Lecture 6

Genetic Algorithms

Model Ensembles

Genetic Algorithms

- Evolutionary computation
- Prototypical GA
- An example: GABIL
- Schema theorem
- Genetic programming
- The Baldwin effect

Evolutionary Computation

1. Computational procedures patterned after biological evolution
2. Search procedure that probabilistically applies search operators to set of points in the search space

Biological Evolution

Lamarck:
- Species “transmute” over time

Darwin:
- Consistent, heritable variation among individuals in population
- Natural selection of the fittest

Mendel/Genetics:
- A mechanism for inheriting traits
- Mapping: Genotype $\rightarrow$ Phenotype

Representing Hypotheses

Represent

$(\text{Outlook} = \text{Overcast} \lor \text{Rain}) \land (\text{Wind} = \text{Strong})$

by

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overcast</td>
<td>10</td>
<td>yes</td>
</tr>
</tbody>
</table>

Represent

IF Wind = Strong THEN PlayTennis = yes

by

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

GA($\text{Fitness}$, $\text{Fitness.threshold}$, $p$, $r$, $m$)

- Initialize: $P \leftarrow p$ random hypotheses
- Evaluate: for each $h$ in $P$, compute $\text{Fitness}(h)$
- While $\left[ \max_{h \in P} \text{Fitness}(h) \right] < \text{Fitness.threshold}$
  1. Select: Randomly select $(1 - r)p$ members of $P$ to add to $P_0$.
     $\Pr(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{h \in P} \text{Fitness}(h)}$
  2. Crossover: Randomly select $\frac{r}{2}$ pairs of hypotheses from $P$. For each pair $(h_1, h_2)$, produce two offspring by crossover. Add all offspring to $P_0$.
  4. Update: $P \leftarrow P_0$
  5. Evaluate: for each $h$ in $P$, compute $\text{Fitness}(h)$
- Return hypothesis from $P$ with highest fitness.
Operators for Genetic Algorithms

<table>
<thead>
<tr>
<th>Single-point crossover</th>
<th>Two-point crossover</th>
<th>Uniform crossover</th>
<th>Point mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Single-point crossover" /></td>
<td><img src="image" alt="Two-point crossover" /></td>
<td><img src="image" alt="Uniform crossover" /></td>
<td><img src="image" alt="Point mutation" /></td>
</tr>
</tbody>
</table>

Selecting Fittest Hypotheses

Fitness-proportionate selection:

\[ \text{Pr}(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^{n} \text{Fitness}(h_j)} \]

... can lead to crowding

Tournament selection:
- Pick \( h_1, h_2 \) at random with uniform probability
- With probability \( p \), select the more fit

Rank selection:
- Sort all hypotheses by fitness
- Prob. of selection is proportional to rank

Example: The GABIL System

Learn disjunctive set of propositional rules

Competitive with C4.5

**Fitness:** \( \text{Fitness}(h) = (\text{correct}(h))^2 \)

**Representation:**

IF \( a_1 = T \land a_2 = F \) THEN \( c = T \); IF \( a_2 = T \) THEN \( c = F \)

represented by

\[
\begin{align*}
  a_1 & a_2 & c & a_1 & a_2 & c \\
  10 & 01 & 1 & 11 & 10 & 0
\end{align*}
\]

Genetic operators:
- Want variable length rule sets
- Want only well-formed bitstring hypotheses

Crossover with Variable-Length Bitstrings

Start with

\[
\begin{align*}
h_1 &= a_1 \ a_2 \ c \ a_1 \ a_2 \ c \\
    &= 10 \ 01 \ 1 \ 11 \ 10 \ 0 \\
h_2 &= a_1 \ a_2 \ c \ a_1 \ a_2 \ c \\
    &= 01 \ 11 \ 0 \ 10 \ 01 \ 0
\end{align*}
\]

1. Choose crossover points for \( h_1 \), e.g., after bits 1, 8
2. Now restrict points in \( h_2 \) to those that produce bitstrings with well-defined semantics, e.g., \( (1,3), (1,8), (6,8) \).
GABIL Extensions
Add new genetic operators, also applied probabilistically:
1. AddAlternative: generalize constraint on a_i by changing it 0 to 1
2. DropCondition: generalize constraint on a_i by changing it every 0 to 1
And add new field to bitstring to determine whether to allow these
\[ a_1 \ a_2 \ c \ a_1 \ a_2 \ c \ AA \ DC \]
01 11 0 10 01 0 1 0
So now the learning strategy also evolved!

Schemas
How to characterize evolution of population in GA?
Schema = string containing 0, 1, * ("don’t care")
- Typical schema: 10*0*
- Instances of above schema: 10101, 10000, ...
Characterize population by number of instances representing each possible schema
- \( m(s, t) = \# \) instances of schema \( s \) in pop, at time \( t \)

Consider Just Selection
- \( f(t) = \) average fitness of pop. at time \( t \)
- \( m(s, t) = \) instances of schema \( s \) in pop. at time \( t \)
- \( \tilde{u}(s, t) = \) average fitness of instances of \( s \) at time \( t \)

Probability of selecting \( h \) in one selection step
\[
Pr(h) = \frac{f(h)}{\sum_{h \in \Phi} f(h)} = \frac{f(h)}{n_f(t)}
\]

Expected number of instances of \( s \) after \( n \) selections
\[
E[m(s, t + 1)] = \frac{\tilde{u}(s, t)}{f(t)} m(s, t)
\]

Schema Theorem
\[
E[m(s, t + 1)] \geq \frac{\tilde{u}(s, t)}{f(t)} m(s, t) \left( 1 - p_c f(t) - 1 \right) (1 - p_m)^{w(s)}
\]
- \( m(s, t) = \) instances of schema \( s \) in pop at time \( t \)
- \( f(t) = \) average fitness of pop. at time \( t \)
- \( \tilde{u}(s, t) = \) ave. fitness of instances of \( s \) at time \( t \)
- \( p_c = \) probability of single point crossover operator
- \( p_m = \) probability of mutation operator
- \( l = \) length of single bit strings
- \( w(s) \) number of defined (non "*") bits in \( s \)
- \( d(s) = \) dist. between left & rightmost defined bits in \( s \)

Genetic Programming
Population of programs represented by trees
E.g.: \( \sin(x) + \sqrt{x^2 + y} \)
**Crossover**

![Crossover Diagram]

**Example: Electronic Circuit Design**
- Individuals are programs that transform beginning circuit to final circuit, by adding/subtracting components and connections
- Use population of 640,000, run on 64-node parallel processor
- Discovers circuits competitive with best human designs

---

**Biological Evolution**

Lamarck (19th century)

- Believed individual genetic makeup was altered by lifetime experience
- But current evidence contradicts this view

What is the impact of individual learning on population evolution?

---

**Baldwin Effect**

**Assume**
- Individual learning has no direct influence on individual DNA
- But ability to learn reduces need to “hard wire” traits in DNA

**Then**
- Ability of individuals to learn will support more diverse gene pool, because learning allows individuals with various “hard wired” traits to be successful
- More diverse gene pool will support faster evolution of gene pool

⇒ Individual learning increases rate of evolution

---

**Baldwin Effect**

Plausible example:
1. New predator appears in environment
2. Individuals who can learn (to avoid it) will be selected
3. Increase in learning individuals will support more diverse gene pool
4. Resulting in faster evolution
5. Possibly resulting in new non-learned traits such as instinctive fear of predator

---

**Computer Experiments on Baldwin Effect**

Evolve simple neural networks:
- Some network weights fixed, others trainable
- Genetic makeup determines which are fixed, and their weight values

Results:
- With no individual learning, population failed to improve over time
- When individual learning allowed
  - Early generations: population contained many individuals with many trainable weights
  - Later generations: higher fitness, while number of trainable weights decreased
Genetic Algorithms: Summary
- Evolving algorithms by natural selection
- Genetic operators avoid (some) local minima
- Why it works: schema theorem
- Genetic programming
- Baldwin effect

Model Ensembles
- Basic idea:
  Instead of learning one model, learn several and combine them
- Typically improves accuracy, often by a lot
- Many methods:
  - Bagging
  - Boosting
  - ECOC (error-correcting output coding)
  - Stacking
  - Etc.

Bagging
- Generate “bootstrap” replicates of training set
  by sampling with replacement
- Learn one model on each replicate
- Combine by uniform voting

Boosting
- Maintain vector of weights for examples
- Initialize with uniform weights
- Loop:
  - Apply learner to weighted examples (or sample)
  - Increase weights of misclassified examples
- Combine models by weighted voting

\[\text{AdaBoost}(S, \text{Learn}, k)\]
\[S: \text{Training set } \{(x_1, y_1), \ldots, (x_m, y_m)\}, \ y_i \in Y\]
\[\text{Learn}: \text{Learn}(S, \text{weights})\]
\[k: \# \text{Rounds}\]
For all \(i\) in \(S: w_1(i) = 1/m\)
For \(r = 1\) to \(k\) do
  For all \(i\): \(p_r(i) = w_r(i)/\sum w_r(i)\)
  \(h_r = \text{Learn}(S, p_r)\)
  \(\epsilon_r = \sum p_r(i) \mathbf{1}[h_r(i) \neq y_i]\)
  If \(\epsilon_r > 1/2\) then
    \(k = r - 1\)
  Exit
  \(\beta_r = \epsilon_r/(1 - \epsilon_r)\)
For all \(i\): \(w_{r+1}(i) = w_r(i) \beta_r^{1-h_r(i)x_i}\)
Output: \(h(x) = \text{argmax}_{y \in Y} \sum_{r=1}^{k} (\log \frac{1}{\beta_r}) \mathbf{1}[h_r(x) = y]\)
Error-Correcting Output Coding

- **Motivation:** Applying binary classifiers to multiclass problems
- **Train:** Repeat $L$ times:
  - Form a binary problem by randomly assigning classes to “superclasses” 0 and 1
    E.g.: A, B, D $\rightarrow$ 0; C, E $\rightarrow$ 1
  - Apply binary learner to binary problem
- **Test:**
  - Apply each classifier to test example, forming vector of predictions $\mathbf{P}$
  - Predict class whose vector is closest to $\mathbf{P}$ (Hamming)

Stacking

- Apply multiple base learners (e.g.: decision trees, naive Bayes, neural nets)
- Meta-learner: Inputs = Base learner predictions
- Training by leave-one-out cross-validation:
  Meta L. inputs = Predictions on left-out examples

Model Ensembles: Summary

- Learn several models and combine them
- Bagging: Random resamples
- Boosting: Weighted resamples
- ECOC: Recode outputs
- Stacking: Multiple learners