Negative Selection Algorithm*

Jason Brownlee
jasonb@CleverAlgorithms.com
The Clever Algorithms Project
http://www.CleverAlgorithms.com

May 16, 2010
Technical Report: CA-TR-20100516-1

Abstract

The Clever Algorithms project aims to describe a large number of Artificial Intelligence algorithms in a complete, consistent, and centralized manner, to improve their general accessibility. The project makes use of a standardized algorithm description template that uses well-defined topics that motivate the collection of specific and useful information about each algorithm described. This report describes the Negative Selection Algorithm using the standardized template.

Keywords: `Clever, Algorithms, Description, Optimization, Negative, Selection`

1 Introduction

The Clever Algorithms project aims to describe a large number of algorithms from the fields of Computational Intelligence, Biologically Inspired Computation, and Metaheuristics in a complete, consistent and centralized manner [2]. The project requires all algorithms to be described using a standardized template that includes a fixed number of sections, each of which is motivated by the presentation of specific information about the technique [3]. This report describes the Negative Selection Algorithm using the standardized template.

2 Name

Negative Selection Algorithm, NSA

3 Taxonomy

The Negative Selection Algorithm belongs to the field of Artificial Immune Systems. The algorithm is related to other Artificial Immune System algorithms such as the Clonal Selection Algorithm, and the Immune Network Algorithm.

4 Inspiration

The Negative Selection algorithm is inspired by the self-nonself discrimination behavior observed in the mammalian acquired immune system. The clonal selection theory of acquired immunity accounts for the adaptive behavior of the immune system including the ongoing selection and proliferation of cells that select-for potentially harmful (and typically foreign) material in the body. An interesting aspect of this process is that it is responsible for managing a population of immune cells that do not select-for the tissues of the body , specifically it does not create self-reactive immune cells known as auto-immunity. This problem is known as 'self-nonself discrimination' and it involves the preparation and on going maintenance of a repertoire of immune cells such that none are auto-immune. This is achieved by a negative selection process that selects-for and removes those cells that are self-reactive during cell creation and cell proliferation. This process has been observed in the preparation of T-lymphocytes, naïve version of which are matured using both a positive and negative selection process in the thymus.

5 Metaphor

The self-nonself discrimination principle suggests that the anticipatory guesses made in clonal selection are filtered by regions of infeasibility (protein conformations that bind to self-tissues). Further, the self-nonself immunological paradigm proposes the modeling of the unknown domain (encountered pathogen) by modeling the complement of what is known. This is unintuitive as the natural inclination is to categorize unknown information by what is different from that which is known, rather than guessing at the unknown information and filtering those guesses by what is known.

6 Strategy

The information processing principles of the self-nonself discrimination process via negative selection are that of a anomaly and change detection systems that model the anticipation of variation from what is known. The principle is achieved by building a model of changes, anomalies, or unknown (non-normal or non-self) data by generating patterns that do not match an existing corpus of available (self or normal) patterns. The prepared non-normal model is then used to either monitor the existing normal data or streams of new data by seeking matches to the non-normal patterns.

7 Procedure

Algorithm 1 provides a pseudo-code listing of the detector generation procedure for the Negative Selection Algorithm. Algorithm 2 provides a pseudo-code listing of the detector application procedure for the Negative Selection Algorithm.

8 Heuristics

- The Negative Selection Algorithm was designed for change detection novelty detection, intrusion detection and similar pattern recognition and two-class classification problem domains.
- Traditional negative selection algorithms used binary representations and binary matching rules such as Hamming distance, and r-contiguous bits.

Algorithm 1: Pseudo Code for detector generation in the Negative Selection Algorithm.

```
Input: SelfData
Output: Repertoire
Repertoire ← 0;
while ¬StopCondition() do
    Detectors ← GenerateRandomDetectors();
    foreach Detector_i ∈ Repertoire do
        if ¬Matches(Detector_i, SelfData) then
            Repertoire ← Detector_i;
        end
    end
end
return Repertoire;
```

Algorithm 2: Pseudo Code for detector application in the Negative Selection Algorithm.

```
Input: InputSamples, Repertoire
for Input_i ∈ InputSamples do
    Inputi_class ← "non-self";
    foreach Detector_i ∈ Repertoire do
        if Matches(Input_i, Detector_i) then
            Inputi_class ← "self";
            Break();
        end
    end
end
```

- A data representation should be selected that is most suitable for a given problem domain, and a matching rule is in turn selected or tailored to the data representation.
- Detectors can be prepared with no prior knowledge of the problem domain other than the known (normal or self) dataset.
- The algorithm can be configured to balance between detector convergence (quality of the matches) and the space complexity (number of detectors).
- The lack of dependence between detectors means that detector preparation and application is inherently parallel and suited for a distributed and parallel implementation, respectively.

9 Code Listing

Listing 1 provides an example of the Negative Selection Algorithm implemented in the Ruby Programming Language. The demonstration problem is a two-class classification problem where samples are drawn from a two-dimensional domain, where $x_i \in [0, 1]$. Those samples in $1.0 > x_i > 0.5$ are classified as self and the rest of the space belongs to the non-self class. Samples are drawn from the self class and presented to the algorithm for the preparation of pattern detectors for classifying unobserved samples from the non-self class. The algorithm creates a set of detectors that do not match the self data, and are then applied to a set of randomly generated samples from the domain. The algorithm uses a real-valued representation. The Euclidean distance function is used during matching and a minimum distance value is specified as a user parameter for approximate matches between patterns. The algorithm includes the

additional computationally expensive check for duplicates in the preparation of the self dataset and the detector set.

```
def random_vector(search_space)
  return Array.new(search_space.length) do |i|
    search_space[i][0] + ((search_space[i][1] - search_space[i][0]) * rand())
  end
end

def contains?(vector, space)
  vector.each_with_index do |v,i|
    return false if v<space[i][0] or v>space[i][1]
  end
  return true
end

def euclidean_distance(v1, v2)
  sum = 0.0
  v1.each_with_index do |v,i|
    diff = v - v2[i]
    sum += (diff*diff)
  end
  return Math.sqrt(sum)
end

def matches?(bitstring, dataset, min_distance)
  dataset.each do |pattern|
    score = euclidean_distance(bitstring, pattern[:vector])
    return true if score <= min_distance
  end
  return false
end

def generate_detectors(max_detectors, search_space, self_dataset, min_distance)
  detectors = []
  begin
    detector = {}
    detector[:vector] = random_vector(search_space)
    if !matches?(detector[:vector], self_dataset, min_distance)
      next if matches?(detector[:vector], detectors, 0.0)
      detectors << detector
    end
  end while detectors.size < max_detectors
  return detectors
end

def apply_detectors(num_test, detectors, search_space, self_space, min_distance)
  correct = 0
  num_test.times do |i|
    input = {}
    input[:vector] = random_vector(search_space)
    predicted = matches?(input[:vector], detectors, min_distance) ? "non-self" : "self"
    actual = contains?(input[:vector], self_space) ? "self" : "non-self"
    result = (predicted==actual) ? "Correct" : "Incorrect"
    correct += 1.0 if predicted==actual
    puts "#{i+1}/#{num_test}: #{result} - predicted=#{predicted}, actual=#{actual},
        vector=#{input[:vector].inspect}"
  end
  puts "Total Correct: #{correct}/#{num_test} (#{(correct/num_test.to_f)*100.0}%)"
end

def generate_self_dataset(num_records, self_space, search_space)
  self_dataset = []
  begin
```

```
    pattern = {}
    pattern[:vector] = random_vector(search_space)
    next if matches?(pattern[:vector], self_dataset, 0.0)
    if contains?(pattern[:vector], self_space)
      self_dataset << pattern
    end
  end while self_dataset.length < num_records
  return self_dataset
end

max_detectors = 300
max_self = 150
min_distance = 0.05
num_test = 50
problem_size = 2
search_space = Array.new(problem_size) {[0.0, 1.0]}
self_space = Array.new(problem_size) {[0.5, 1.0]}
self_dataset = generate_self_dataset(max_self, self_space, search_space)
puts "Done: prepared #{self_dataset.size} self patterns."
detectors = generate_detectors(max_detectors, search_space, self_dataset, min_distance)
puts "Done: prepared #{detectors.size} detectors."
apply_detectors(num_test, detectors, search_space, self_space, min_distance)
puts "Done. completed testing."
```

Listing 1: Negative Selection Algorithm in the Ruby Programming Language

10 References

10.1 Primary Sources

The seminal negative selection algorithm was proposed by Forrest, et al. [8] in which a population of detectors are prepared in the presence of known information, where those randomly generated detectors that match against known data are discarded. The population of pattern guesses in the unknown space then monitors the corpus of known information for changes. The algorithm was applied to the monitoring of files for changes (corruptions and infections by computer viruses), and later formalized as a change detection algorithm [6, 5].

10.2 Learn More

The Negative Selection algorithm has been applied to the monitoring of changes in the execution behavior of Unix processes [7, 11], and to monitor changes in remote connections of a network computer (intrusion detection) [9, 10]. The application of the algorithm has been predominantly to virus host intrusion detection and their abstracted problems of classification (two-class) and anomaly detection. Esponda provides some interesting work showing some compression and privacy benefits provided by maintaining a negative model (non-self) [4] Ji and Dasgupta provide a contemporary and detailed review of Negative Selection Algorithms covering topics such as data representations, matching rules, detector generation procedures, computational complexity, hybridization, and theoretical frameworks [12]. More recently, the validity of the application of negative selection algorithms in high-dimensional spaces has been questioned, specifically given the scalability of the approach in the face of the exponential increase in volume within the problem space [13].

11 Conclusions

This report described the Negative Selection algorithm from the field of Artificial Immune systems. Elements of this report were drawn from some of the author's previous works including

his dissertation work [1].

12 Contribute

Found a typo in the content or a bug in the source code? Are you an expert in this technique and know some facts that could improve the algorithm description for all? Do you want to get that warm feeling from contributing to an open source project? Do you want to see your name as an acknowledgment in print?

Two pillars of this effort are i) that the best domain experts are people outside of the project, and ii) that this work is (somewhat) wrong by default. Please help to make this work less wrong by emailing the author 'Jason Brownlee' at jasonb@CleverAlgorithms.com or visit the project website at http://www.CleverAlgorithms.com.

References

[1] Jason Brownlee. *Clonal Selection as an Inspiration for Adaptive and Distributed Information Processing*. PhD thesis, Complex Intelligent Systems Laboratory, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2008.

[2] Jason Brownlee. The clever algorithms project: Overview. Technical Report CA-TR-20100105-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[3] Jason Brownlee. A template for standardized algorithm descriptions. Technical Report CA-TR-20100107-1, The Clever Algorithms Project http://www.CleverAlgorithms.com, January 2010.

[4] Carlos Fernando Esponda Darlington. *Negative Representations of Information*. PhD thesis, The University of New Mexico, 2005.

[5] P. D'haeseleer. An immunological approach to change detection: theoretical results. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 18–26. IEEE Computer Society, 1996.

[6] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis and implications. *IEEE Symposium on Security and Privacy*, pages 110–119, 1996.

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society, 1996.

[8] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 202–212. IEEE Computer Society Press, 1994.

[9] S. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, volume 2, pages 1289–1296. Morgan-Kaufmann, 1999.

[10] S. A. Hofmeyr. *An Immunological Model of Distributed Detection and its Application to Computer Security*. PhD thesis, Department of Computer Sciences, University of New Mexico, 1999.

[11] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[12] Zhou Ji and Dipankar Dasgupta. Revisiting negative selection algorithms. *Evolutionary Computation*, 15(2):223–251, 2007.

[13] T. Stibor. *On the Appropriateness of Negative Selection for Anomaly Detection and Network Intrusion Detection*. PhD thesis, Darmstadt University of Technology, Germany, 2006.