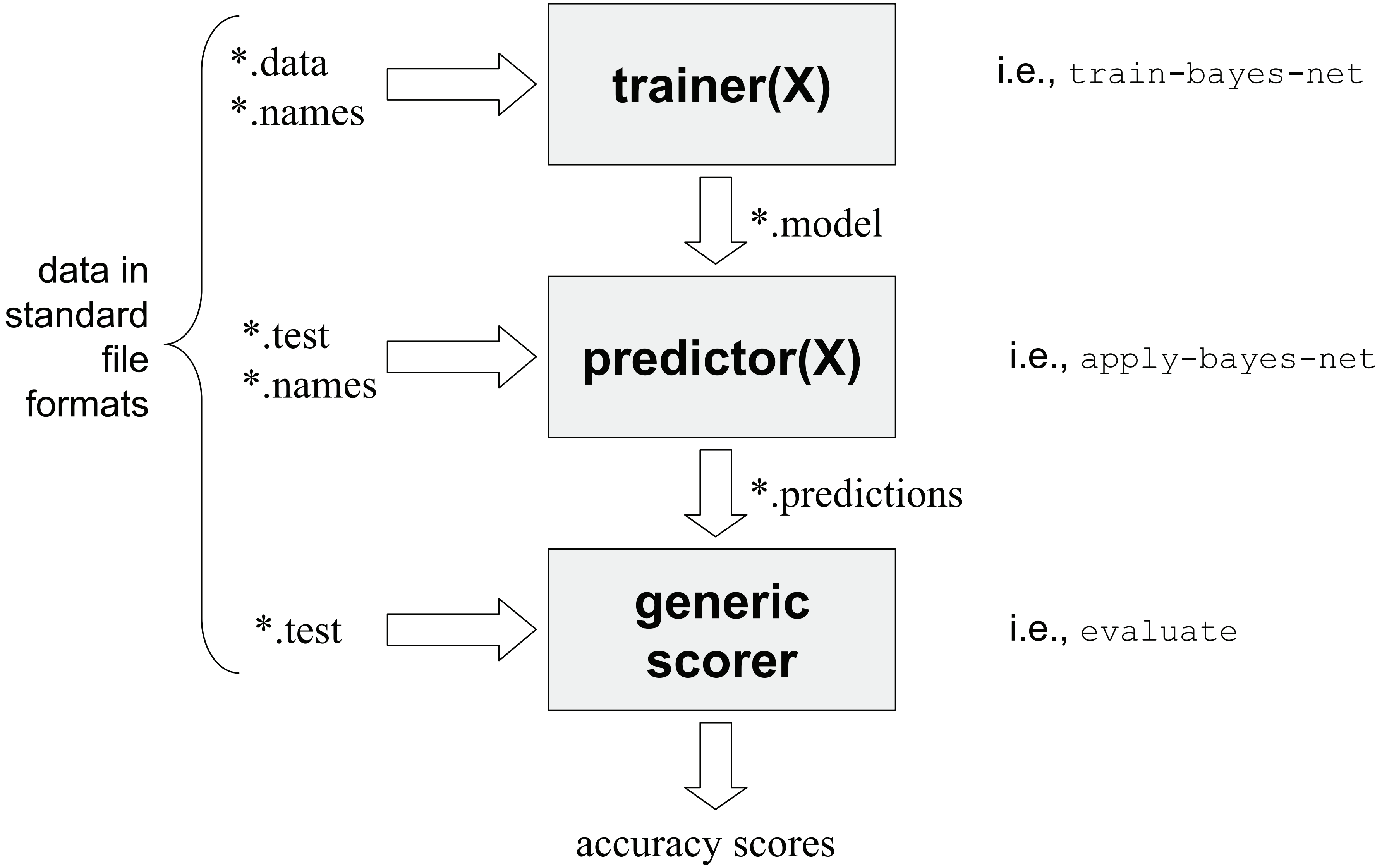
BioMaLL

- common = source code common to all classifiers
- BOOM* = container class library (Bioinformatics Object-Oriented Modules)
- annealing = simulated annealing
- bayes = naive Bayes classifier
- bayes-net = Bayesian networks
- ET = entropy-based decision trees
- f-ratio = feature selection via F-ratio
- GP = genetic programming
- knn = K-nearest neighbors classifier
- LDA = Fisher’s linear discriminant analysis
- logistic = logistic regression
- neural = feedforward neural network classifier
- PCA = principal components analysis
- progen = synthetic problem generator
- regress = multivariate linear regression classifier

each type of classifier is in a separate subdirectory

***BOOM is built on the standard template library (STL), the gnu scientific library (GSL), and the template numerical toolkit (TNT)**

Applying Algorithm X



data in standard file formats

*.data
*.names

trainer(X)

i.e., `train-bayes-net`

*.model

*.test
*.names

predictor(X)

i.e., `apply-bayes-net`

*.predictions

*.test

generic scorer

i.e., `evaluate`

accuracy scores

File Formats

The *.names file specifies the attributes (and their data types) of the objects to be classified, and the number of categories into which they can be classified:

```
2 categories
orf_length:        continuous
signal1_score:     continuous
signal2_score:     continuous
hexamer_score:     continuous
```

The possible data types are `continuous` (meaning numerical) and `discrete` (meaning categorical). Categorical attributes such as color must be encoded into integer values (i.e., representing red white and blue as 1 2 and 3).

The *.data (for training) and *.test (for accuracy evaluation) files contain one line per object to be classified, with attribute values separate by whitespace; attributes must be in the same order as given in the *.names file:

```
-7.2200        -46.4053        -81.4875        15.5713        1
-7.0832        -56.6218        -65.6119        -15.9614       0
-7.1820        -56.4384        -65.6939        -5.89178       0
  ...            ...              ...             ...          ...
```

The last column indicates the correct category of the object. Categories must be numbered starting at zero.

Accuracy Evaluation

The `evaluate` program in the root BioMaLL directory compares a set of predictions to a *.test file and reports the accuracy:

```
[bmajoros $] apply-bayes 1.model 1.names 1.test 1.out
[bmajoros $] ../evaluate
evaluate <predictions> <test-cases>
[bmajoros $] ../evaluate 1.out 1.test
84% accuracy
```

A *baseline accuracy* can be assessed using the `baseline` program from the root BioMaLL directory:

```
[bmajoros $] ../baseline 1.data 1.test
50 %     [UNIFORM RANDOM GUESSING]
52.14 %  [ALWAYS PREDICT CLASS=0]
50.09 %  [RANDOM GUESSING BY TRAINING DISTRIBUTION]
```

Part II

Algorithm Descriptions and Examples

Naïve Bayes Classification

Classify an object (=feature vector) **X** into the most probable category Y_i according to $P(Y_i|\mathbf{X})$.

Use Bayes' Theorem to invert $P(Y_i|\mathbf{X})$:

$$P(Y_i \mid X) = \frac{P(X \mid Y_i)P(Y_i)}{\sum_j P(X \mid Y_j)P(Y_j)}$$

Since the denominator is invariant w.r.t. Y_i, it suffices to compute:

$$Y^* = \underset{Y_i}{\arg\max}\, P(X \mid Y_i)P(Y_i)$$

$P(Y)$ is trivial (just count training cases), so we are left with:

$$P(X|Y_i) \approx P(X_1{=}x_1|Y_i) \cdot P(X_2{=}x_2|Y_i) \cdot \ldots \cdot P(X_n{=}x_n|Y_i),$$

assuming conditional independence (the "naive Bayes" assumption).

Example: Training and Applying a Naive Bayes Classifier

```
[eaglet] BioMaLL/bayes> cat arab1.names
2 categories
length_prob:    continuous
signal1_score:  continuous
signal2_score:  continuous
hexamer_score:  continuous
[eaglet] BioMaLL/bayes> less arab1.data
-7.22008    -46.4053    -81.4875    15.5713         1
-7.08321    -56.6218    -65.6119    -15.9614        0
-6.1875     -40.117     -80.3785    -13.286         0
-7.18202    -56.4384    -65.6939    -5.89178        0
...etc...
[eaglet] BioMaLL/bayes> less arab1.test
-4.9694     -79.1143    -52.7902    -9.49414    1
-5.21918    -79.577     -55.1701    4.30175     1
-6.1543     -50.455     -62.5431    -80.2211    0
-6.25661    -56.3978    -72.3367    12.7841     0
...etc...
[eaglet] BioMaLL/bayes> train-bayes arab1.data arab1.names arab1.bayes 10
[eaglet] BioMaLL/bayes> apply-bayes arab1.bayes arab1.names arab1.test arab1.predictions
[eaglet] BioMaLL/bayes> ../evaluate arab1.predictions arab1.test
85.71 %
[eaglet] BioMaLL/bayes> ../baseline arab1.data arab1.test
50 %     [UNIFORM RANDOM GUESSING]
47.85 %  [ALWAYS PREDICT CLASS=1]
49.98 %  [RANDOM GUESSING BY TRAINING DISTRIBUTION]
```

Bayes Network Classification

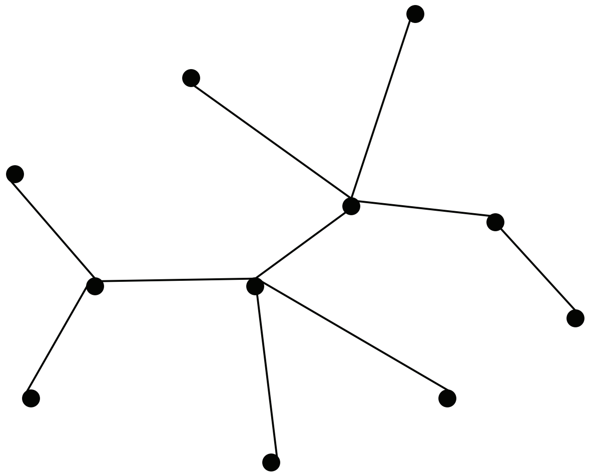
Just like Naive Bayes, except that we allow some attributes to be dependent on other attributes:

$$P(X|Y_i) \approx P(X_1{=}x_1|X_{parent(1)},Y_i) \cdot P(X_2{=}x_2|X_{parent(2)},Y_i) \cdot \ldots \cdot P(X_n{=}x_n|Y_i),$$

and assume conditional independence of all others. One option for building the dependence network is to compute all pairwise χ^2 independence statistics, and then build a *maximal spanning tree* (MST) using these χ^2 values as edge weights:

$$\chi^2 = \sum_i \sum_j \frac{(o-e)^2}{e} \qquad o_{i,j} = M_{i,j}$$

$$e_{i,j} = \frac{\left(\sum_k M_{k,j}\right)\left(\sum_k M_{i,k}\right)}{\sum_h \sum_k M_{h,k}}$$



Example: Training and Applying a Bayes Network Classifier

```
[eaglet] BioMaLL/bayes-net> cat arab1.names
2 categories

[eaglet] BioMaLL/bayes-net> ./train-bayes-net arab1.names arab1.data arab1.bn 8
Accuracy on the training set: 87%

[eaglet] BioMaLL/bayes-net> apply-bayes-net arab1.bn arab1.names arab1.test
arab1.predictions

[eaglet] BioMaLL/bayes-net> ../evaluate arab1.predictions arab1.test
88.71 %
```

K-Nearest Neighbors Classification

Given object *X*, find the *K* most similar training examples and classify *X* into the most common category *Y* among the *K* neighbors.

Compute object similarity using Euclidean distance:

$$d(X_i, X_j) = \sqrt{\sum_i (X_i - X_j)^2}$$

Or use Mahalanobis distance to control for correlations:

$$D = \sqrt{(\vec{x}_1 - \vec{x}_2)^T V^{-1} (\vec{x}_1 - \vec{x}_2)}$$

$$V^{-1} = \text{inverse of covariance matrix :}$$

$$\mathrm{V} = [\mathrm{c}_{\mathrm{jk}}]$$

$$c_{jk} = \frac{\sum_{i=1}^{n} (x_{ij} - \overline{x}_j)(x_{ik} - \overline{x}_k)}{n-1}$$

Example: Training and Applying K-Nearest-Neighbors

```
[eaglet] BioMaLL/knn> knn
knn [-wsm] <K> <names-file> <train-file> <test-file> <out-file>
   -w : weight variables by F ratio (between-group MS/within-
   group MS)
   -s : stepwise - drop variables with low F ratio
   -m : mahalanobis - account for multicollinearity

[eaglet] BioMaLL/knn> knn 3 arab1.names arab1.data arab1.test
ar a b1.predictions
5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80%
85% 90% 95% 0.475333 sec

[eaglet] BioMaLL/knn> ../evaluate arab1.predictions arab1.test
92 %

[eaglet] BioMaLL/knn> knn -m 3 arab1.names arab1.data
arab1.test ara b1.predictions
5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80%
85% 90% 95% 3.8644 sec

[eaglet] BioMaLL/knn> ../evaluate arab1.predictions arab1.test
93 %
```

Fisher's Linear Discriminant Analysis

Find linear combination(s) of variables that maximize F-ratio:

$F = MS_{between}/MS_{within}$ = largest eigenvalue of **B** (see below)

and take coefficients from the corresponding eigenvector.

B & **W** = matrices of sums of squares & cross-products (B="between groups," W="within groups")

$$\mathbf{B} = \mathbf{T} - \mathbf{W} \qquad \mathbf{T} = [t_{rc}] \qquad \mathbf{W} = [w_{rc}]$$

$$t_{rc} = \sum_{j=1}^{m} \sum_{i=1}^{n_j} (x_{ijr} - \bar{x}_r)(x_{ijc} - \bar{x}_c)$$

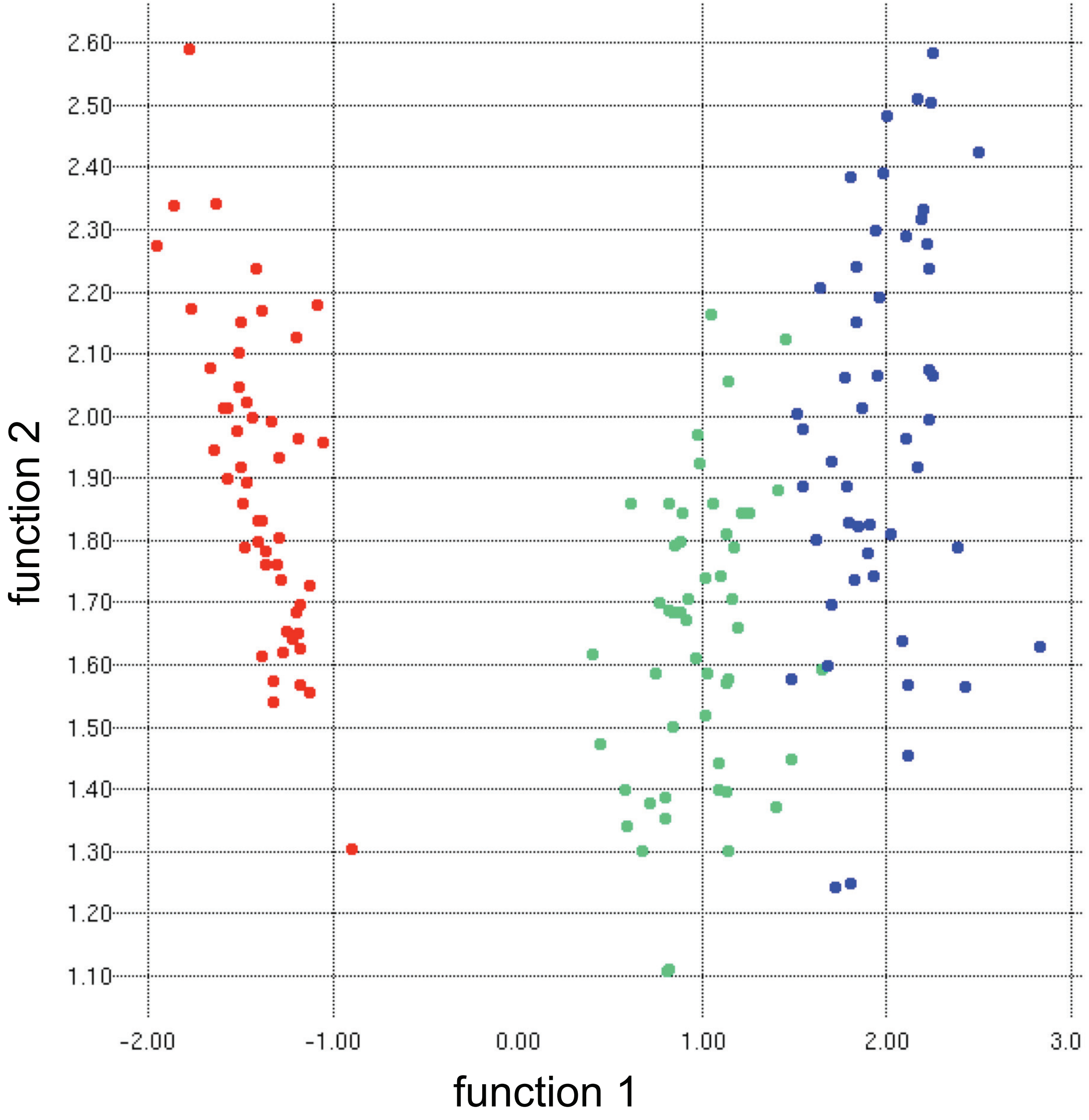
$$w_{rc} = \sum_{j=1}^{m} \sum_{i=1}^{n_j} (x_{ijr} - \bar{x}_{jr})(x_{ijc} - \bar{x}_{jc})$$

Apply significant eigenvectors as linear combinations, collect into a vector, and use nearest-centroid to classify test case.

Example: Training and Applying LDA

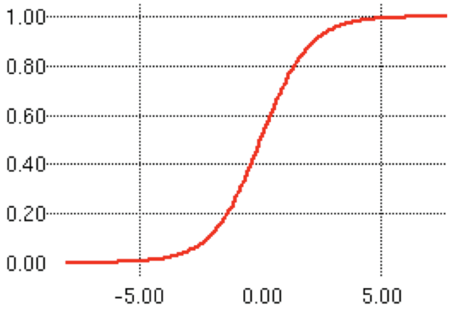
```
[eaglet] BioMaLL/LDA> train-lda -d 2 arab1.data arab1.names
arab 1.lda
rounded eigenvalues: 0.84, 0, 0, 0
using 1 discriminant function
accuracy on training set: 85

[eaglet] BioMaLL/LDA> apply-lda arab1.lda arab1.names
arab1.test arab1.predictions

[eaglet] BioMaLL/LDA> ../evaluate arab1.predictions arab1.test
84.57 %
```

Distinguishing Three Species of Iris by LDA



Logistic Regression

$$P_X = P(X \in S) = \frac{1}{1+e^{-(a+b_1x_1+b_2x_2+\ldots+b_nx_n)}}$$



Maximize log-likelihood L of training data:

$$L = \sum_{X \in S} \ln P_X + \sum_{X \notin S} \ln(1 - P_X)$$

$$\frac{\partial P}{\partial b_i} = \frac{x_i \lambda}{(1+\lambda)^2} \qquad \lambda = e^{-a-b_1x_1-\ldots-b_nx_n}$$

$$\frac{\partial L}{\partial b_i} = \sum_{X \in S} \frac{x_i \lambda}{(1+\lambda)^2 P_X} - \sum_{X \notin S} \frac{x_i \lambda}{(1+\lambda)^2 (1-P_X)}$$

*rounding errors in the computer can cause division by zero when P_X approaches 0 or 1

($\partial L/\partial a$ can be obtained by setting x_i=1)

Example: Classification using Logistic Regression

```
[eaglet] BioMaLL/logistic> train-logistic
train-logistic [options] <*.names> <*.data> <outfile>
  where:
    -i <N> : use N iterations of gradient ascent (default 50)
    -r <N> : randomly restart N-1 times and take the best (def 5)
    -t <T> : quit when error<T (threshold) (default 0.0001)
    -s <s> : use stepsize s for the gradient ascent (default 0.1)
    -a <G> : use optimization algorithm G (default BFGS)
             G can be: BFGS, STEEPEST_DESCENT, FLETCHER_REEVES,
                       POLAK_RIBIERE, SIMPLEX

[eaglet] BioMaLL/logistic> train-logistic arab1.names arab1.data
arab1.model
88% accuracy on training set

[eaglet] BioMaLL/logistic> apply-logistic arab1.names
arab1.model arab1.test arab1.predictions

[eaglet] BioMaLL/logistic> ../evaluate arab1.predictions
arab1.test
87.71 %
```

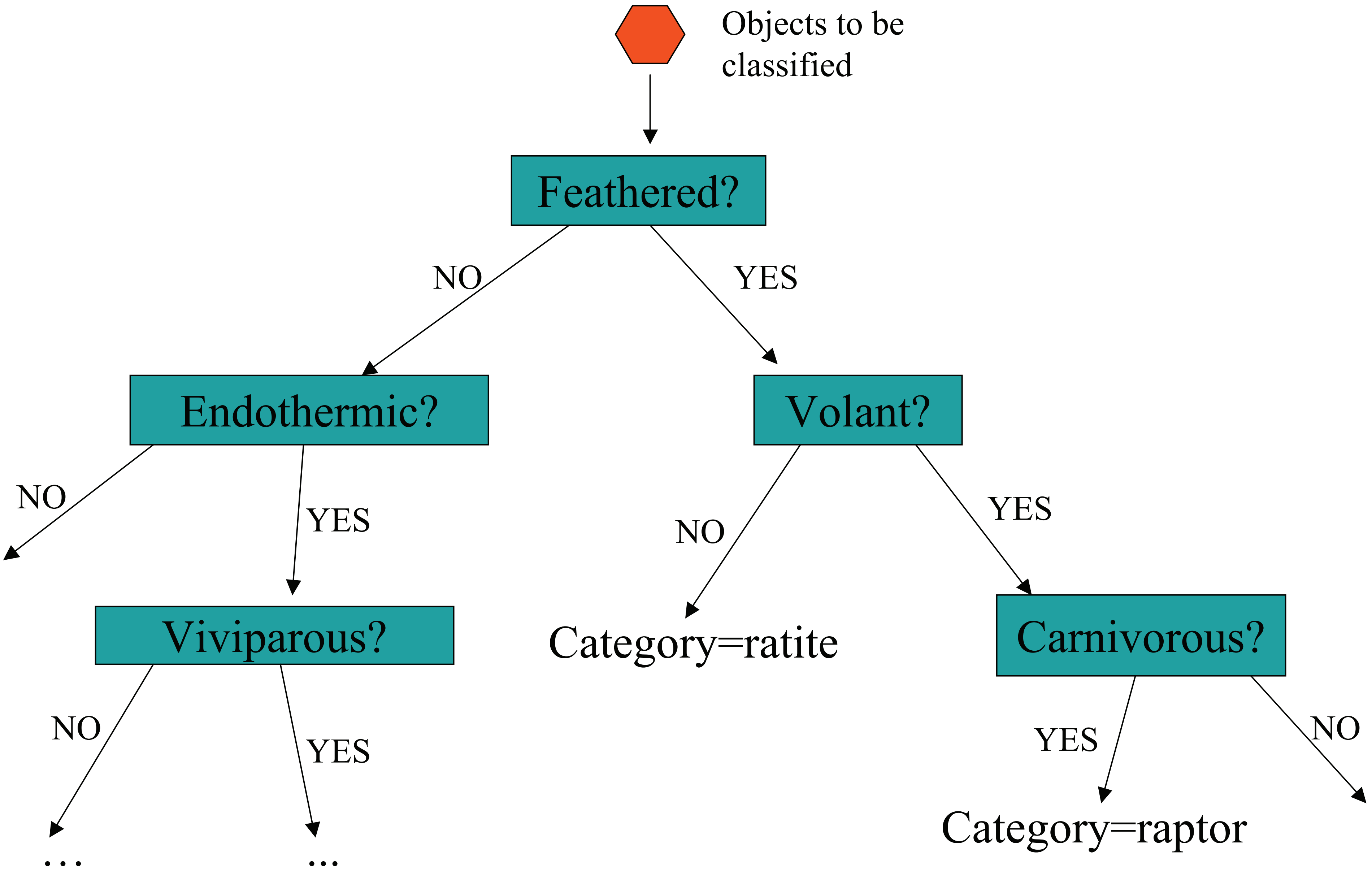
Multivariate Linear Regression Classification

$$\mathbf{A}=(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Example: Multivariate Linear Regression for Classification

```
[eaglet] BioMaLL/regress> regress arab1
discriminator: 0.118892*x0 + -0.00944905*x1 +
0.0044891*x2 + 0.00 475142*x3 + 0.986058
Accuracy on training set:  85.4%
Accuracy on test set:  84.4286%
```

Entropy-Based Decision Trees (ID3, C4.5)



Objects to be classified

Feathered?

NO

YES

Endothermic?

Volant?

NO

YES

NO

YES

Viviparous?

Category=ratite

Carnivorous?

NO

YES

YES

NO

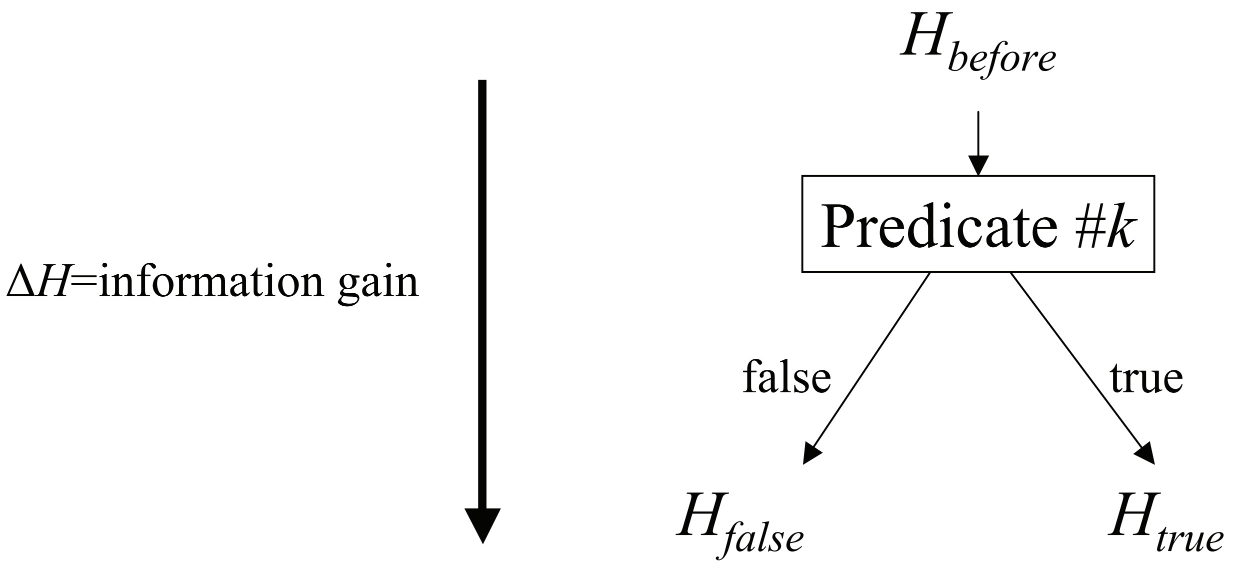
…

…

Category=raptor

Induction of Decision Trees

Grow the tree downward from the root. At each node, select the predicate that maximally reduces entropy (uncertainty):



H_{before}

Predicate #k

false

true

H_{false}

H_{true}

ΔH=information gain

$\Delta H = H_{before} - (H_{after,\mathrm{NO}} + H_{after,\mathrm{YES}})/2$ (actually uses a weighted average)

Best predicate=argmax$_k(\Delta H_k)$.

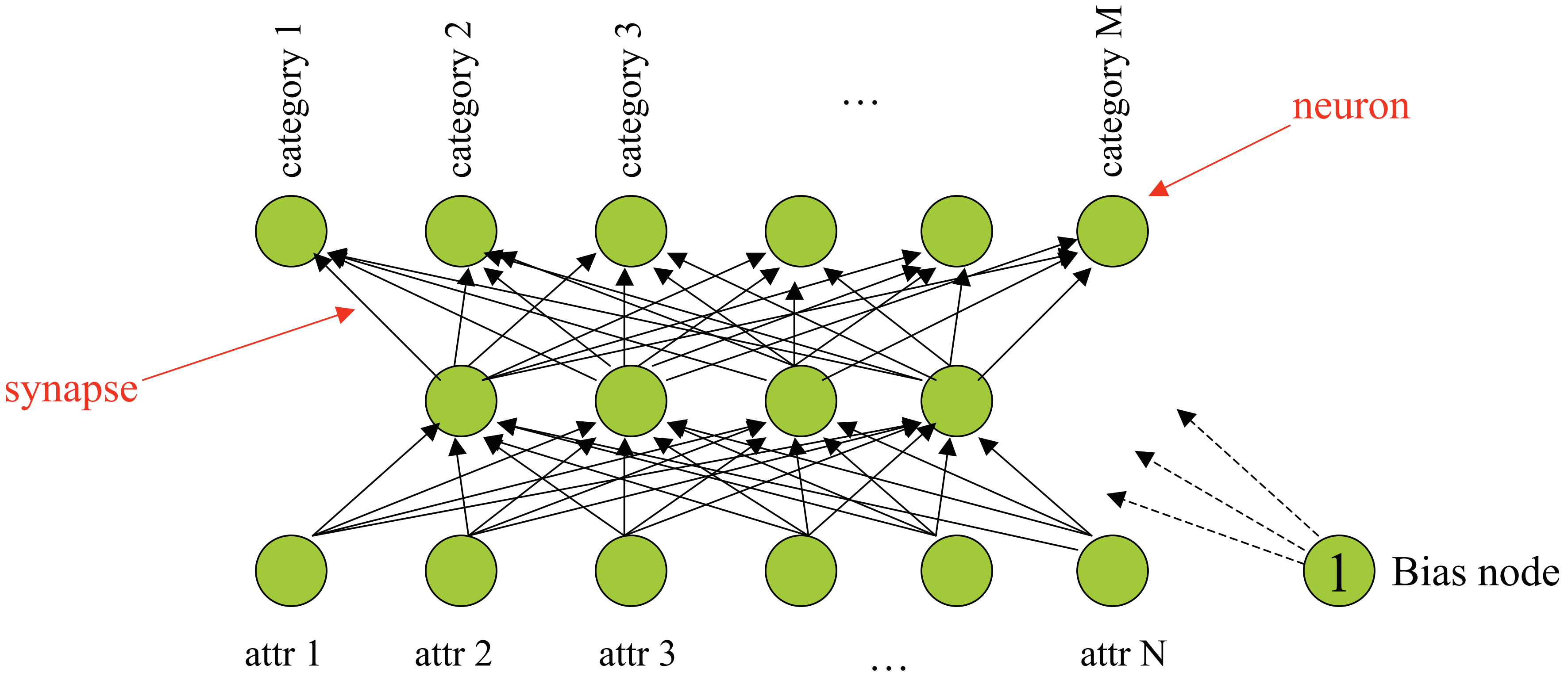
Can also use *gain ratio*, but I found no difference in performance.

Example: Training and Applying an Entropy-based Decision Tree

```
[eaglet] BioMaLL/ET> build-tree arab1.data arab1.names arab1.tree
1.97269 sec
[eaglet] BioMaLL/ET> apply-tree arab1.tree arab1.names
arab1.test arab1.predictions
0.045388 sec
[eaglet] BioMaLL/ET> ../evaluate arab1.predictions arab1.test
88.71 %
[eaglet] BioMaLL/ET> prune-by-index -c arab1.tree x 0 arab1.names
Prune index must be between 0 and 66
[eaglet] BioMaLL/ET> prune-by-index arab1.tree arab1.pruned 45
arab1.names
pruning with threshold 45 out of 67 (0.851393)
0.00304 sec
[eaglet] BioMaLL/ET> print-tree arab1.pruned arab1.names
signal2_score<=-58.5959:
|   hexamer_score<=1.51366:
|   |   category=0
|   |   signal2_score<=-74.594:
|   |   |   category=1
|   |   |   hexamer_score<=38.7892:
|   |   |   |   category=0
|   |   |   |   category=1
|   category=1
```

Backpropagation Neural Networks

Largest output = predicted category



Inputs = normalized attributes [0,1]

Transfer function: $1/(1+e^{-\Sigma \text{ inputs}})$

Train the network by gradient descent / hill-climbing.

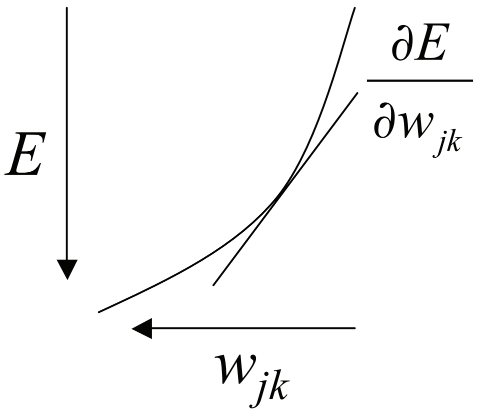
Derivation of Backprop

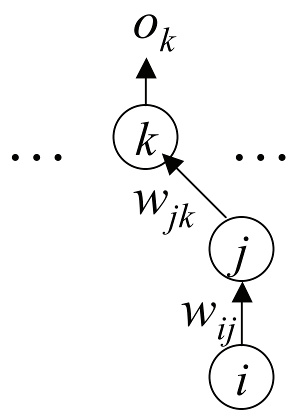
$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

$$E = \frac{1}{2} \sum_k (\overset{\text{expected}}{\downarrow\, t_k} - \overset{\text{observed}}{\downarrow\, o_k})^2$$

$$o_k = \frac{1}{1 + e^{-in_k}}$$

$$in_k = \sum_j w_{jk} o_j$$





For output layer:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial in_k} \frac{\partial in_k}{\partial w_{jk}} = \underbrace{-(t_k - o_k)}\overbrace{o_k(1 - o_k)}\underbrace{o_j}$$

Derivation, cont.

For middle layer:

$$\frac{\partial E}{\partial w_{ij}} = \sum_k \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial in_k}\frac{\partial in_k}{\partial o_j}\frac{\partial o_j}{\partial in_j}\frac{\partial in_j}{\partial w_{ij}}$$

$$= \sum_k \overline{-(t_k - o_k)}\,\overline{o_k(1-o_k)}\,\overline{w_{jk}}\,\overline{o_j(1-o_j)}\,\overline{o_i}$$

More generally for any layer containing neuron *j*,

$\Delta w_{ij} = \eta\delta_j o_i$, where

$\delta_j = (t_j - o_j)o_j(1-o_j)$ for the output layer

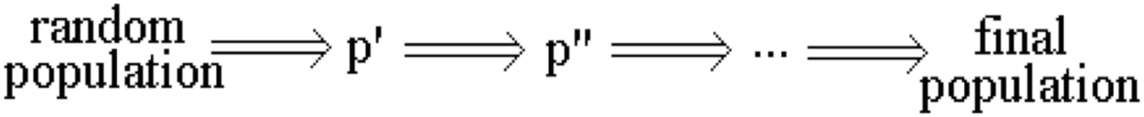
$\delta_j = o_j(1-o_j)\sum_{j\rightarrow k} w_{jk}\delta_k$ for any hidden layer (recursively follows all paths from w_{ij} to o_k)

Example: Training and Applying a Neural Network

```
[eaglet] BioMaLL/neural> cat arab1.config
maxIterations=200
learningRate=0.025
numLayers=1
neuronsPerLayer=1
networkFilename=none
min-adj=1
max-adj=1
randomize=1
noise-factor=0.99

[eaglet] BioMaLL/neural> train-net arab1.data arab1.names arab
1.config arab1.net
3.989 sec

[eaglet] BioMaLL/neural> net-classify arab1.net arab1.test arab
1.names arab1.predictions
0.027137 sec

[eaglet] BioMaLL/neural> ../evaluate arab1.predictions arab1.test
92 %
```

Genetic Algorithms

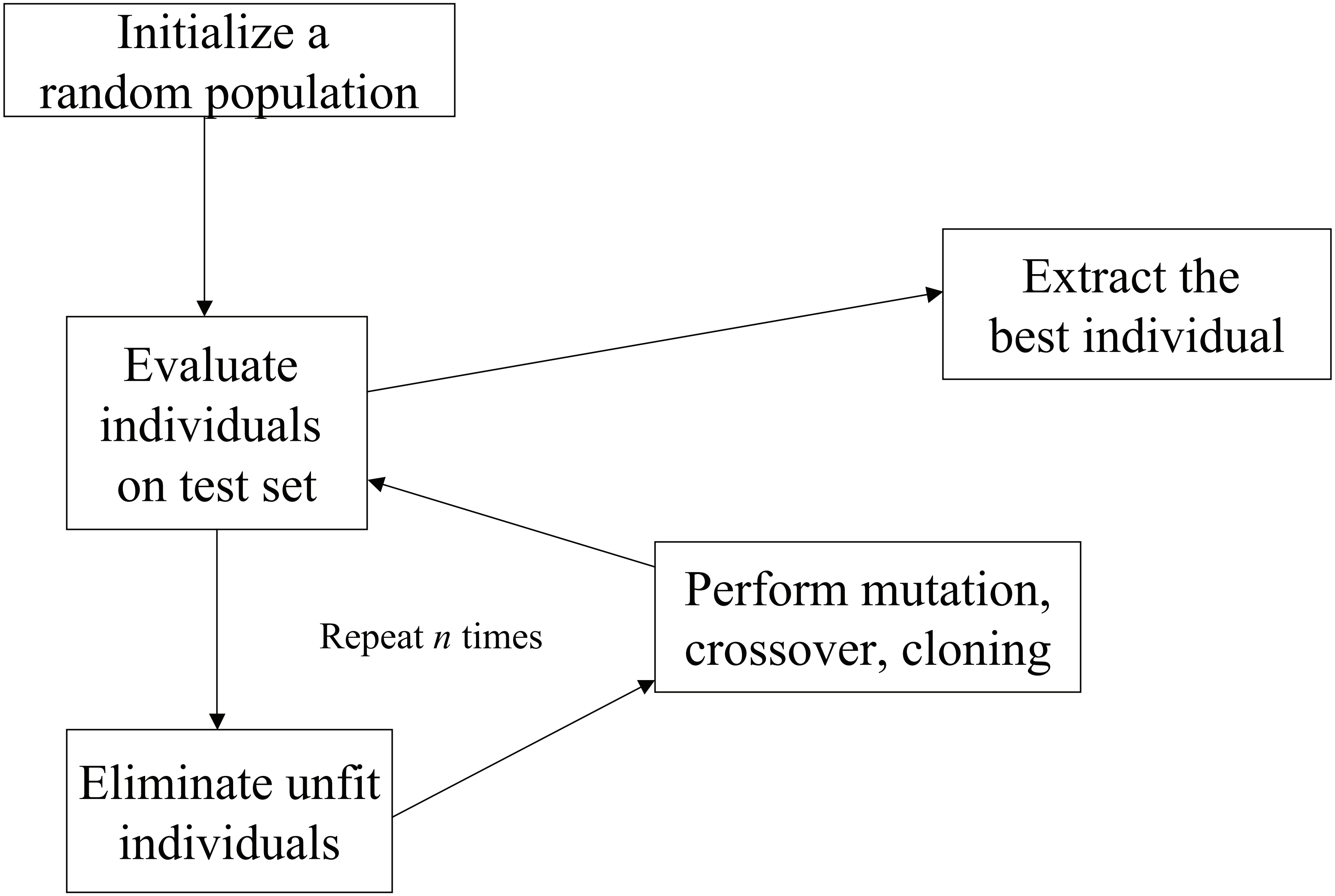
- Start with a randomly-generated population of domain objects
- Apply mutation operators (find neighbors in topological space)
- Probabilistically eliminate low-quality solutions
- Repeat until convergence



random population $\Longrightarrow$ p' $\Longrightarrow$ p'' $\Longrightarrow$ ... $\Longrightarrow$ final population

-----------------> higher average fitness

Evolutionary Algorithms



Initialize a random population

Evaluate individuals on test set

Extract the best individual

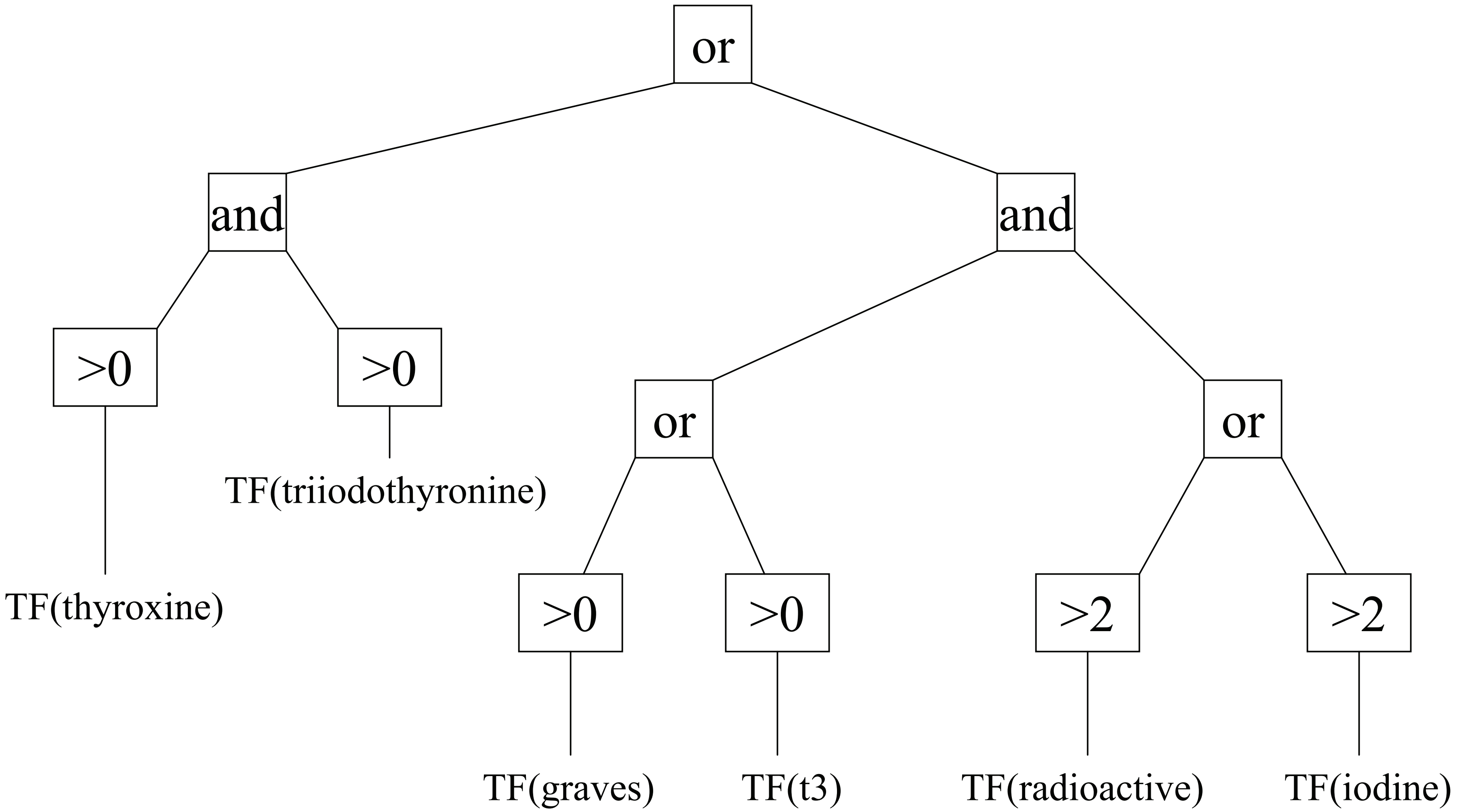
Perform mutation, crossover, cloning

Repeat *n* times

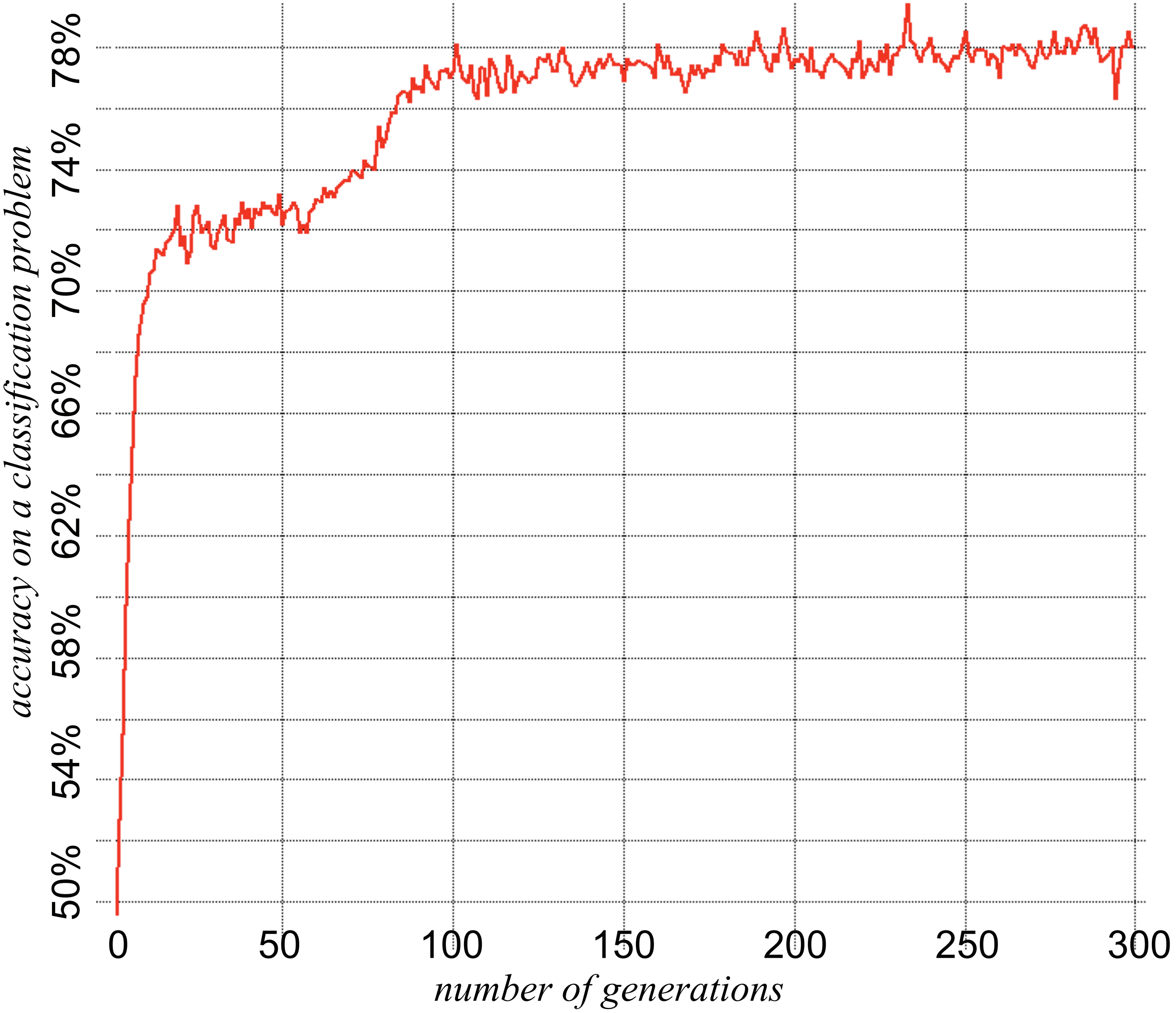
Eliminate unfit individuals

Genetic Programming

"the programming of computers by means of natural selection"



Average Fitness Over Time



Example: Evolving a Classifier using Genetic Programming

```
[eaglet] BioMaLL/GP> gp arab1
generation 0: accuracy(0.173-0.814 av=0.503) av height=0
generation 1: accuracy(0.17-0.815 av=0.537) av height=1.37
generation 2: accuracy(0.182-0.817 av=0.571) av height=1.56
...etc...
generation 298: accuracy(0.19-0.875 av=0.74) av height=2.61
generation 299: accuracy(0.19-0.875 av=0.733) av height=2.61
3.07959 min
accuracy of winner on test set: 86.1%

[eaglet] BioMaLL/GP> cat arab1.gp.tree
double mainFunction() { return if(length>sig2) then hex+length
else 1.5685/(-9.12556>sig1)); }
```

```
[eaglet] BioMaLL/GP> cat arab1.gp
max-generations         = 300
population-size         = 1000
log-file                = /dev/null
crossover               = 0.2
point-mutation          = 0.2
subtree-mutation        = 0.2
immigration             = 0.1
cloning                 = 0.3
percent-training-set    = 0.9
tournament-selection    = 0
tournament-size         = 0
min-const               = -20
max-const               = 2
initial-tree-height     = 3
max-tree-height         = 3
seed                    = 0
max-adf-call            = 10
adf-arities             = 0
entry-point-arities     = 0
result-producing-branch = 0
nonterminals            = +,-,*,/,if,<,>
terminals               = const,var
```

Simulated Annealing

Start with a random element.

Mutate the element.

If the mutant is superior, accept it. If the mutant is inferior, accept it with probability *p**.

Repeat until convergence.

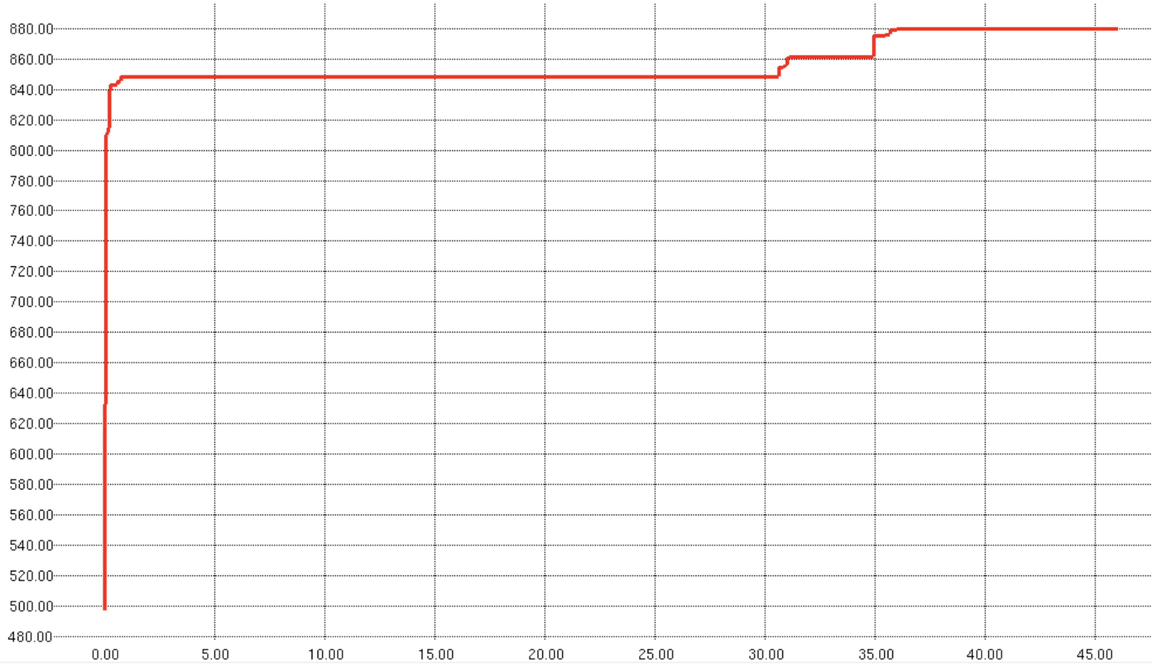
---

*p is inversely proportional to the loss in quality, and it decreases over time, as we approach convergence. It is based on the Boltzmann probability distribution, and is motivated by an analogy to the change in energy levels of molecules as the temperature is slowly decreased (i.e., time elapses).

$$\boxed{p = e^{\frac{-\triangle E}{kT}}}$$

Example: Simulated Annealing

```
[eaglet] BioMaLL/annealing> anneal arab1.config arab1.names arab
1.data arab1.tree
45.2363 sec
final accuracy on training set: 88%

[eaglet] BioMaLL/annealing> cat arab1.tree
double mainFunction() { return (((length-8.1023)/0.206617)-(if(-
1 1.1519) then sig1 else hex+(-10.4093-hex))); }
```



Accuracy (y-axis) vs. generations (x-axis) of simulated annealing. K=2.8e-10, initial temperature=100, final temperature = 1, temperature decay factor = 0.9999.

Feature Selection Methods

- F-ratio : select features exhibiting large $F=MS_{between}/MS_{within}$
- PCA : recode problem into principal components
- LDA : recode problem using linear discriminant functions
- Mutual Information *(not yet implemented)*
- Information Gain *(not yet implemented)*
- χ^2 *(not yet implemented)*
- Fisher-exact test *(not yet implemented)*

Example: Feature Selection via F-ratio

```
[eaglet] BioMaLL/f-ratio> f-ratio arab1.names arab1.data
F(length)=114.693
F(sig1)=477.876
F(sig2)=259.967
F(hex)=359.783
```

Example: Feature Selection via Principle Components Analysis

```
[eaglet] BioMaLL/PCA> pca arab1.data arab1.names arab1.model 3
rounded eigenvalues: 2512, 273, 43, 0
component 0: 4 -0.00701833  -0.0277586  0.0671525      0.997332
component 1: 4 -0.0193795    0.742358   -0.666521      0.0654039
component 2: 4 0.114123     -0.663282   -0.73892       0.0320951
[eaglet] BioMaLL/PCA> apply-components arab1.model arab1.names
arab1.data arab1.recoded
[eaglet] BioMaLL/PCA> head -5 arab1.data
-7.22008        -46.4053        -81.4875        15.5713     1
-7.08321        -56.6218        -65.6119        -15.9614    0
-6.1875         -40.117         -80.3785        -13.286     0
-7.18202        -56.4384        -65.6939        -5.89178    0
-5.51827        -51.3482        -76.2935        -6.37986    1
[eaglet] BioMaLL/PCA> head -5 arab1.recoded
11.3965         21.0221               90.6683       1
-18.7034        0.791394              84.7175       0
-17.4912        23.0437               84.8696       0
-8.67051        1.6427                84.9684       0
-10.0221        12.4221               89.5986       1
```

Part III

Sample Data Sets

Distinguishing Exons from Non-Exons

Exons (category 1) were randomly selected from the annotated DNA of a target genome. Non-exons (category 0) were obtained by randomly sampling open reading frames (ORFs) from DNA containing both coding and noncoding segments -- overlap with true exons was not prevented and probably occurred; thus, some non-exons will have characteristics similar to exons. Numbers of true and false exons were roughly equal in all data sets.

Input features:

1. weight matrix score of the first signal (acceptor splice site start-codon)
2. weight matrix score of the second signal (donor splice site or stop-codon)
3. exon length probability (from empirical training distribution of true exons)
4. hexamer score = ∑ log P(H|coding)/P(H) over all hexamers H in the interval

Categories:

- 0 = not an exon
- 1 = an exon

Data sets:

arab1 = arabidopsis thaliana
human1 = homo sapiens
aspergillus1 = aspergillus fumigatus