UNIVERSITÀ
DI PARMA

Implementation of Sub Machine-Code Genetic Programming for Digit Recognition using the DEAP Python Library

Andrea Bettati
Gianmarco Carraglia

Outline

- Project Objective
- Genetic Programming
- Digit Recognition
- Python DEAP Library
- Evolutionary Algorithm: eaSimple
- Developed Software
- Conclusions and future work

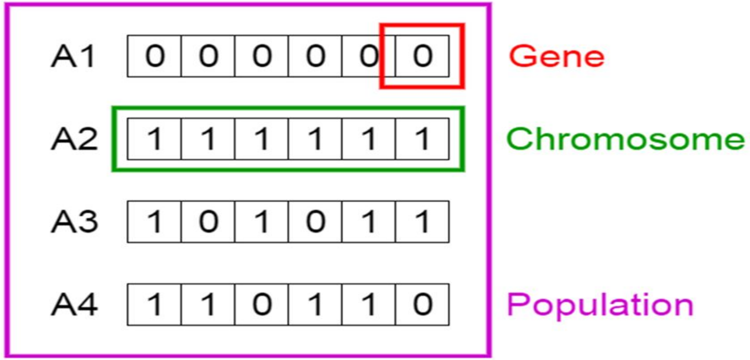
Project Objective

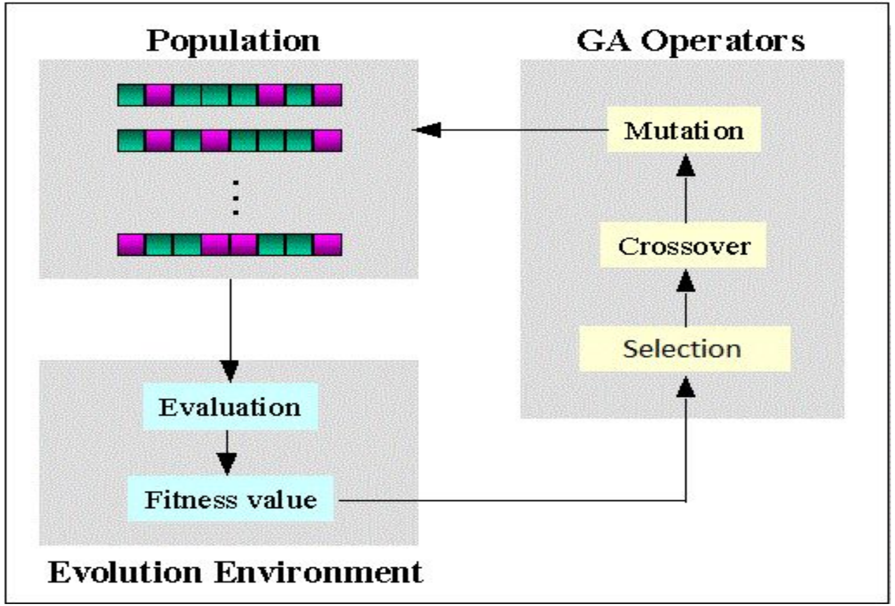
Main goal: train a set of binary classifiers for digit recognition using DEAP and Smc technique (i.e. bit-wise operators).
Our project directly relates with the studies of Prof. Cagnoni (described in [cagnoni2005]): it is a direct extension of his paper.

[cagnoni2005]	Our Implementation
10-classes digit classifier	10 binary digit classifiers
developed in C	developed in Python
Smc Technique	Smc Technique
Binary tree encoding	Binary tree encoding
Population variation strategy: VarOr	Population variation strategy: VarAnd
32 bit operands/operators	64 bit operands/operators

Genetic Programming

- **Biological evolution-emulating** approach to machine learning
- Random generation of a **population** of N individuals or **chromosomes** composed by **genes**
- **Fitness** evaluation
- **Evolution** of population





Genetic Programming - Fitness Evaluation

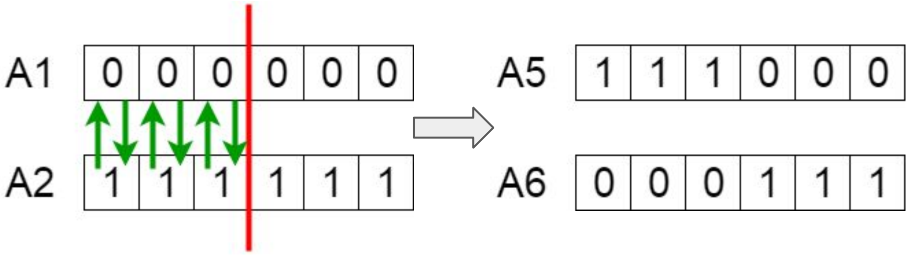
- Assigning a fitness value to every individual based on its ability to compete with other individuals
- Fitness score evaluated using entries of the **confusion matrix** and size of the associated **binary tree**:

$$F = 1 - \sqrt{\frac{fp^2 + fn^2}{N_p^2 + N_n^2}} - K_s Size$$

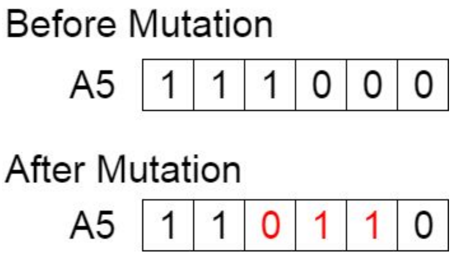
Genetic Programming - Evolution

- Based on the fitness score, an individual can be **passed down** to the next generation(iteration of the algorithm)
- Also, every individual has a probability to experience:
  - **Crossover**
  - **Mutation** (low probability)

Crossover:



Mutation:

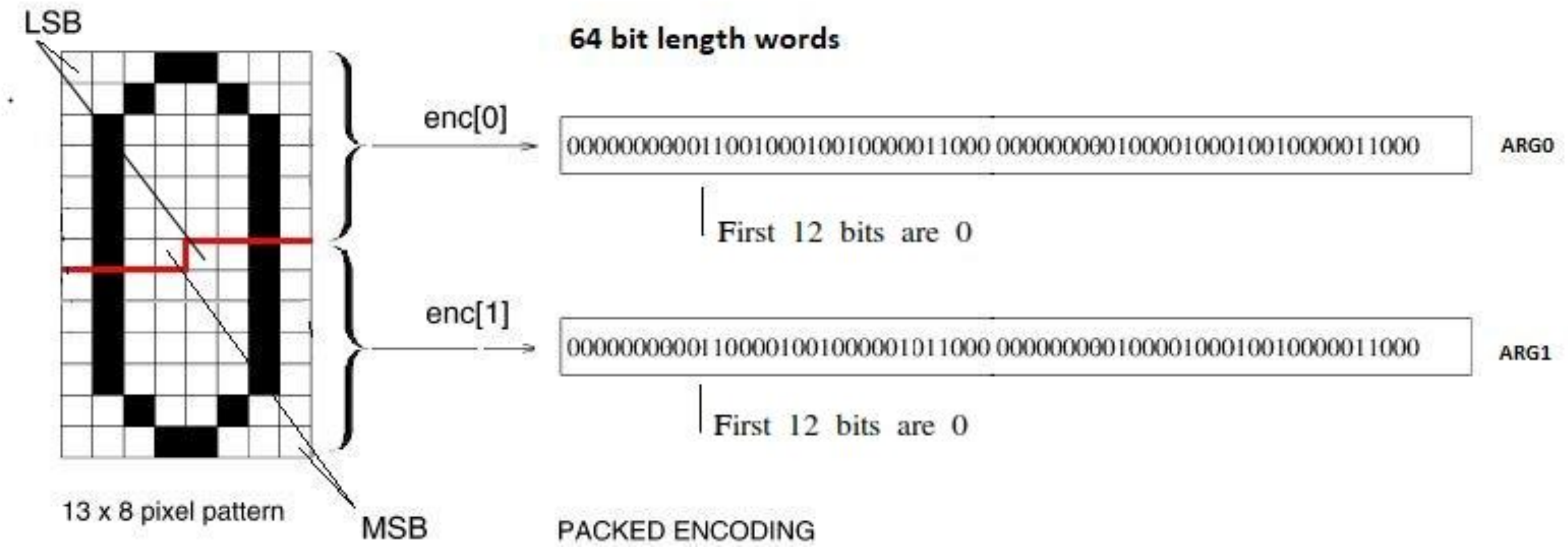


Digit Recognition - Data Set

- 10 classifiers to recognize **0-9 digits**
- With a data set composed of:
    - **Training set** of 6024 entries of digits
    - **Test set** of 5010 entries of handwritten digits
- Both digitalized from a license-plate database

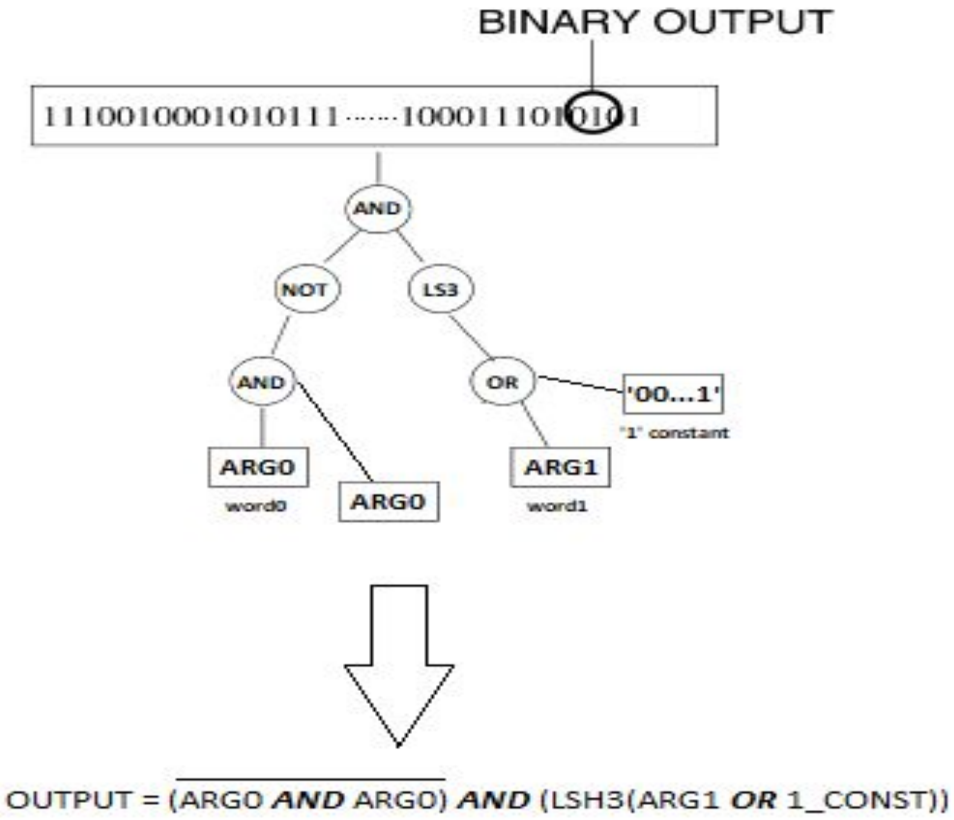
Andrea Bettati
Gianmarco Carraglia

Digit Recognition - Encoding



Digit Recognition - Binary Trees

- Each individuals is a **binary tree** with a 1-1 relation with a **logical function**: each node is a **bitwise** operator and every leaf is an input of the function

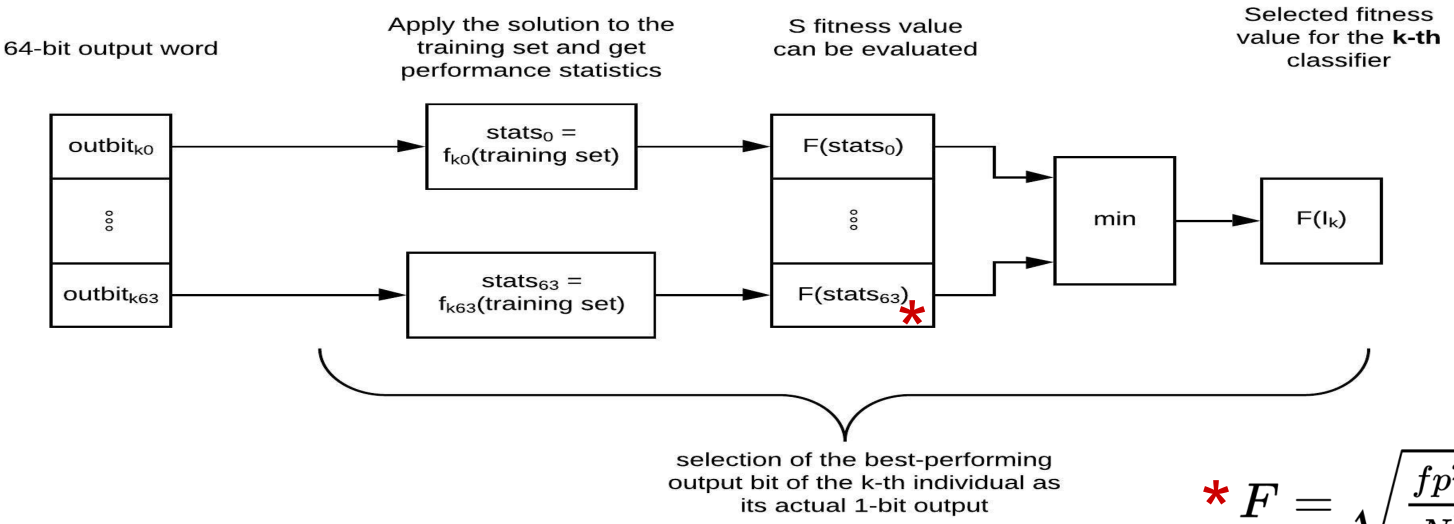


Digit Recognition - Binary Tree operators

- Every **node**(bitwise operator) in a tree can be:
  - AND
  - OR
  - NOT
  - XOR
  - NAND
  - NOR
  - Left/Right bit shift operations
- Every **leaf**(64 bit operand) in a tree can be:
  - ARG0
  - ARG1
  - 000...01 constant
  - 000...00 constant
  - Random 64 bit constant

Digit Recognition - Output

- The output will be a 64 bit word, in which every bit is a possible classifier
- According to the fitness value the best bit will be selected
- The couple *[fitness value, best bit]* will characterize the individual



$$* \; F = \sqrt{\frac{fp^2 + fn^2}{N_p^2 + N_n^2}} + K_s Size$$

Andrea Bettati
Gianmarco Carraglia

Python DEAP Library

- A novel **evolutionary computation framework** written in Python.
- Works with **parallelisation** mechanism such as multiprocessing.
- **Open Source** (code on GitHub).
- Provides **classes** and **methods** to **implement** and **tune** classical GP data structures and algorithms

DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON

UNIVERSITÉ LAVAL

Python DEAP Library

The DEAP library provides **classes** and **methods** to **implement** and **tune** classical GP **data structures** and **algorithms**.

Typical workflow:

1. Define **individual type** and **primitives set**
2. Define a **fitness function**
3. Register type's attributes (e.g. **initialization**)
4. Register and decorate **operators** (e.g. crossover/mate and mutation)

Problem description

5. Call the **evolutionary algorithm** (e.g. eaSimple)

Training

6. Evaluate best individual

Test

DEAP - Individual Type and Primitives Set

Individual Type: Prefix Tree

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin)
```

Primitives Set:

```
pset = gp.PrimitiveSetTyped("MAIN", [int, int], int)
pset.addPrimitive(operator.__and__, [int, int], int)
pset.addPrimitive(operator.__or__, [int, int], int)
pset.addPrimitive(operator.__invert__, [int], int)
pset.addPrimitive(operator.__xor__, [int, int], int)
pset.addPrimitive(nand, [int, int], int)
pset.addPrimitive(nor, [int, int], int)
pset.addPrimitive(rshift1, [int], int)
pset.addPrimitive(lshift1, [int], int)
pset.addPrimitive(rshift2, [int], int)
pset.addPrimitive(lshift2, [int], int)
pset.addPrimitive(rshift4, [int], int)
pset.addPrimitive(lshift4, [int], int)
```

Constants Leaves:

```
pset.addPrimitive(const0, [], int)
pset.addPrimitive(const1, [], int)
pset.addEphemeralConstant("ERC",
lambda: random.randint(0, 2 ** word_len - 1), int)
```

DEAP - Evolutionary Algorithm (`eaSimple`)

`eaSimple`: “This algorithm reproduces the **simplest evolutionary algorithm** as presented in chapter 7 of *Back, Fogel and Michalewicz, "Evolutionary Computation 1 : Basic Algorithms and Operators", 2000*”

Python pseudocode:

```
evaluate(population)
for g in range(ngen):
    population = select(population, len(population))
    offspring = varAnd(population, toolbox, cxpb, mutpb)
    evaluate(offspring)
    population = offspring
```

DEAP - Evolutionary Algorithm (`eaSimple`)

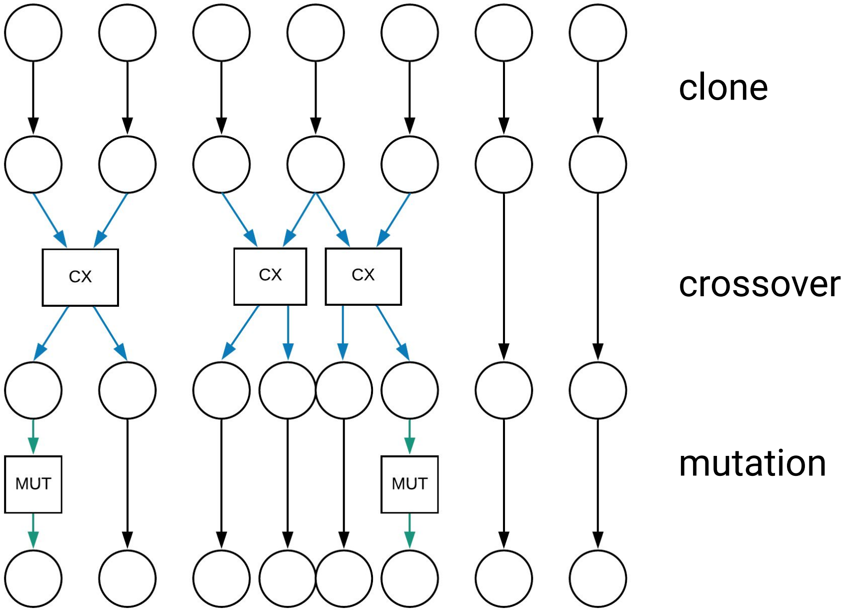
```
population = select(population, len(population))
```

deap.tools.selTournament: "Select the best individual among ***tournsize*** randomly chosen individuals, ***k*** times." (here ***k*** = `len(population)`)

1:1 replacement ratio: the selection procedure is **stochastic** and can select **multiple times the same individual**.

```
offspring = varAnd(population, toolbox, cxpb, mutpb)
```

varAnd function is applied to produce the next generation population.

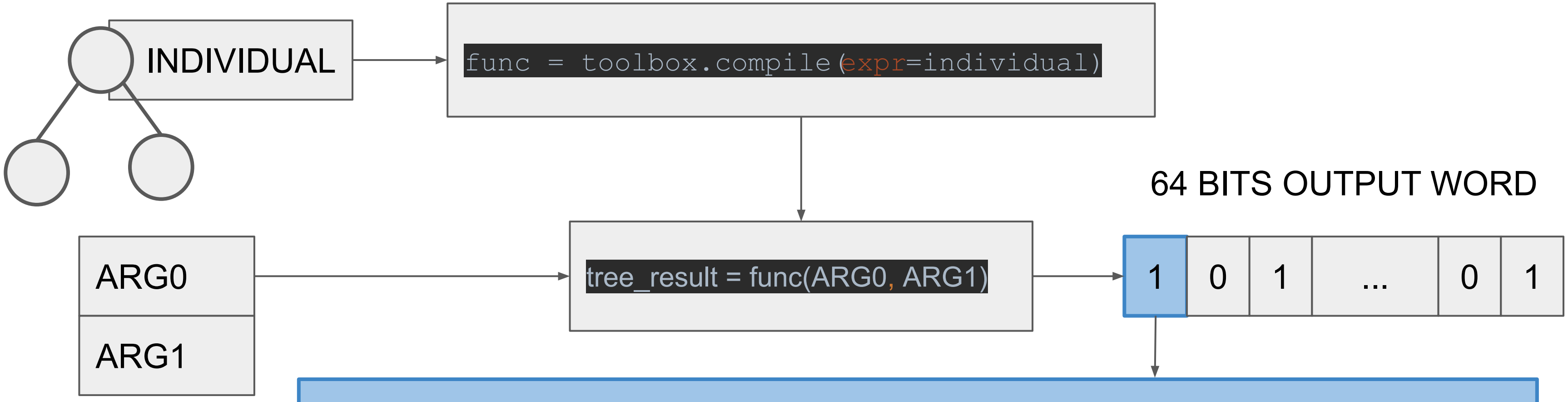


Andrea Bettati
Gianmarco Carraglia

DEAP - Evolutionary Algorithm (`eaSimple`)

`evaluate(offspring)`

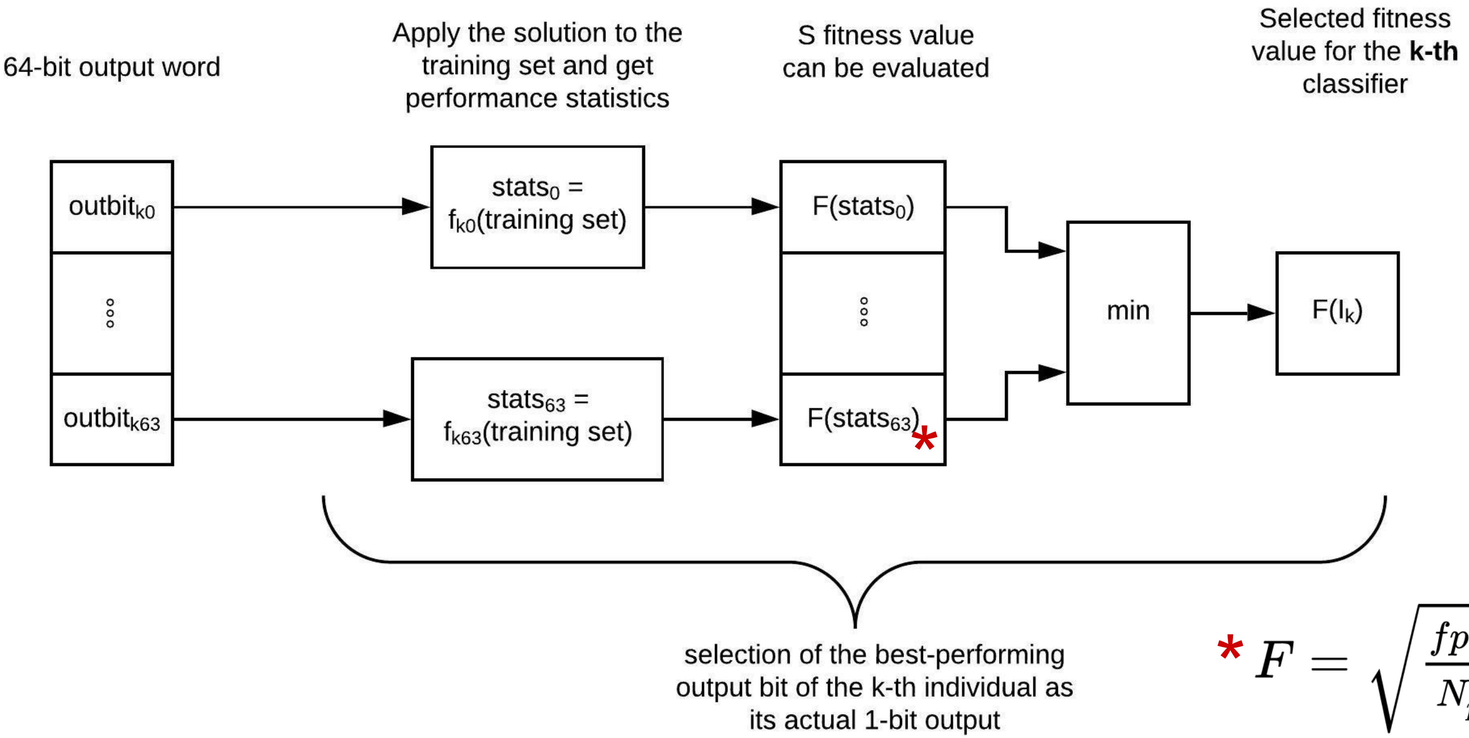
Evaluate the new individuals and compute the statistics on this population: calls **user-defined fitness function**



For every input pair (ARG0, ARG1) (i.e. every image in the training set) **each bit is evaluated as a binary classifier for the selected digit** and a fitness value is obtained: **the lowest is returned as the fitness for the individual**.

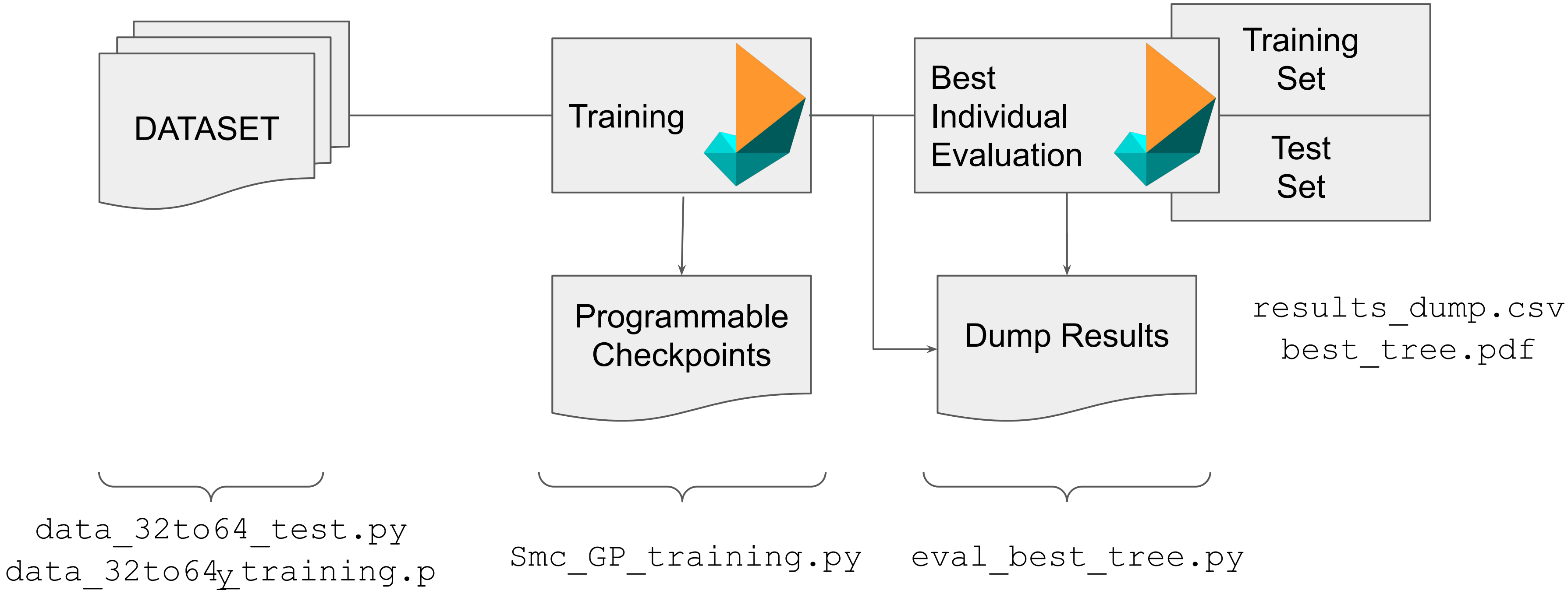
DEAP - Evolutionary Algorithm (`eaSimple`)

user-defined fitness function



$$* \; F = \sqrt{\frac{fp^2 + fn^2}{N_p^2 + N_n^2}} + K_s Size$$

Developed Software



Andrea Bettati
Gianmarco Carraglia

Developed Software - Numerical Results

Upon executing the `Smc_GP_main.py` script, two results are dumped to the memory:

- A textual output (appended to the file `dataset/results_dump.csv`)
- A graphical representation of the best individual (best_`tree.pdf`)

Textual Output

```
TRAINING ENDED 13/02/2020 16:27:57 with PARAMS
----------------------------------------------------------------------------------------------
64  1000    1000    1   7   7   1   6   12  12  4096    4096    1   1   1   1e-06   0   23831 sec
TRAINING:   98.6%   tp  548 tn  5467    fit 0.009516    bit 23
TEST    :   96.2%   tp  482 tn  4508    fit 0.026906    bit 23
```

parameters used during evolution

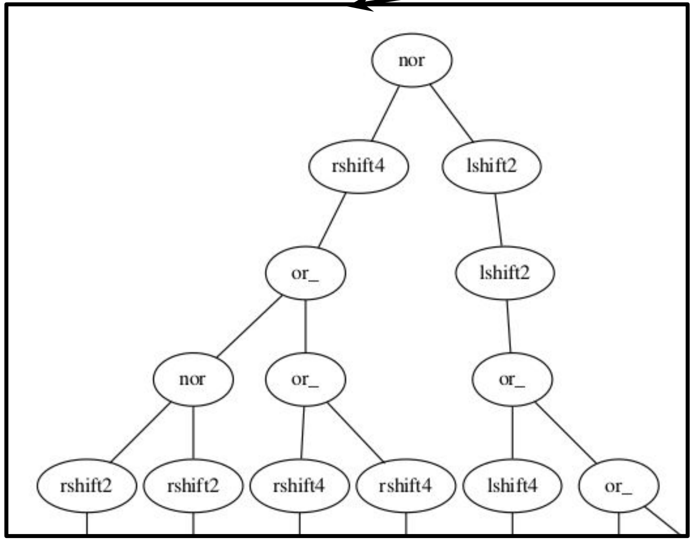
best individual performance over training set

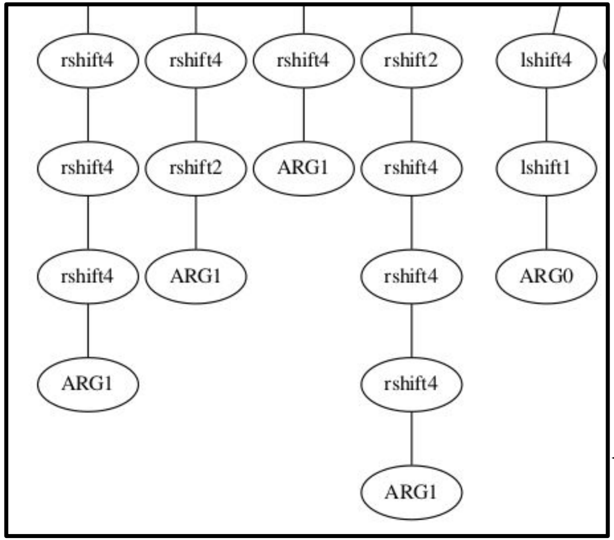
best individual performance over test set

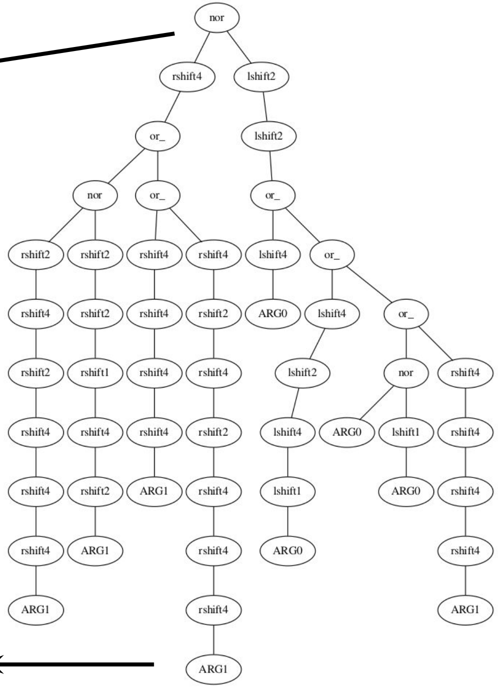
This information is appended to the dump file each time a complete experiment (**training + best individual evaluation**) is executed.

Developed Software - Numerical Results

Graphical representation of the best individual (`best_tree.pdf`)







Conclusions

A set of single digit classifier was built using the DEAP Library, and compared to the results in [cagnoni2005]. Here's a comparison of the used parameters.

Parameter	gen num	pop	1G H	tournsize	mut H	cx H	cxpb	mutpb
[cagnoni2005]	1000	1000	5-7	7	4-6, 12	12	0.8	0.03
Our Implementation	400	1000	5-7	7	4-6, 12	12	0.5	0.1

Conclusions

So far results keep up fairly well with the ones from the previously developed classifiers (data relative to **test set**, taken from [cagnoni2005]).

DIGIT	0	1	2	3	4	5	6	7	8	9
TNR	99.87	99.56	99.69	99.45	99.58	99.42	99.69	99.69	99.47	99.58
TPR	97.21	98.00	95.21	95.81	95.21	93.81	94.41	94.01	93.21	96.01
TNR (DEAP)	99.89	99.6	99.53	99.45	99.76	99.87	98.98	99.69	99.53	
TPR (DEAP)	96.41	95.01	95.81	86.83	95.41	90.02	83.64	93.01	89.82	

Future Work

- Extend classification to 10 classes (e.g. combining the outputs of 10 binary classifiers).
- Explore different implementations of the Evolutionary Algorithm and Fitness Function.
- Make scriptable the file `Smc_GP_main.py`, to try out different parameters configurations within a single run.

Bibliography

- [cagnoni2005] S. Cagnoni, F. Bergenti, M. Mordonini, and G. Adorni, "Evolving Binary Classifiers Through Parallel Computation of Multiple Fitness Cases"
- V. Mallawaarachchi, "Introduction to Genetic Algorithms — Including Example Code"
- DEAP 1.3.1 documentation