MASTER OF SCIENCE
IN ENGINEERING

Haute Ecole Spécialisée
de Suisse occidentale

Project Report

"Genes Selection with Wrapped Classifiers"

Machine Learning on Big Data

Jorge Albaladejo Pomares
Nicolas Jakob

June 2013

Table of Contents

- General context and goal
- Dataset and structures
- Machine Learning methods used
    - Genetic Wrapper
    - Accuracy vs. specificity
    - Fitness value
- Analysis and Tests
    - Test #1
    - Test #2
    - Test #3
    - Test #4
- Futures researches
    - Different classifier
    - Process all available genes
    - Co-occurrence of variables
- Conclusion

General context and goal

The goal of this project is to experiment a wrapper method around a classifier to select genes that are determinant to diagnose leukemia. Since the meaningful genes for this diagnosis are not fully known, the cost of these analysis is high due to the big size of the research space (54K genes). This becomes highly expensive not only in terms of computational resources but also from a practical point of view: a test to detect the illness should consider as few genes as possible, while still providing an acceptable accuracy. We are facing, thus, an NP-complete combinatorial research problem that might not have a “flawless” solution.

General approaches to reduce the number of genes consist of applying classification algorithms on a set of test cases the consider only a subset of the genes. Then with classifier algorithms like C5.0 or fuzzy rules systems it is possible to extract patterns to detect the illness, get a list of meaningful genes for this detection, and consider them as relevant for further analysis. However, there are some limitations with this approach.

First, we assume that the more a variable is used the more it is relevant. This is far from the truth, though, since a gene alone might mean nothing while in collaboration with others may actually become a pattern. Additionally, the fact that we are only using a subset of the variables, this pre-selection is a key: some variables that when tested alone seemed not relevant, in combination with others may become meaningful. This leads to a only one “precise” solution, which implies analyzing the whole space of solutions in search for all combinations of variables. This is just impracticable.

Therefore, our idea is to use a heuristic method based on a genetic algorithm to select randomly a subset of genes from the initial records, then evaluate their result when used with a classifier. With a fitness function based on how well a set of variables identified the illness through a provided set of medical records, the genetic selection will keep the best individuals and cross them in search for better solutions.

Ideally, we would use correlation and co-occurrence analysis of the selected genes according to their meaningfulness to the classificator, as well as statistical frequency, so that actual successful combinations prevail over just random “luck”, and so that the genetic algorithm can converge faster, and to a more accurate solution.

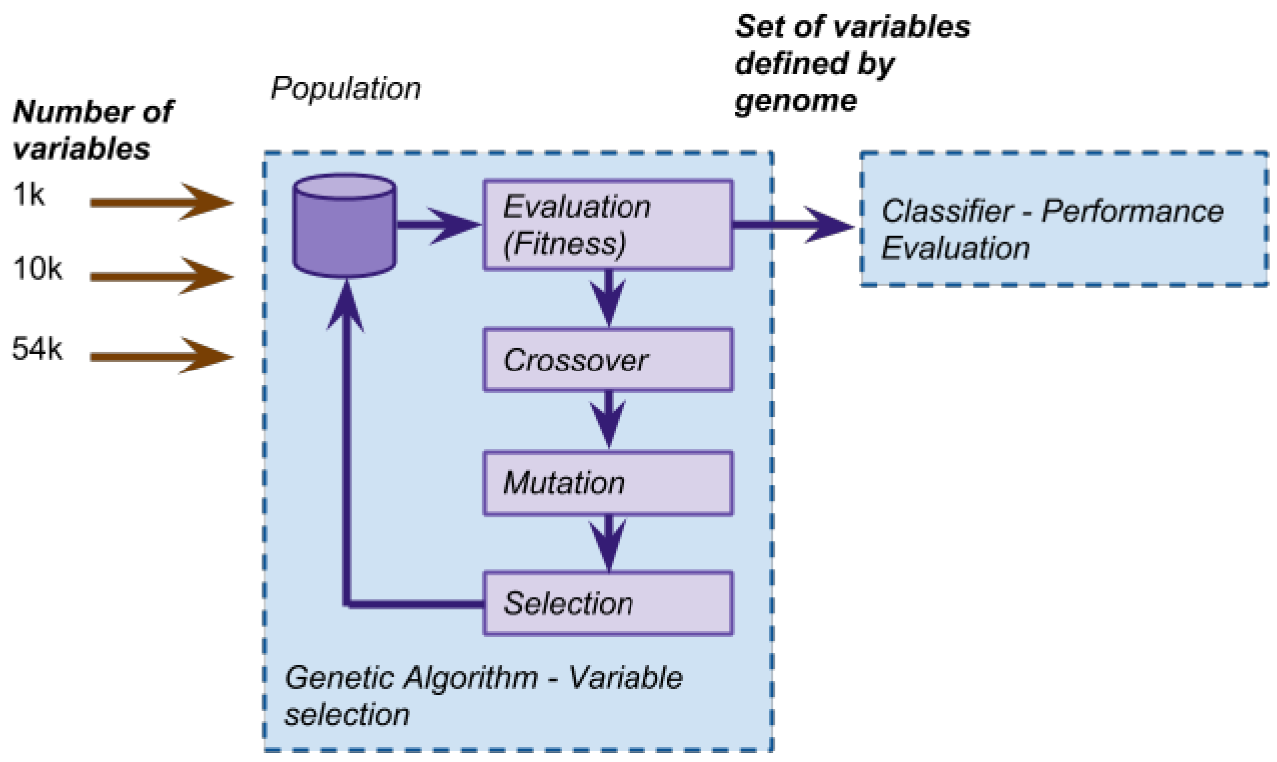
Set of variables defined by genome
Population
Number of variables
1k
10k
54k
Evaluation (Fitness)
Crossover
Mutation
Selection
Classifier - Performance Evaluation
Genetic Algorithm - Variable selection

Figure 1: High-level model of the wrapper-classificator workflow

The sources for this lab can be found at https://github.com/njakob/Leukemia. This repository contains both data, scripts and results of our researches.

Dataset and structures

We have used the dataset provided for the second laboratory[1] of the MLBD course. We have validated our model with the data sets at 1K genes, 5K and 10K. As we will see later, there wasn't much difference between the 5K's and the 10K's results, probably because they come from a previous filtering process on the original raw 54K genes data set.

Once the model validated, it will be interesting to apply it to the original raw data set, task that we couldn't complete due to lack of processing resources in our laptops. The real added value is in that, applying our genetic wrapper to the full dataset, classificators can execute at a reasonable performance, much better than if they had to consider the full set of variables.

Machine Learning methods used

We use a main wrapper based on a genetic algorithm with R and its package "GA"[2]. This one is very useful since it allows customizing the mutation, selection and fitness functions and

[1] http://ape.iict.ch/teaching/MLBD/MLBD_Lab/Lab2_ML_R/
[2] http://cran.r-project.org/web/packages/GA/index.html

therefore it adapts well to our purposes. In the evaluation function of the genetic algorithm, we instantiate and execute a classifier, which will return a result that will be evaluated against the training recordset and computed to a result, or fitness value.

The classifiers used are: C5.0 (although quickly dismissed since it produced internal errors for big data sets, presumably due to a memory leak or related problem), and KNN, which is relatively faster and produced neat results for the aim of calibrating the model.

Other classificators like SVM, or even fuzzy systems can be wrapped into the genetic algorithm. The programmer only needs to launch them from the evaluation function, get their results and compute their score according to the context or other parameters.

The figure 2 show the detailed workflow of our process with which kind of data structured are used between components.

Genetic Wrapper

In the code below we can see the initialization of the genetic library.

```
## Launch genetic algorithm
GA <- ga(
  "binary",
  nBits = ncol(input),
  population = populate, # function !
  fitness = evaluate, # function !
  maxiter = numGenerations, #param
  popSize = populationSize, #param
  pcrossover = 0.8,
  elitism = 0.01,
  pmutation = 0.2,
);
```

The evaluate function will be executed for each candidate chromosome and will contain logic that tests its fitness (normally, invoking a classifier with the genes selected by the chromosome and measuring the accuracy and specificity of its result). The populate function will produce an initial population based on the chromosome rules established.

The chromosomes are *n*-bit vectors, where *n* is the size of the genes collection (1K, 5K, 10K, 54K). Binary chromosomes allow faster selection of the columns to be used within the R semantics, and they are faster computed for crossover, mutation and similarity; besides, binary is the default mode for the GA package. Considering the worst case (54K bits), that makes around 6,5MB in RAM per thousand chromosomes.

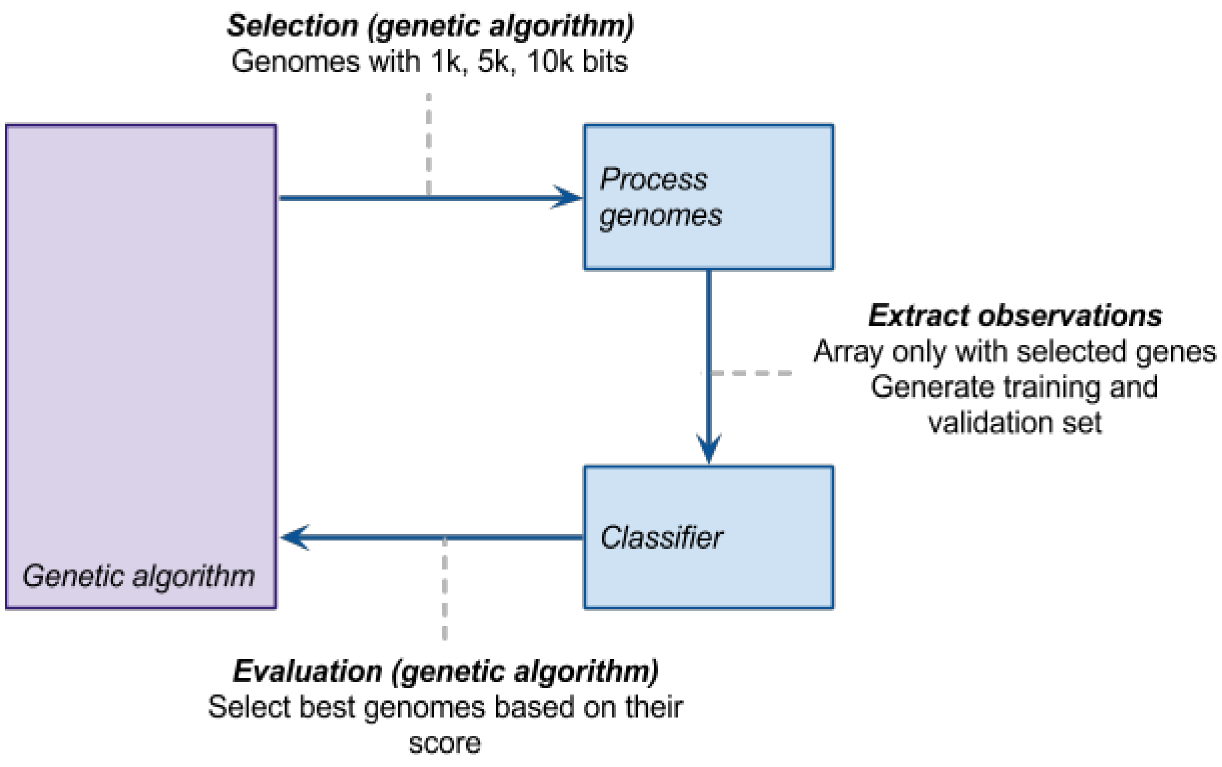
Selection (genetic algorithm)
Genomes with 1k, 5k, 10k bits
Process genomes
Extract observations
Array only with selected genes
Generate training and validation set
Classifier
Genetic algorithm
Evaluation (genetic algorithm)
Select best genomes based on their score

Figure 2: Details of workflow with intermediate data structures

Accuracy vs. specificity

We have implemented a simple evaluation model based on accuracy and specificity. We have "abused" of both terms for simplicity too: in our model, accuracy is how well the result detected the right classes (number of correct classifications / number of total records). Specificity, however, measures how "different" or "unwanted" is the predicted class from the actual one. This choice produces several benefits:

1. Programming and modeling is easier, we don't have to run several times the same process on the same dataset to detect each possible leukemia.
2. The model is more generic: it can be applied to any dataset with any collection of possible classes.
3. Several types of leukemia can be detected with just one test.

As we will see from the results, this modeling choice produces results that are very accurate but not much specific (the system detects a leukemia although not the right type). To avoid "false negatives" such as a "no leukemia" result when there is actually one of the types, we have used the following weighting matrix:

```
scoreByClassesMatrix <- data.frame(
  #c("real/guessed","AML","CML","ALL","CLL","NO"),
  #c(         "AML",  -  ,  -  ,  -  ,  -  ,  - ),
  #c(         "CML",  -  ,  -  ,  -  ,  -  ,  - ),
  #c(         "ALL",  -  ,  -  ,  -  ,  -  ,  - ),
  #c(         "CLL",  -  ,  -  ,  -  ,  -  ,  - ),
  #c(          "NO",  -  ,  -  ,  -  ,  -  ,  - )
  c(1.00, 0.25, 0.90, 0.10, 0.00),
  c(0.90, 1.00, 0.80, 0.70, 0.25),
  c(0.90, 0.10, 1.00, 0.25, 0.00),
  c(0.70, 0.80, 0.90, 1.00, 0.25),
  c(0.50, 0.75, 0.50, 0.75, 1.00)
);
```

Where the meaning of each cell is “how well the system does if, for a given leukemia type (y axis), the right leukemia type (x axis) is detected”. Only four classes (AML, CML, ALL, CLL) plus “no leukemia” are considered, and the weights are set so that certain rules are favorised:

- It is always better if we detect the right class.
- We’d rather detect an acute than a chronic version of the illness (faster medical reaction).
- We’d rather detect a chronic illness than none (slower medical reaction, time for other tests).
- We’d rather detect any illness than none at all (false positives are allowed, even encouraged).
- Not detecting acute illnesses is critical, but not so much with chronic ones (false negatives are forbidden).

Of course, this weights matrix responds to our view of the problem, and the important and useful fact is that it can be parametrized to better suit the researcher’s needs. For instance, health care staff could confirm that it is better having false positives while the administrations might decide they prefer false negatives.

Fitness value

The score or fitness value is given by the following formula:

```
score = lengthScore * weight.chromosomeLength + accuracyScore * weight.accuracy +
specificityScore * weight.specificity;
```

Where each score has a parametrable weight, and three different values are considered:

- **lengthScore** is based on the chromosome length. It is 0.5 for the selected size and increases linearly from one gene to double the selected size (values 1.0 and 0.0

respectively)

- **accuracyScore** measures right classification ratio over the total number of records.
- **specificityScore** measures “false positives” in terms of distance from the actual class to the predicted one.

Analysis and Tests

In order to run all the tests below, the script follows this procedure:

1. Load a dataset (1K, 5K, 10K).
2. Pre-filter it to reduce the records to the 5 classes presented before.
3. Create a test and validation subsets (2/3 & 1/3 each).
4. Feed the genetic algorithm with an initial population with the size of the genes collection.
5. Run the evolution; check fitness on each individual on each generation by launching the classifier. Use the weights matrix to compute a score and return the fitness value.
6. A best individual is returned, which contains a set of genes that should produce the best results found with the given classifier. Of course, changing classifier might produce different results, as it would happen if other factors like co-occurrence and correlation and frequency analysis.

Test #1

Testing with a big population on the biggest dataset, it took several hours. Generations raised to 100 to allow convergence (which happened around the 65th iteration).

Weights are set so that accuracy is important then specificity and finally length is a little bit considered. It is interesting how starting from a small population (30) is easier to converge to smaller chromosomes (14 in this case) than when using larger initial populations (50 which converged to around 40 genes). The size of the dataset helps here identifying better candidates.

Finally, the specificity matrix is set based on the strong precautionary principle and therefore it is better to be diagnosed with a more serious condition than the actual one (acute vs chronic, chronic vs healthy). The matrix is not symmetric to emphasize "false positives" (more severe condition detected than it is actually suffered) than "false negatives" (which would lead to a deterioration in the patient's health).

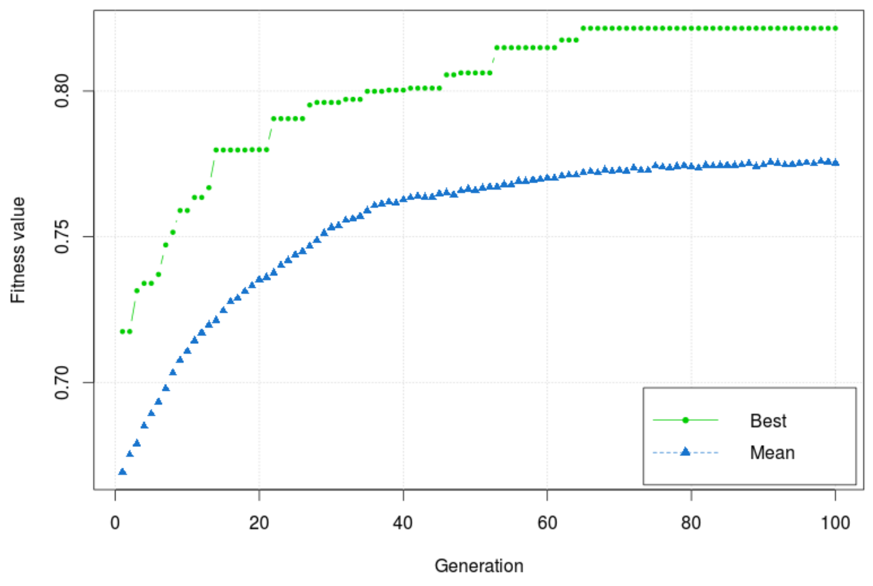

Figure 2: Fitness value over generation for test #1

Configuration

```
datasetSize             <- 10000;
maxGeneSelectionSize    <- 30;
populationSize          <- 1000;
numGenerations          <- 100;
weight.chromosomeLength <- 0.1;
weight.accuracy         <- 0.6;
weight.specificity      <- 0.3;

# "AML", "CML", "ALL", "CLL", "NO"
scoreByClassesMatrix <- data.frame(
  c(1.00, 0.50, 0.75, 0.25, 0.00),
  c(0.50, 1.00, 0.25, 0.75, 0.50),
  c(0.75, 0.25, 1.00, 0.50, 0.00),
  c(0.25, 0.75, 0.50, 1.00, 0.50),
  c(0.50, 0.75, 0.50, 0.75, 1.00)
);

elitism = 0.05
mutation = 0.1
```

Best result

```
Scores:
  length: 0.633333333333333
  accuracy: 0.911722141823444
  specificity: 0.676229508196721
  total: 0.813235470886416

Iter = 100  | Mean = 0.7753402  | Best = 0.813235470886416
```

Best chromosome (14 variables)

```
18, 48, 72, 172, 226, 589, 596, 1025, 1406, 1821, 2342, 2450, 2644, 6686
```

Test #2

Compared with the first test, in this one we are giving more importance to the crossed classes identification, which we have named "specificity" in the context of this project. We have changed the specificity matrix so that differences are more evident between desired or tolerated and unwanted results. In addition, we have set weights so that specificity is slightly more important than accuracy, in the hope that this will lead the algorithm to different positions in the space of solutions.

Additionally, we've set this test for a short run so that we can quickly get some results (1000 genes, 200 individuals, 30 generations).

The results prove that we are actually getting slightly worse accuracy results (0.858 vs 0.911) but slightly better specificity results too (0.772 vs 0.676). So the increase in specificity is about one tenth but the decrease in accuracy is only around half that value. This might lead to somewhat less accurate tests but that in turn are less harmful for the patient when they are wrong.

In the next test, we will extend this conditions to a larger population and gene set to see whether by increasing the space of solutions we can get a better convergence.

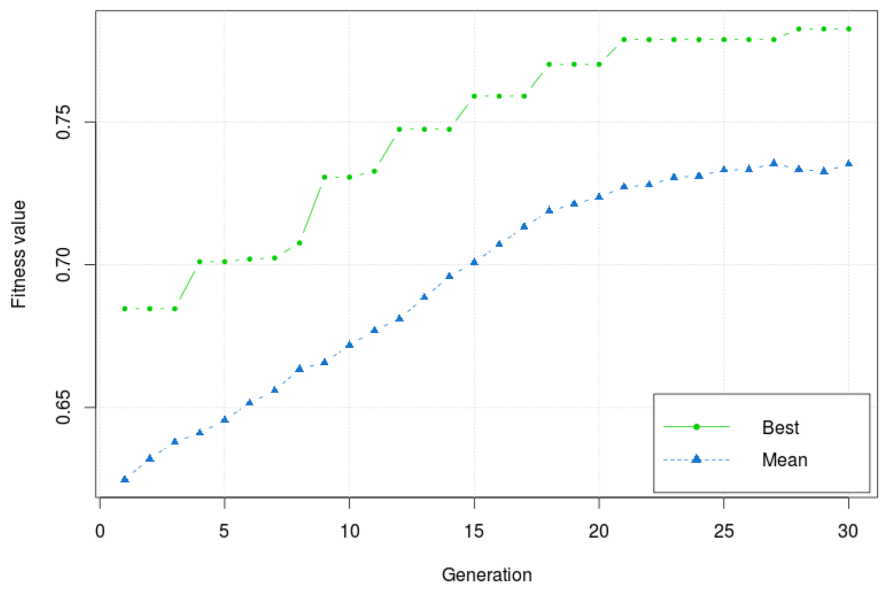

Figure 3: Fitness value over generation for test #2

Configuration

```
datasetSize              <- 1000;
```

```
maxGeneSelectionSize    <- 30;
populationSize          <- 200;
numGenerations          <- 30;
weight.chromosomeLength <- 0.1;
weight.accuracy         <- 0.4;
weight.specificity      <- 0.5;

# "AML", "CML", "ALL", "CLL", "NO"
scoreByClassesMatrix <- data.frame(
  c(1.00, 0.25, 0.90, 0.10, 0.00),
  c(0.90, 1.00, 0.80, 0.70, 0.25),
  c(0.90, 0.10, 1.00, 0.25, 0.00),
  c(0.70, 0.80, 0.90, 1.00, 0.25),
  c(0.50, 0.75, 0.50, 0.75, 1.00)
);

elitism = 0.05
mutation = 0.1
```

Best result

```
Scores:
  length: 0.533333333333333
  accuracy: 0.858176555716353
  specificity: 0.771938775510204
  total: 0.782573343374976

Iter = 30  | Mean = 0.7351626  | Best = 0.7825733
```

Best chromosome (14 variables)

```
12, 95, 104, 258, 281, 301, 338, 348, 414, 424, 518, 564, 732, 857
```

Test #3

This third test takes the same premises than the test #02 but enlarges the research to a bigger dataset (5000 genes), with a bigger population size (1000) and 100 generations so that the algorithm is able to converge.

The results are quite similar so that we can conclude that there might not be a better solution to be found that maximizes accuracy while minimizing the effect of negative specificity. There is, however, a remarkable improvement in the size of the chromosome found (9 genes) which gives us some hope since, even if not more performant, hypothetic leukemia tests may be less expensive.

We can also spot a pattern: the more we prefer specificity over accuracy, the higher is the cost to improve the first one in terms of diminishing the second one.

As we can see in this and previous genomes, several genes over 1000 are involved in the leukemia detection, but only one (Test #01) above 5000. We will be using this dataset in future test for better results.

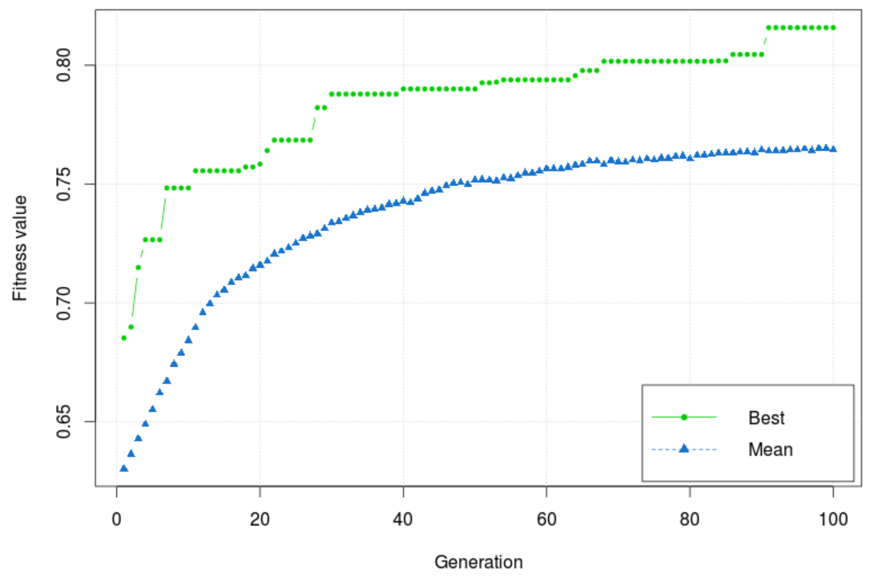

Figure 4: Fitness value over generation for test #3

Configuration

```
datasetSize             <- 5000;
maxGeneSelectionSize    <- 30;
populationSize          <- 1000;
numGenerations          <- 100;
weight.chromosomeLength <- 0.1;
weight.accuracy         <- 0.4;
weight.specificity      <- 0.5;

# "AML", "CML", "ALL", "CLL", "NO"
scoreByClassesMatrix <- data.frame(
  c(1.00, 0.25, 0.90, 0.10, 0.00),
  c(0.90, 1.00, 0.80, 0.70, 0.25),
  c(0.90, 0.10, 1.00, 0.25, 0.00),
  c(0.70, 0.80, 0.90, 1.00, 0.25),
  c(0.50, 0.75, 0.50, 0.75, 1.00)
);

elitism = 0.05
mutation = 0.1
```

Best result

```
Scores:
  length: 0.7
  accuracy: 0.819102749638206
  specificity: 0.8168
  total: 0.806041099855282

Iter = 100  | Mean = 0.764375  | Best = 0.806041
```

Best chromosome (9 variables)

```
184, 605, 682, 1110, 1170, 1226, 2717, 3982, 4023
```

Test #4

In this test, we are sticking to the 5000 dataset since it seems to be good enough for genes recognition, and faster than the 10000 dataset.

Since it seems quite easy to find good accuracy results, but it is more difficult for the specificity, we are going to tweak a little bit the weights to prefer the latter above the former.

In addition, since it seems difficult to improve specificity score over 0.8, and the research space is so big, we want the algorithm to "jump" more often but to keep the best individuals it finds. Therefore, we will be setting the GA parameters to a higher mutation rate (0.2 vs 0.1), and a lower elitism (0.01 vs 0.05), so that we can go find in different places in the space of solutions.

Looking for more divergent solutions might be slower than the previous tests, so we will keep the 1000 individuals population for 100 generations.

As a conclusion, however, the results are not really better than in Test #03. An interesting pattern is that, to slightly improve the specificity, accuracy drops dramatically. We can conclude that a perfect solution does not exist, or that it cannot be found with the current methods used.

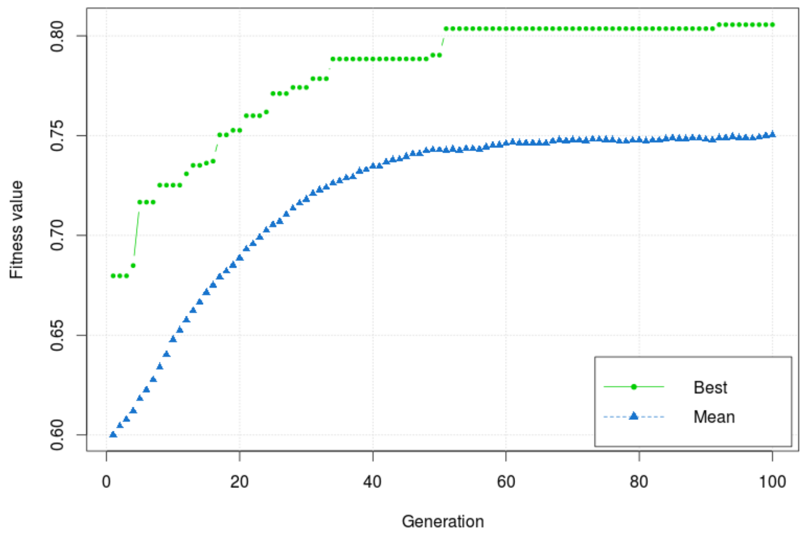

Figure 5: Fitness value over generation for test #4

Configuration

```
datasetSize             <- 5000;
maxGeneSelectionSize    <- 30;
```

```
populationSize          <- 1000;
numGenerations          <- 100;
weight.chromosomeLength <- 0.1;
weight.accuracy         <- 0.3;
weight.specificity      <- 0.6;

# "AML", "CML", "ALL", "CLL", "NO"
scoreByClassesMatrix <- data.frame(
  c(1.00, 0.25, 0.90, 0.10, 0.00),
  c(0.90, 1.00, 0.80, 0.70, 0.25),
  c(0.90, 0.10, 1.00, 0.25, 0.00),
  c(0.70, 0.80, 0.90, 1.00, 0.25),
  c(0.50, 0.75, 0.50, 0.75, 1.00)
);

elitism = 0.01
mutation = 0.2
```

Best result

```
Scores:
  length: 0.6
  accuracy: 0.769898697539797
  specificity: 0.827044025157233
  total: 0.787196024356279

Iter = 100  | Mean = 0.7503561  | Best = 0.787196
```

Best chromosome (11 variables)

```
28, 605, 695, 2182, 2189, 2555, 2788, 2827, 3160, 3628, 4833
```

Futures researches

The preparation of the data, modelization of the problem and initial tests and parameter tweaking took as much of the available time for this project. However, we would like to number for the record a few improvements or future researches that could continue our work.

Different classifier

Any classifier can be wrapped by the genetic algorithm, provided there is a list of classification results and/or a list of genes used for that result. In our project, we only tried to wrap an KNN (K-nearest neighbors) and C5.0 classifier. However, the C5.0 produced an internal error during its classification process which make it totally unusable, presumably due to the big amount of data (memory leak?).

SVM, fuzzy rules systems, a genetic algorithm nested into the wrapper, any technique for MLBD we have seen during the course can be harnessed in this approach.

Process all available genes

The initial dataset holds about 54'000 different genes per observation. However this huge amount of variables couldn't be processed by our computers due to memory issues. Therefore a preselected subset of these variable was used. So, we might have lost some important combinations with this preselection and it would be extremely interesting if we could validate our model with a real big dataset. This would have even a bigger impact, explained in the following subsection.

Co-occurrence of variables

For a sake of simplicity, we have only considered the result of the classifiers (correctly/incorrectly classified records. However, we have not considered the co-occurrence, neither the frequency of variables on the results of these classifiers.

It is more than possible that some genes act in combination with others and our current model is completely oblivion to that fact. Getting the list of genes used by the classifier and building a histogram on the fly would be a first improvement in order to add a fourth evaluation factor (chromosome size, accuracy, specificity): use of frequent genes. This way, a chromosome that makes use of genes frequently used will have a better score and therefore the algorithm will converge faster to proven genetic combinations. There is the risk, however, that it might converged to a local minimum since the global scope would be quickly dismissed.

The second (and probably best) improvement on this sense would be to have a co-occurrence global matrix where values are being updated as the classifiers return their results. We wouldn't be counting how often a gene is used, but how often it is used in combination with the others. This technique would allow to keep together combinations that are often found useful by the classifiers, and that might, therefore, be part of the global solution. For this technique to be effective, combinations of several sizes should be considered (2 genes together, 3, 4...).

A drawback is that computational complexity increases with the algorithms implementation, so these improvements should be considered carefully and their improvement weighted thoroughly.

Behind this approach, the idea is to bias the performance when patterns are detected to help the genetic algorithm to include them into future generations. This kind of matching could be done with algorithms like Apriori for example.

Conclusion

The main conclusion we can draw from our experience is that there is **tradeoff** to make when dealing with complex systems and NP-complete problems. In this particular case, *accuracy*, *specificity*, *chromosome size* and *computation resources* have to be balanced and it will depend on the actual real case that we use the model on a way or another.

More precisely, we can only get more specificity by decreasing in accuracy. Chromosome size can be reduced but it impacts both specificity and accuracy. Finally, better or more complete methods would increase the computational resources needed. From our point of view, the best test scenarios were:

1. **Test #3**. With only 9 genes, it produces an accuracy of 0.819 and specificity of 0.817. This is the best balance between the three dimensions we have got.
2. **Test #1**. With the best accuracy (0.912), it counts with only 14 genes and has a specificity of 0.676. Useful when accuracy is more important than specificity.

An additional interesting point drawn from this project is that with the right modelization and correctly nesting processing blocks, **big problems can be split** into several smaller ones, and therefore be approached more easily.

Finally, the importance of correctly understand the **interests behind a problem** are capital. While we might think that accuracy is the only score that matters, smaller genomes produce cheaper tests and more specific tests are too expensive to achieve. The “real world” and “human” factors have to be considered, and we have made our model parametrable with the purpose of adaptability in mind.