Using Genetic Programming for Document Classification

Børge Svingen
Department of Computer Systems
Norwegian University of Science and Technology (NTNU)
N-7034 Trondheim
Norway
bsvingen@idi.ntnu.no

Abstract

Genetic programming is successfully used to evolve agents capable of classifying textual documents according to the interests of the user. The system uses a preclassified set of documents to train the agents, and the results are then tested on an alternative set of documents.

Introduction

With the large amounts of information easily available today, a major problem is to find the parts that are interesting. This paper describes an experiment that attempts to create an agent that takes a textual document and decides whether it is of interest to the user of the system — more specifically, it attempts to do this by using genetic programming (Koza 1992; 1994; Altenberg 1994; Angeline & Kinnear, Jr. 1996).

In the next section, the document classification process is described in detail, before the actual experiment is presented. Subsequently, a specification of how genetic programming is used to evolve the document classification agents is given. The results of the experiment are presented, and finally a conclusion is drawn.

Document Classification

When a group of documents is presented to a user, the documents will be of varying interest. On the basic level, some documents are considered interesting enough to be read, while others are not. Although this classification is situation dependent, it should be possible to make some general claims about which documents are interesting and which are not. The set of documents could be divided into several classes depending on the degree of interest the user has in them, maybe even a continuous, numerical measure of interest could be given, but this would complicate matters; dividing the set of documents into two classes will usually be sufficient, and maybe even preferable, for most users.

The actual content of a document is the meaning assigned to it by the user. This is obviously difficult to capture by a computer program. On a lower level, it is possible to analyze the grammatical structure of the document and the meaning of the individual phrases. Although theoretically possible, this method has proved difficult in practical applications (Russell & Norvig 1995).

The approach taken here is the one traditionally used in information retrieval and information filtering: each document is seen as a set of words, with no mutual relations between the words. Furthermore, no meaning is assigned to the words. This means that the only information about a document that can be used to classify the document as interesting or not interesting is the set of words present in the document, and, conversely, the set of words not present in the document. This is a rather simple view of the content of a document, but a great deal of information is still present, often enough to make the correct prediction.

The aim of the experiment described in this paper is therefore to show that genetic programming can be used to evolve agents that, based on the set of words present in a document, decide whether the document is of interest to a specific user.

The Experiment

In the following, an experiment is described that attempts to show that genetic programming can be used to evolve document classification agents.

A total of 617 example documents are used. These documents are the messages posted to the genetic programming mailing list from January 2 through June 14, 1993. They have all been manually classified as being interesting or uninteresting; documents regarding different selection methods, for instance fitness proportionate selection, tournament selection, or the use of demes (Koza 1992; Wright 1932; Tanese 1989; Andre & Koza 1995; 1996; Niwa & Iba 1996), have been classified as interesting, and all other documents have been classified as uninteresting. Of the total 617 documents, 101 are classified as interesting, and the other 516 documents are classified as uninteresting.

This group of documents is then arbitrarily divided in two; group A, with 62 interesting and 223 uninteresting documents, is used as training data for the genetic programming algorithm. Group B, with 39 interesting and

293 uninteresting documents, is then used, after the genetic programming algorithm has produced a result, to test how good the evolved programs are at classifying documents they have not been trained on.

The experiment therefore consists of using genetic programming to evolve a document classification agent using document group A to calculate the fitness values of the evolved programs, before the resulting agents are tested on document group B.

Genetic Programming Specification

Based on the situation described so far, the function and terminal sets used to do genetic programming are presented in Tables 1 and 2, respectively. As can be seen from the tables, these are the normal boolean and arithmetic functions and constants — the boolean functions treat 0 as FALSE, and all other values as TRUE. In addition, there is a function PRESENT, which checks to see if a certain word is present in the current document — this is the only way the agent programs can get information about the documents.

The agent is equipped with a dictionary of words; this dictionary contains all the words used in any of the documents processed by the agent, and assigns to each of these words an integer number — this number is subsequently used to refer to that word. The function PRESENT takes such a number, and checks the current document to see if the word represented by the given number is present. PRESENT returns this number if the word turns out to be present in the document, and returns zero otherwise, representing boolean TRUE and FALSE, respectively. An arbitrary maximum number of words to be processed by the system is set to 10000 in this experiment; integer numbers are therefore all modulo 10000.

The functions AND and OR are implemented as the minimum and the maximum functions, respectively. This is done to avoid having all numbers calculated in lower parts of the program trees replaced by 1's as they progress through these functions on their way up the program tree. This is also why the PRESENT function returns its argument instead of just 1.

Every agent program is allowed to have multiple result producing branches, and each such result producing branch returns a boolean value. The decision of the agent is then taken to be a "majority vote" from these result producing branches: if more than half of the result producing branches return a value of TRUE for a given document, the document is classified as interesting, otherwise it is classified as uninteresting.

The values used for the genetic programming parameters are shown in Table 3. As can be seen from the table, the deme approach is used (Wright 1932; Tanese 1989; Andre & Koza 1995; 1996; Niwa & Iba 1996), along with automatically defined functions and multiple result producing branches — details can be found in (Svingen 1996). Mutation is implemented as duplication or deletion of branches in the trees. A total of 5 runs are completed.

Function	Arity	Explanation
AND	2	Performs the boolean operation AND on its two arguments.
OR	2	Performs the boolean operation OR on its two arguments.
NOT	1	Performs the boolean operation NOT on its argument.
IF	3	The first argument is evaluated. If it evaluates to TRUE, then the second argument is evaluated and returned, otherwise the third argument is evaluated and returned.
+	2	Returns the sum of the arguments.
−	2	Returns the first argument subtracted the second argument.
*	2	Returns the product of the two arguments.
/	2	Returns the first argument divided by the second argument, or zero if the second argument is zero.
−	1	Returns the negation of the first argument.
%	2	Returns the first argument modulo the second argument, or zero if the second argument is zero.
PRESENT	1	Interprets the number given by its argument as a word, and returns that number if the word is present in the document, and returns 0 otherwise.

Table 1: Function Set

Terminal	Explanation
TRUE	Returns 1.
FALSE	Returns 0.
0 — 9999	Ephemeral, returns a random value between 0 and 9999.

Table 2: Terminal Set

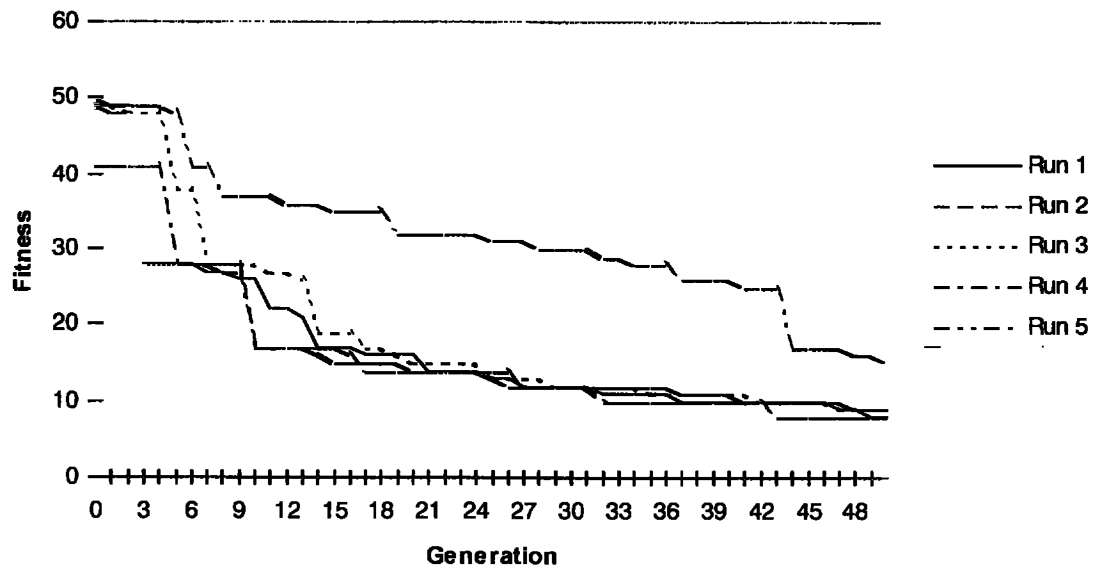

Figure 1: Fitness of the Best Programs

Parameter	Value
Number of demes	25
Population size per deme	1000
Maximum number of ADFs	4
Maximum number of arguments	2
Maximum number of RPBs	4
Crossover rate	90%
Mutation rate	1%

Table 3: Parameter Values

Results

The evolution of the programs in the different runs is shown in Figure 1. The fitness value used in the figure is the number of documents that the evaluated programs missed to classify correctly; this means that the best possible value is 0, for a perfect program, and that the worst possible value is 285, since there are 285 documents to classify; however, since it is easy to find a program that returns a constant value, the worst program that is likely to appear will have a fitness of 62, since there are 62 documents that are classified as interesting and 223 documents that are classified as uninteresting.

Out of the 5 runs, the fitness of the best program in the initial generation in 4 of them is just below 50, while in the other run it is just above 40. Given that every generation consists of 25 demes with 1000 programs in each, and that 5 runs have been performed, a total of $25 \cdot 1000 \cdot 5 = 125000$ random programs have been created. Since none of those have fitness values better than 40,

ADF3	(IF (PRESENT tournament) 1 P0)
RPB0	(ADF3 0)
RPB1	0
RPB2	1

Table 4: Best Program in Generation 3 in Run 1

the probability of such a solution appearing by random search seems to be very small.

Apart from run 5, the runs behave similarly; the initial programs have fitness values close to 50, except for run 4, as mentioned in the previous point. A jump is then made, caused by the evolvement of some important feature, down to 28. A few generations passes without much improvement, before another jump down to somewhere between 15 and 20. From there on, the evolvement takes place more gradually, ending with fitness values of just below 10. Run 5, on the other hand, lacks the two jumps and consists mainly of gradual improvement until generation 43, where a jump is made from just below 30 to just below 20. This run ends up at a fitness of about 15.

In order to gain more insight into the success of the genetic programming process, one of the more typical runs, run 1, will now be examined in detail.

The best program in the initial generation 0 has a fitness of 49. This program is completely random, so no examination of its operation is given. For the next two generations, no improvement is achieved. Then, in generation 3, a program with a fitness value of 28 appears.

Percentage of documents correctly classified	92
Percentage of interesting documents correctly classified	36
Percentage of uninteresting documents correctly classified	99

Table 5: Correctness of Best Program in Generation 3 in Run 1

In order to explain this jump, a simplified version of the best program from this generation is shown in Table 4. The numbers are here replaced by the words they represent. Automatically defined functions that are not used are not shown in the table.

A very good explanation of the jump in the fitness with generation 3 can now be given; the word "tournament", which is used in the context of tournament selection, one of the selection methods, and therefore is present in many of the documents that are marked as interesting, is included in the program. Since RPB1 and RPB2 neutralize each other, the result is given by RPB0, which again calls ADF3 with an argument of 0, so the result of the program is given by (IF (PRESENT tournament) 1 0). The agent program therefore accepts documents that contain the word "tournament", and rejects all others.[1]

The correctness of this program on the documents in group B is shown in Table 5. Although 92% correct classifications might seem good, 36% correct classifications for interesting documents cannot be considered satisfiable.

As can be seen from Figure 1, after a fitness value of 28 has been achieved, the fitness values fall gradually until a fitness value of 8 is achieved in the final generation 50. The best program from this generation, which has 3 automatically defined functions and 3 result producing branches, is given in Table 6. Although it is hard to understand the actual operation of the program (see (Svingen 1996) for an attempt), it should be clear from the table that the program checks for the presence of several words that are natural in documents on the topic of selection methods, such as "tournament", "candidate" and "deme". It is also interesting to notice that both singular and plural form of "tournament" and "deme" are used by the program.

The correctness of this program on the documents

[1]The word "tournament" is in fact such a strong characteristic of the documents classified as interesting that the word, represented by the number 1545, also appears in runs 3 and 4 in the first generation after the jump down to a fitness value of 28. In run 2, this number does not appear. The number 8455 does, however, appear, in the best program in generation 1, which is the first with a fitness of 28, as the argument to a unary minus operator. And since all numbers are given modulo 10000, as was described above, $-8455 = 10000 - 8455 = 1545$.

```
ADF1   (IF (OR P0 (PRESENT candidate))
         (IF (+
               (PRESENT tournament)
               (PRESENT demes)
             )
           1
           P0
         )
         (IF (PRESENT tournaments)
           8607
           (IF (PRESENT tournament)
             1
             (PRESENT (- (PRESENT scant) 1))
           )
         )
       )
ADF2   (+ 3980 (NOT P0))
ADF3   (IF (PRESENT tournament)
         1
         (- (ADF1 P0))
       )
RPB0   (IF (ADF2 1 1)
         (-
           (- (PRESENT deme))
           (ADF3 (PRESENT pet))
         )
         (ADF3 0))
RPB1   (IF (PRESENT galapagos)
         5976
         (PRESENT deme)
       )
RPB2   1
```

Table 6: Best Program in Generation 50 in Run 1

Percentage of documents correctly classified	94
Percentage of interesting documents correctly classified	59
Percentage of uninteresting documents correctly classified	99

Table 7: Correctness of Best Program in Generation 50 in Run 1

in group B is shown in Table 7. The behavior is now significantly better, 94% of all the documents, and 59% of the interesting documents, are correctly classified.

Conclusion

In all the 5 runs, solutions with fairly high fitness values were found. The interesting thing, however, is how well these evolved programs perform on documents they have not been trained on, that is, on group B. The percentage of documents that are correctly classified is not very useful in itself, since it depends heavily on the relationship between the number of interesting and the number of uninteresting documents it is more interesting to look at the percentage of interesting documents that are correctly classified, and the percentage of uninteresting documents that are correctly classified.

In Table 7, the results from the best program in run 1 of the experiment were shown. This solution must be considered good, since it is likely to be satisfying for most users to get access to 60% of all interesting documents, while having to read just 1% of the uninteresting documents[2]. As a conclusion, it can therefore be said that genetic programming seems to be a useful method for creating document classification agents. The model used in this text is extremely simple, and large improvement should be possible by using more advanced information filtering techniques.

This experiment is described in greater detail, along with further work, in (Svingen 1996).

Acknowledgments

This work has received support from The Research Council of Norway (Program for Supercomputing) through a grant of computing time.

References

Altenberg, L. 1994. The evolution of evolvability in genetic programming. In Kinnear, Jr., K. E., ed., *Advances in Genetic Programming*. MIT Press. chapter 3, 47 74.

Andre, D., and Koza, J. R. 1995. Parallel genetic programming on a network of transputers. In Rosca, J. P., ed., *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, 111 120.

Andre, D., and Koza, J. R. 1996. Parallel genetic programming: A scalable implementation using the transputer network architecture. In Angeline, P. J., and Kinnear, Jr., K. E., eds., *Advances in Genetic Programming 2*. Cambridge, MA, USA: MIT Press. chapter 16, 317 338.

Angeline, P. J., and Kinnear, Jr., K. E., eds. 1996. *Advances in Genetic Programming 2*. Cambridge, MA, USA: MIT Press.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Natural Selection*. Cambridge, MA, USA: MIT Press.

Koza, J. R. 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press.

Niwa, T., and Iba, H. 1996. Distributed genetic programming: Empirical study and analysis. In Koza, J. R.; Goldberg, D. E.; Fogel, D. B.; and Riolo, R. L., eds., *Genetic Programming 1996: Proceedings of the First Annual Conference*, 339–344. Stanford University, CA, USA: MIT Press.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence, A Modern Approach*. Prentice Hall.

Svingen, B. 1996. Evolving autonomous agents using genetic programming. Master's thesis, Norwegian University of Science and Technology (NTNU).

Tanese, R. 1989. Distributed genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*.

Wright, S. 1932. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth International Congress of Genetics*.

[2]If this is not the case, it can be controlled by adjusting the fitness function; if the fitness function puts more weight on incorrectly classified interesting documents than on incorrectly classified uninteresting documents, then more documents will be classified as interesting, and less information will be lost at the expense of having to read a higher number of documents.