# genee

A Learning Classifier based on Genetic Algorithms

Patric Fornasier
patricf@cse.unsw.edu.au

Version 0.1 – June 8, 2006

**Abstract**

Genetic algorithms are a particular class of algorithms that provide an approach to learning inspired by the model of biological evolution, i.e. selection, crossover and mutation. The goal of this project was to implement a genetic algorithm including different crossover rules. Two standard UCI data sets were given, to evaluate the performance of the implementation. WEKA [8] has been chosen as a framework to facilitate the development.

## 1 Introduction

As in other machine learning techniques, genetic algorithms involve searching through a space of possible hypotheses. The approach taken by genetic algorithms to find the best hypothesis is the following: The algorithm randomly creates a number of hypotheses (the individuals forming the population) and evaluates their performance (fitness) over the given training set. Following that it will probabilistically select a set of hypotheses according to their fitness to be taken into the new generation. Similarly it will select another set of hypothesis pairs that will be recombined to form new hypotheses (offspring). Finally, the algorithm will combine these two sets to form a new generation and pick some hypotheses for mutation, by randomly changing some of their values. The hypotheses of the new generation will subsequently be re-evaluated based on their fitness and the process of selection, crossover and mutation will be repeated. The algorithm stops when a certain termination criterion has been reached, e.g. a hypothesis with a minimum fitness has been found or the number of maximum generations has been exhausted.

The remainder of this paper is organized as follows. In Section 2, I introduce the basic idea of the software implementation, which has been the biggest part of this work. This implementation is then put to test in Section 3, where I also discuss the results. Finally, I evaluate the results in Section 4 and map out directions for future work.

## 2 Implementation

One of the first problems that I addressed was how to encode the hypothesis into a form suitable for easy manipulation. First, I considered representing the hypothesis as a tree, since tree structures are well understood and easy to manipulate. However one of the requirements of the project was to implement single-point, two-point or uniform crossover rules and these kind of crossover rules didn't seem to be very well applicable to tree structures. That's why I decided to encode the hypotheses, i.e. the rules, as it is traditionally done in bit strings of 0s and 1s.

For every possible value of each attribute in the rule, one bit is reserved. This implies that all the rules have the same rule length, i.e. number of bits, and that the order of the attributes is crucial.

Note that the class attribute is subject to the additional constraint that at any given time, only one bit can be set. This will avoid creating rules with disjunctive consequents. The hypothesis on the other end is merely a set of an arbitrary number of disjunctive rules.

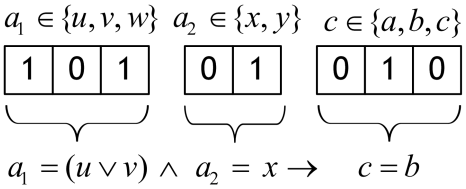


Figure 1: Rule encoding

The genetic algorithm itself has been implemented as a WEKA classifier. On one side this allowed me to take advantage of the functionality provided by the WEKA library and on the other side I leveraged WEKA by providing an additional classifier that can easily be integrated into the framework [7]. The core of the algorithm roughly works as outlined in section 1. The classifier has been designed in a flexible way that allows further extension. For example users can provide their own `Fitness` or `Crossover` implementations at runtime, without having to modify the existing code.

The software package [4] has been developed in-line with current best practices in open-source software engineering, e.g. unit tests [2], automated build including project report and metrics generation [5], version control [1], and can be obtained along with further documentation under the LGPL license from SourceForge [3].

# 3 Experimentation

The classifier has been tested on the two given data sets using 3-fold cross-validation and different parameters. Below are two charts that show the evolution on the mushroom dataset over roughly 1300 generations with a population size of 100 individuals. Further experimentation results are summarized in the appendix.

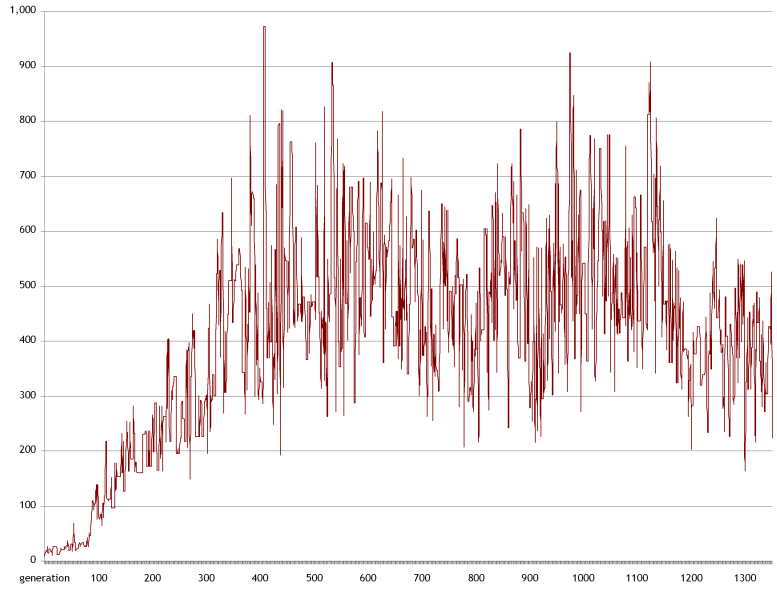


Figure 2: Number of rules per generation

The performance in terms of correctly classified instances is quite acceptable, if the correct parameters are chosen. However, the time to build the classifier seems to take very long, especially when hypotheses contain a large number of rules.

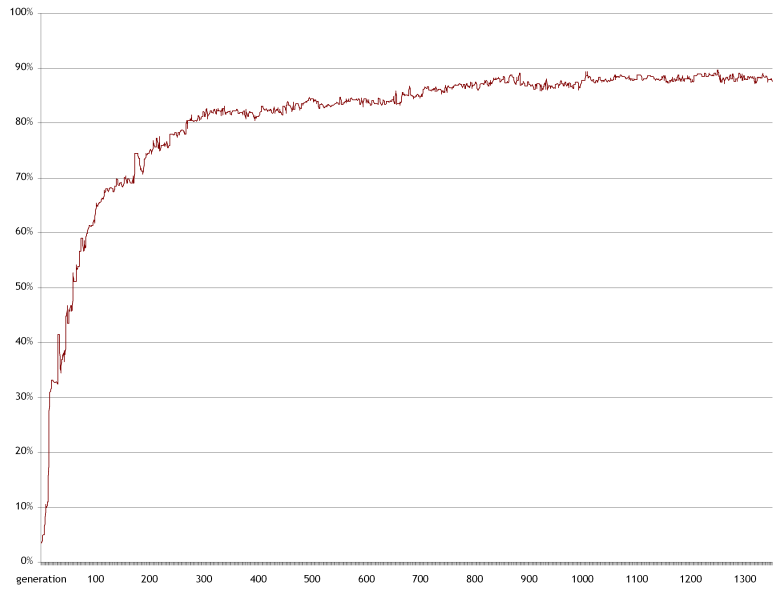


Figure 3: Number of correctly classified instances per generation

# 4 Conclusion and Future Work

In this project I have shown how to implement a genetic algorithm to learn from data. Also, I have demonstrated how to leverage WEKA and make use of some of the rich features it provides.

The project should be seen as a starting point for further development. It provides a solid and thoroughly tested base that serves as a proof of concept. However, there is certainly a lot of room for improvement in some of the performance aspects, especially in regard of the time it takes to build the classifier. Also, it might be worthwhile to explore different `Fitness` implementations. For example: additionally considering hypothesis complexity, i.e. number of rules, in the fitness calculation might help to avoid overfitting of the data.

# References

[1] CVS. http://www.nongnu.org/cvs/.

[2] JUnit. http://www.junit.org/.

[3] SourceForge.net. http://www.sourceforge.net/.

[4] Patric Fornasier. orcus-genee. http://orcus.sourceforge.net/genee.

[5] The Apache Software Foundation. Maven2. http://maven.apache.org/.

[6] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[7] The University of Waikato. GenericObjectEditor and GenericPropertiesCreator. http://www.cs.waikato.ac.nz/ml/weka/goe/.

[8] The University of Waikato. WEKA. http://www.cs.waikato.ac.nz/ml/weka/.

# Appendix

Below are two excerpts of the output that has been generated by the algorithm for the given two data sets. In both sets 3-fold cross-validation has been used. Due to time-constraints the mushroom datasets could only be tested with 100 generations.

### mushroom.arff

```
Options: -G 100 -P 200

genee.GeneticClassifier@ee1abe[
  hypothesis=genee.model.Hypothesis@f45a93[size=20320,numRules=160]
  numTotal=8124
  numCovered=5596
  numUncovered=2528
  numCorrect=5150
  numIncorrect=446
  correct=0.6339241752831117
  inCorrect=0.3660758247168882
  coverage=0.6888232397833579
  accuracy=0.9203002144388849
]]

Time taken to build model: 2610.19 seconds
Time taken to test model on training data: 0.28 seconds

=== Error on training data ===

Correctly Classified Instances        5150               63.3924 %
Incorrectly Classified Instances       446                5.4899 %
Kappa statistic                          0.8345
Mean absolute error                      0.0549
Root mean squared error                  0.2343
Relative absolute error                 16.0219 %
Root relative squared error             56.7162 %
UnClassified Instances                2528               31.1177 %
Total Number of Instances             8124

=== Confusion Matrix ===

    a    b   <-- classified as
 3136   60 |    a = e
  386 2014 |    b = p

=== Stratified cross-validation ===

Correctly Classified Instances        5614               69.1039 %
Incorrectly Classified Instances       986               12.1369 %
```

```
Kappa statistic                          0.702
Mean absolute error                      0.1214
Root mean squared error                  0.3484
Relative absolute error                 29.9595 %
Root relative squared error             77.4619 %
UnClassified Instances                1524               18.7592 %
Total Number of Instances             8124

=== Confusion Matrix ===

    a    b   <-- classified as
 2872  675 |    a = e
  311 2742 |    b = p
```

### balance-scale.arff

```
Options: -C 0.8 -G 200 -P 100

net.sf.orcus.genee.GeneticClassifier@48ff61[
  hypothesis=net.sf.orcus.genee.model.Hypothesis@1302bb6[size=2438,numRules=106]
  numTotal=625
  numCovered=625
  numUncovered=0
  numCorrect=430
  numIncorrect=195
  correct=0.688
  inCorrect=0.312
  coverage=1.0
  accuracy=0.688
]]

Time taken to build model: 147.01 seconds
Time taken to test model on training data: 0.03 seconds

=== Error on training data ===

Correctly Classified Instances         430               68.8    %
Incorrectly Classified Instances       195               31.2    %
Kappa statistic                          0.4214
Mean absolute error                      0.208
Root mean squared error                  0.4561
Relative absolute error                 54.7709 %
Root relative squared error            104.7048 %
Total Number of Instances              625

=== Confusion Matrix ===
```

```
   a   b   c   <-- classified as
 227   0  61 |   a = L
  28   0  21 |   b = B
  85   0 203 |   c = R
```

```
=== Stratified cross-validation ===
```

| | | |
|---|---|---|
| Correctly Classified Instances | 429 | 68.64 % |
| Incorrectly Classified Instances | 196 | 31.36 % |
| Kappa statistic | 0.4269 | |
| Mean absolute error | 0.2091 | |
| Root mean squared error | 0.4572 | |
| Relative absolute error | 55.0292 % | |
| Root relative squared error | 104.9719 % | |
| Total Number of Instances | 625 | |

```
=== Confusion Matrix ===

   a   b   c   <-- classified as
 206   6  76 |   a = L
  23   1  25 |   b = B
  60   6 222 |   c = R
```