Project Report

Grammatical Evolution for the classification of Alzheimer's disease patients

Ferrara Luigina 0622900144
Gargiulo Anna 0622900139

Sommario

Aim of the Experiment

The aim of the experiment is to perform a binary classification on the Darwin dataset for the Alzheimer with explainable artificial intelligence in order to identify patients and healthy people. The idea is to choose an evolutionary algorithm to perform a classification task.

Dataset

The dataset chosen for the experiment is the publicly available Darwin dataset for the Alzheimer. The Darwin (Diagnosis AlzheimeR WIth haNdwriting) dataset includes data from 174 participants taken from handwriting samples from both patients and healthy controls. 89 participants are classified as Patients and 85 as Healthy. Therefore, the dataset is quite balanced. It is composed of 451 features represented by numerical or categorical values. The patients belong to the class labelled with 0 while the heathy to the class labelled with 1. Handwriting data were collected according to the acquisition protocol composed of 25 handwriting tasks.

For the experiments, the dataset has been divided into training and test sets. More specifically, the 80% of the available data has been adopted for the training phase while the remaining 20% for the testing phase.

Preprocessing

Before the split, the data has been preprocessed by applying the MinMaxScaler transformation. It scales and translates each feature individually such that it is in range (0,1).

Grammatical Evolution

Grammatical Evolution is an evolutionary algorithm that can evolve programs, similarly to Genetic Programming. Each individual is a variable – length linear structure that contains in its genome the information to select production rules from a grammar. Typically, each codon is an integer. The grammar is exploited in the process of mapping the genotype to the phenotype. To do so, it contains domain knowledge about the problem, in order to define a rule that makes sense in the scenario.

The goal of the algorithm is to minimize a cost function.

The implementation used for the algorithm is the PonyGE2 available on Github[1].

Hyperparameters

This implementation is comprised of many hyperparameters that can be set in the *src/algorithm/parameters.py* file. More specifically, the ones of interest for this application are:

[1] https://github.com/PonyGE/PonyGE2.

Default step and search loop functions

- *SEARCH_LOOP, this parameter defines the way the evolution is carried out throughout the generations. It set to the standard search loop for an evolutionary algorithm, looping over a predefined number of generations.*
- *STEP, defines the process of evolution for a single generation. The one chosen is the classical evolution step.*

Evolutionary Parameters

- *POPULATION_SIZE, the number of individuals of a population.*
- *GENERATIONS, number of generations.*

Class of problem

- *FITNESS_FUNCTION, this parameter carries out the evaluation of the individuals. The one chosen is that for a classification task which requires as error metric the F1 score.*

Dataset

- *DATASET_TRAIN, specifies the path for the data to use to train the model.*
- *DATASET_TEST, specifies the path for the data to use to test the model.*

Grammar

- *GRAMMAR_FILE, specifies the path of the grammar to use to evolve the population.*

Error metric

- *ERROR_METRIC, bound to the fitness function, the error metric is set to F1 score.*

Max sizes of individuals

- *CODON_SIZE, the maximum value a codon can take. It is set to 100000.*
- *MAX_GENOME_LENGTH, maximum number of codons in a genome. It is set to None.*
- *MAX_WRAPS, the maximum number of times the genome is cycled over to obtain a complete grammar. It is set to 0 as default.*

Initialization

- *INITIALISATION, the technique to use to initialize the population. It is set to Ramped half and half, which merges the Full method (trees are grown until a prespecified depth is reached on all branches) and the Grow method (). In the Full method, , at which point only terminals are selected to complete the tree (trees are grown randomly until a branch reaches the maximum defined depth; at this point, only terminals are selected in order to ensure that the max depth limit is not breached).*

- *INIT_GENOME_LENGTH, the initial size of the genome. It is set to 200.*
- *MAX_INIT_TREE_DEPTH, the maximum tree depth for the initialization. It is set to 10.*
- *MIN_INIT_TREE_DEPTH, the minimum tree depth for the initialization. It is set to None.*

Selection

- *SELECTION, the technique used to select the fittest individuals in the population, that are passed on to the next generation. It is set to tournament, where k individuals are selected and tournament is run among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation.*
- *TOURNAMENT SIZE, the number of individuals to select at each tournament.*
- *INVALID_SELECTION, defines the possibility of selecting invalid individuals during selection. It is set to False.*

Crossover

- *CROSSOVER, the technique used to cross over two individuals in the population, then passed on to the next generation.*
- *CROSSOVER_PROBABILITY, the probability of crossover occurring.*

Mutation

- *MUTATION, the technique used to mutate the individuals in the population, then passed on to the next generation.*
- *MUTATION_PROBABILITY, the probability of mutation occurring.*
- *WITHIN_USED, to confine mutations only within the used portions of the genome. This parameter is set to True.*

Replacement

- *REPLACEMENT, the technique used to select which parents and children survive into the next generation. It is set to generational which replaces the entire parent population with the new one at each generation.*
- *ELITE_SIZE, the size of the best individuals in the parent population copied unchanged in the next generation. It is set to the 1% of the population size.*

BNF Grammar

The context-free grammar chosen respects the if-else classifier paradigm. It is composed of seven production rules. Each production rule is composed of non-terminal symbol (enclosed by angle brackets), followed by the ”goes-to” symbol ::=, then, a list of production choices separated by the | symbol. Production choices can be composed of any combination of terminals or non-terminals. The start symbol is $<$ cf $>$, a condition defined by the choice taken for the other rules. The second and the sixth rules are recursive, or rather they contain themselves as a production choice. The grammar was defined as so:

```
<cf> ::= np.where(<cond>, 0, 1) | <os>
<cond> ::= (<var> <relop> <var>) | (<cond> <lop> (<var> <relop> <var>))
<relop> ::= > | < | >= | <=
<lop> ::= & | "|"
<os> ::= 0 | 1
<var> ::= x[:, <varidx>]  | (<var>+<var>) | (<var>-<var>) | (<var>*<var>) | pdiv(<var>, <var>) | psqrt(<var>) | plog(<var>)
<varidx> ::= GE_RANGE:dataset_n_vars
```

During the preliminary tests, we have also tried to add constants as terminal symbols in the grammar. Unfortunately, the performance for the test fitness got worse. Therefore, constants have been removed characterizing the grammar on the direct comparison between either simple features or combination of them.

Testing phase

The testing phase has started with the execution of several preliminary tests in order to understand the set of parameters on which the experiment focuses. In the tests, we have analyzed the importance given by each of the parameters in the algorithm to choose which ones to remain fixed and which to change.

We have decided to set the generation size to 150 because in most trials the training fitness showed an improvement up to the 100^{th} generation in average. We have considered extra 50 generations to give the algorithm enough time to stabilize. The figures are examples of the best training fitness in two runs.

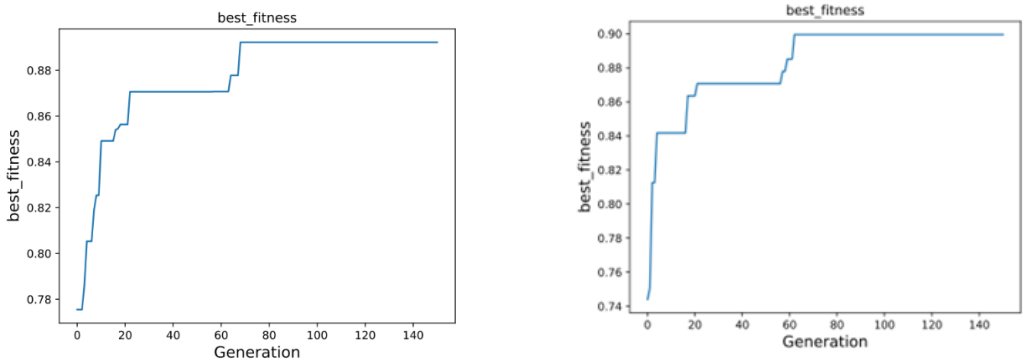

Figure 1: Examples of runs to evaluate the number of generations needed.

For the population size, we have decided to set it to 500. We started with larger values (700, 1000), but the performance were not improved by a bigger number of individuals. For this reason, we have chosen 500, which gave better performance for the fitness and also made tests faster.

At the end of the preliminary tests, we have focused on the operations which make up the evolutionary process, the selection, the crossover and the mutation. More specifically, we have worked by tuning with grid search the tournament size, the mutation probability and the crossover probability.

For both mutation and crossover, we tested different approaches:

- Mutation per codon and mutation per individual;
- One point crossover and two point crossover.

For the tests, 30 different seeds have been adopted: 101304, 432071, 752782, 659486, 929644, 412962, 205916, 202578, 785371, 215023, 523700, 548219, 864499, 129638, 976786, 129442, 976984, 327105, 131493, 835671, 178753, 390797, 346876, 814265, 250751, 820127, 253826, 402281, 342547, 299942. They have been generated by the seed 12347.

For each test, the tables display only the most relevant results.[2]

Test 1

This test evaluated different combinations for mutation (the values of which are set to 0.1, 0.3, 0.5, 0.7, 0.9), crossover (0.2, 0.4, 0.6, 0.8) and tournament (2, 10, 20, 50, 100).

Mutation	**Crossover**	**Tournament**	***Avg. Train Fitness (%)***	***Avg. Test Fitness (%)***	***Standard Deviation (%)***
0.1	0.4	10	89.9	69.0	8.5
0.1	0.4	50	90.4	69.2	5.2
0.1	0.6	20	90.4	69.9	8.4
0.1	0.6	100	90.3	69.4	8.2
0.1	**0.8**	**20**	**90.5**	**70.2**	**6.3**

Test 2

From the previous test, the overall conclusion is that a high value for mutation is not significative to get a better solution for the problem. Overall, the highest performance was achieved for lower values of the probability of mutation (0.1), whereas the crossover and tournament seem to be more relevant in this context. Therefore, we evaluated all different possible values for the cross over (0.1, 0.2, 0.3,

[2] The complete outcomes are available in the *outcomes.xlsx* file for test 1 and test 2.

0.4, 0.5, 0.6, 0.7, 0.8, 0.9) in combination with different values of tournament (2, 10,20, 50, 100) with mutation set to None, which implies a value of 1/GENOME_LENGTH for each codon.

Crossover	Tournament	Avg. Training Fitness (%)	Avg- Test Fitness (%)	Standard Deviation (%)
0.1	50	89.4	69.7	6.8
0.1	100	90.4	69.3	6.7
0.2	50	90.7	69.7	6.6
0.3	2	89.6	70.1	7.7
0.3	50	91.1	70.0	8.2
0.8	**2**	**88.7**	**70.3**	**6.2**
0.8	20	91.9	70.4	6.7

Overall, the best configuration to obtain the highest performance is the mutation = None; crossover = 0.8; tournament = 2, which gives the best performance with the lowest standard deviation.

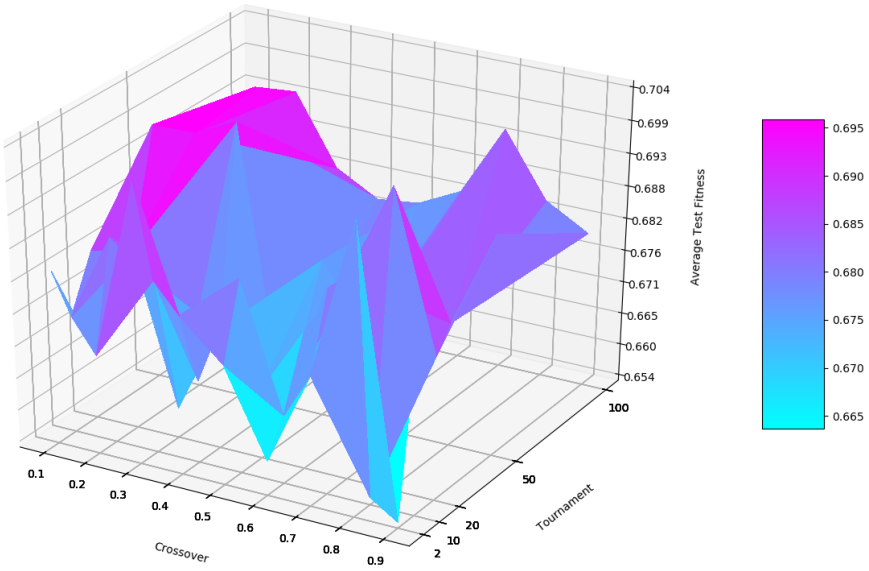

Figure 2: Representative graph of Test 2, with mutation implicitly represented as None.

Test 3

The best configuration from the previous experiment (n.2) was used to test whether the one point crossover was the best choice or not. So, this experiment tests the two point crossover modality instead of the one point. The results were:

Crossover	Tournament	Avg. Training Fitness (%)	Avg- Test Fitness (%)	Standard Deviation (%)
0.8	2	86.8	64.8	6.8

Test 4

The best configuration from experiment 2 was used to test the performance of the algorithm when the mutation probability was associated with the individual, rather than with the codons. The results were:

Crossover	Tournament	Avg. Training Fitness (%)	Avg. Test Fitness (%)	Standard Deviation (%)
0.8	2	89.5	65.8	6.9

Test 5

The best configuration from experiment 2 was used to test the performance of the algorithm when the mutation probability was associated with the individual and two point crossover was used. The results were:

Crossover	Tournament	Avg. Training Fitness (%)	Avg. Test Fitness (%)	Standard Deviation (%)
0.8	2	87.8	67.1	7

Conclusions

Performing Grid – Search – like tests, the overall conclusion is that the Grammatical Evolution algorithm is not fit to perfectly classify the samples in the dataset, but rather, changing the hyperparameters of the algorithm has no significative effect on the overall classification accuracy.

Nevertheless, the best performance was achieved with the following configuration:

- Crossover: cross one point, with probability 0.8
- Mutation: mutation per codon, with probability 1 / genome_length
- Tournament size: 2
- 150 generations

Which displays the lowest overfitting, with the highest test accuracy but lowest standard deviation.

The second best result is:

- Crossover: cross one point, with probability 0.8
- Mutation: mutation per codon, with probability 0.1
- Tournament size: 20
- 150 generations.

Although the first result achieves better performances, when looking at the best individual produced by that configuration, the characteristics were underwhelming, since it had a very long phenotype and genotype, with no significative improvement on the performance. Therefore, we preferred the best individual produced by the second best result, which was characterized by a shorter phenotype and genotype, but the rule chosen is more significative in the application (with performances on test equal to 83% compared to 80% of the previous individual).

Summing up, the best individual achieved with these tests is:

```
np.where((x[:, 305] >= ((x[:, 190]* (x[:, 79]*
(x[:, 398]-plog(x[:, 437])))) + (x[:, 116]*
x[:, 421]))), 0, 1)
```

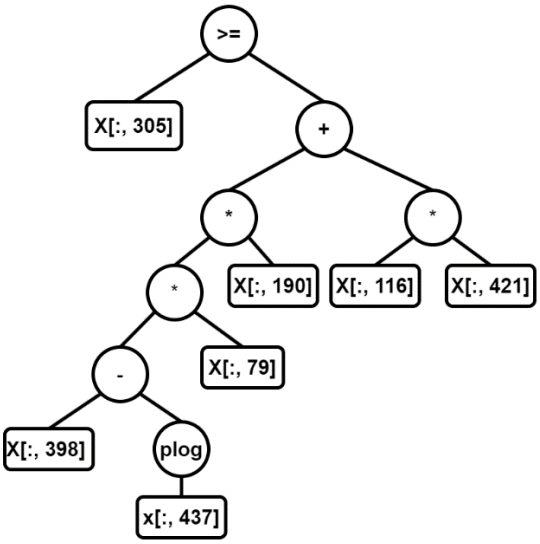

Figure 3: Tree representation of the decision rule

with genotype

[67044, 86980, 24724, 14255, 510, 51157, 97534, 41846, 29440, 9880, 58058, 32929, 29080, 97258, 80498, 43238, 62608, 87287, 41548, 89747, 31616, 85106, 95371, 80498, 43238, 62608, 40701, 41548, 89747, 31616, 85106, 95371, 89695, 9880, 58058, 85537, 53454, 46151, 80498, 55963, 62608, 57220, 55328, 89747, 31616, 17276, 62337, 62608, 57220, 41548, 85106, 95371, 17276, 62337, 43297, 14091, 26099, 61523, 62654, 1994, 43370, 46371, 81761, 35350, 35377, 45358]

and Training Fitness 90.6%; Test Fitness 83.5%.

This is the confusion matrix associated to the phenotype:

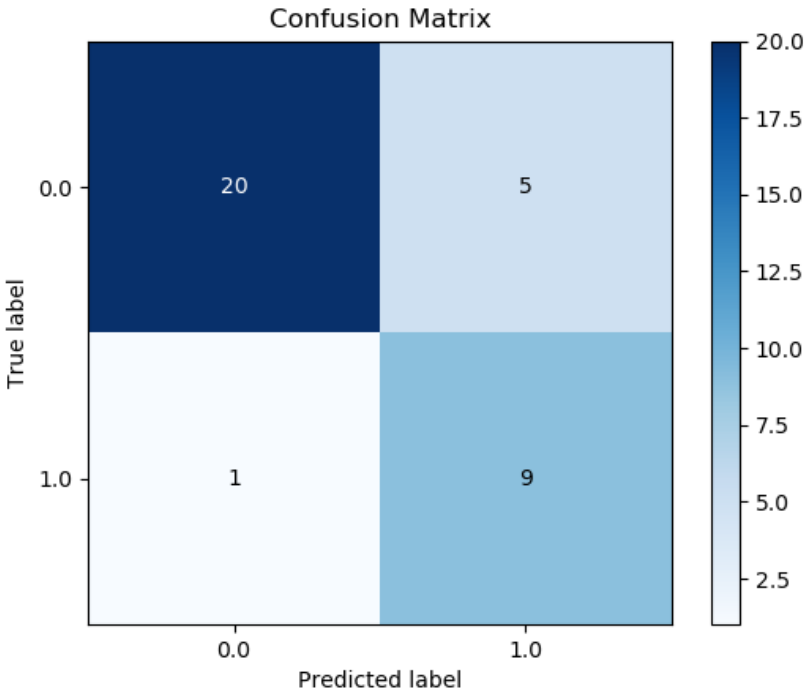

Figure 4: Confusion Matrix

One last analysis can be made regarding the choice of the grammar. As previously stated, when the features of the dataset were compared to constants, the results of the algorithm were inadequate. This brings us to believe that, in this context, to safely diagnose healthy people from Alzheimer's patients, the relationship between features, and the combination of values among them is more important than verifying if a specific feature has a value above or under a certain threshold.