NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Computational Intelligence for Optimization

Project – Optimizing pharmolocological bioavailability predictions

Diogo Tomás Peixoto – 20210993
João Morais Costa – 20211005
Gonçalo Gomes – 20211007

Github: https://github.com/gnmg7900/cifo_project

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Introduction and problem description

Drug discovery is often a very laborious and expensive process, namely when trying to predict pharmacological characteristics of new compounds, such as toxicity or bioavailability. This is where our project finds its context. Starting from a dataset of known chemical compounds from which the bioavailability is known, our goal is to generalize by finding a function that makes a prediction of the bioavailability with the minimum error possible.

To tackle the problem of predicting bioavailability, we have implemented a multiple linear regression, as shown in figure 1, with genetic algorithms (GAs) as the optimization function, in order to find the coefficients (slopes) values of each molecular descriptor, that minimizes the error between the prediction and the compared target. As a fitness function, we have used the root mean square (RMSE), which implementation is detailed later in this document.

Our dataset is composed of 359 rows and 242 columns. Each row represents a different pharmacological compound. The first 241 columns are the different molecular descriptors, and the last one is the bioavailability of the compound, which corresponds in the model defined in the figure 1, by the independent variables (Xi) and the dependent variable (Y), respectively.

$$\boxed{Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_i X_i}$$

Y : Dependent variable
β_0 : Intercept
β_i : Slope for X_i
X = Independent variable

Figure 1 - Multiple linear regression model

1. The Individuals Representation

The representation generated for our problem is a fixed length vector with 242 elements: the first value is b_0 (the regression line intercept) and the remaining 241 values are the coefficients (slopes), for each molecular descriptor (feature or X_i), represented from b_1 to $b_{241.}$

In order to find an appropriate valid set range, different representations were generated with different values ranges and then tested in parallel using the same configuration (*Selection algorithm*: Tournament; *Crossover algorithm*: Single point crossover; *Mutation algorithm*: Swap Mutation; *Generations*: 5; *Population size*: 100; *pc*: 0.8; *pm*: 0.2). The results retrieved were summarized on table 1 to facilitate the interpretation.

It is relevant to mention, the set of values tested are varying with three decimal places between the range presented with square brackets, except the first two iterations, which allow values with 4 decimal places.

Validation Set	Run	Fitness		Validation Set	Run	Fitness
[0 : 0.001] (+0.0001)	1	70.53		[-0.001 : 0.001] (+0.0001)	1	70.64
	2	70.48			2	70.62
	3	70.48			3	70.58
	4	70.39			4	70.47
	5	70.40			5	70.59
	Avg	70.46			***Avg***	70.58
	Std	0.053			***Std***	0.059

Validation Set	Run	Fitness		Validation Set	Run	Fitness
[0 : 0.01] (+0.001)	1	60.51		[-0.01 : 0.01] (+0.001)	1	60.88
	2	60.4			2	60.76
	3	60.49			3	60.66
	4	60.31			4	60.54
	5	60.34			5	60.47

	Avg	60.41			***Avg***	60.66
	Std	0.079			***Std***	0.148
Validation Set	Run	Fitness		Validation Set	Run	Fitness
[0 : 1] (+0.001)	1	72.49		[-1 : 1] (+0.001)	1	258.36
	2	58.79			2	40.34
	3	97.03			3	42.51
	4	237.04			4	53.6
	5	76.27			5	39.03
	Avg	108.32			***Avg***	86.77
	Std	65.51			***Std***	85.95
Validation Set	Run	Fitness				
[0: 0.1] (+0.001)	1	**45.96**				
	2	**43.87**				
	3	**43.87**				
	4	**43.71**				
	5	**43.93**				
	Avg	**44.27**				
	Std	0.849				

*Table 1: Valid set testing (**Avg** – Average; **Std** – Standard deviation)*

It was noticeable that the range values allowed to be taken by the individuals during the generation phase would have a huge impact on the final GA fitness value. Out of the seven iterations tested, the lowest average fitness value was obtained for the interval [0 : 0.1], with incrementation of 0.001. This means, that the coefficients could take any value between the referred interval with three decimal places. We acknowledge that better fitness values could be obtained for different interval values and/or values with higher decimal places. Anyway, we have decided to not further iterate, since the reason behind is comprehended and the results with the current solution would be sufficient.

2. The Fitness function

In order to measure the difference between the target real value and the predictions obtained, several different regression metrics are available, namely: Mean square error (MSE), root mean square error (RMSE) and mean absolute error (MAE). Each of these metrics have advantages and disadvantages. The RMSE has been chosen as our fitness function supported by the theoretical properties of this metric. RMSE, as an extension of MSE, "punishes" models with large errors, since it inflates the average error score, at the same time being less sensitive than MSE to the presence of outliers. It is a very common metric and is also a good general-purpose error metric for numerical predictions [1].

In summary, our fitness function (RMSE), is the square root of the mean of the square of all the errors between the real value of the target value and the prediction obtained.

The fitness function code implemented on the "Bioavailability.py " python file, follows the reasoning below:

1- Starts by creating an empty list called "*predicted_values*"
2- Iterate using a *For loop* with the length of the number of compounds, in this case 359.
    - In each iteration, each element of the compound vector with the index[i] is multiplied elementwise with the representation vector, starting from the element with the index[i+1] (that corresponds to b_1).
    - To the result from the previous operation is added the element of the representation vector in the initial position (index [0]) (that corresponds to b_0).

- The result is the predicted value, which is appended to the *“predicted_values”* list.

The steps described above are summarized in the following figure.

$$\begin{bmatrix} \hat{Y}_1 \\ \hat{Y}_2 \\ \hat{Y}_3 \\ \vdots \\ \hat{Y}_n \end{bmatrix} = \begin{bmatrix} \beta_0 & \beta_1 x_{11} & \beta_2 x_{12} & \beta_3 x_{13} & \cdots & \beta_m x_{1m} \\ \beta_0 & \beta_1 x_{21} & \beta_2 x_{22} & \beta_3 x_{23} & \cdots & \beta_m x_{2m} \\ \beta_0 & \beta_1 x_{31} & \beta_2 x_{32} & \beta_3 x_{33} & \cdots & \beta_m x_{3m} \\ \vdots & & & & & \\ \beta_0 & \beta_1 x_{n1} & \beta_2 x_{n2} & \beta_3 x_{n3} & \cdots & \beta_m x_{nm} \end{bmatrix}$$

Figure 2: Matrix representation of the multiple linear regression model

3- Using the scikit-learn MSE function, the *“predicted_values”* list is then passed as an argument, together with the target value vector (y), retrieved from the initial dataset.
4- The former step is nested inside the *numpy* function that retrieves the square root, retrieving the root mean square error of the representation, as the fitness function.

3. GA Parameters Definition

Some parameters have been set *a priori* before running the GAs.

The **Population size** is equal to 100. The value has been chosen with sufficient size to allow to explore different solutions. Nonetheless, we have understood that a higher value would not compensate for the increment in the computational effort.

The number of **generations** has been set to 100. The idea behind it, is to guarantee the GA’s convergence, which has been achieved based on the results presented in question 6.

With regards to the **selection algorithm**, we have used the *fitness proportion selection*, the *tournament* and the *ranking* selection. It is important to mention that, in the *fitness proportion selection* implementation, we had to make some changes due to the fact that we were facing a minimization problem. Here, we aimed to associate solutions with lower fitness to a higher probability of being chosen. In this sense, for this particular algorithm, we had to reformulate the fitness to its inverse, so that it was possible to maintain its logic. Regarding the *tournament* algorithm, it has been set with a size equal to 50. The *ranking* method, considering its algorithm definition, leads to a higher chance of individuals with smaller fitness being picked to the next generation. Therefore, we do believe that’s the least efficient selection method considering our problem definition.

The **crossover** and **mutation** rate have been chosen with the value 0.80 and 0.20, respectively. According to Leonardo’s booklet “*This is similar to what happens in nature, where crossover is an event with much higher probability than mutation. This is also the configuration that allows GAs to obtain the best results most of the time.*” [2]

Concerning the **crossover** algorithms, the *single point* and *arithmetic* crossover were the only ones used, since the *cycle* and *partial matched* crossover couldn’t be used in our problem. Its algorithm definition requires both parents to have the same values in order to be able to produce the offspring’s. That’s a requirement not possible to achieve since our individuals have a high variability of values in their representation.

The *swap* and *inversion* mutation were the **mutation** techniques used in our project. We have also developed the *insertion* mutation, which selects a value at random and inserts it at a random position. Further information might be consulted on the reference [3].

With reference to **elitism**, the selection, crossover and mutation algorithms used to build a new population might lead to a worse best individual fitness in that generation compared to the previous one. That’s the reason why elitism technique is used, to assure the best individual fitness for each population after each new generation is at least as good as the previous one. Based on that, we have always implemented elitism.

4. Minimization problem

The implementation is a concrete multiple linear regression problem with the goal to find the slopes, through the genetic algorithms, that minimize the root mean squared error and hence, provide the closest predictions compared to the target. Therefore, this problem works only with minimization. If we would have used maximization, basically we would get consecutively a worse linear regression model, providing predictions further distant from the target and moving against the goal of machine learning development models.

5. Results

In the *"Charles.py*" file, we have created the attribute *fitness_list*, as a list, under the population class, to store the best individual fitness after each generation. With that information, we have managed to plot the results in a line graph, as shown in figure 3, to compare and understand the influence of different configurations on the GA convergence.

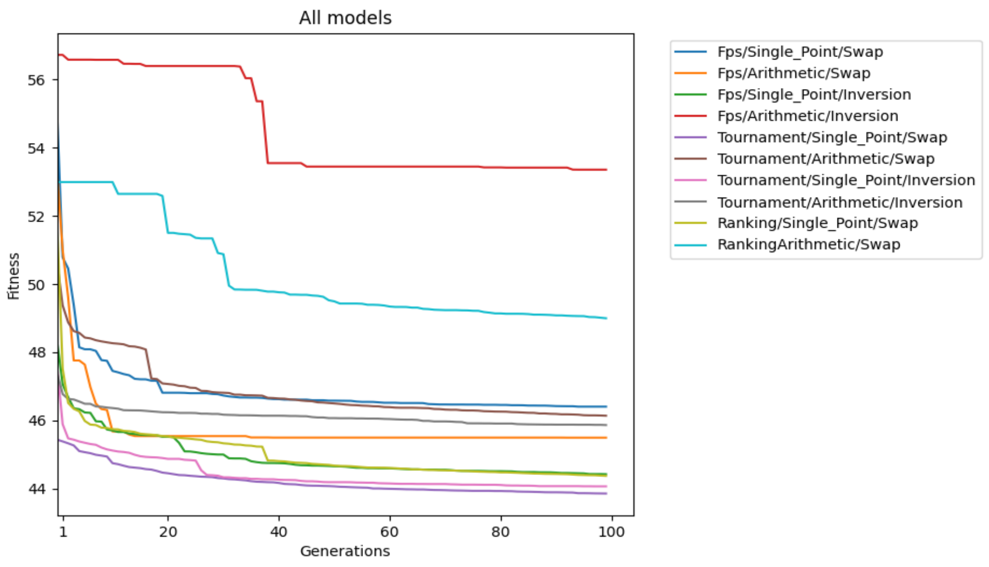

Figure 3: Results with different models along 100 generations

To make this analysis statistically representative, each model in the graph corresponds to the average of the respective 5 runs we performed. In total, we have developed 10 models.

The main insights drawn from the results in figure 3 are:

1. The two best configurations with the lowest associated error have in common, the application of the *tournament* and *single point*, as the selection and the crossover algorithm, respectively.
2. The four best configurations, all presented *single point* as a crossover algorithm.
3. The results show the convergence of the different models, since the line plots show a line approximately horizontal on the last generations. It can be said, the models are converging to the local optimum
4. There's a certain correlation between the first and last fitness values, meaning to say, the lowest the fitness value is in the first generation, the lowest will be in the last one

From the first insight, it is not a surprise that the two best results are with the *tournament* selection. This method has a higher probability of choosing the best individuals for each next generation, especially if its size parameter is high in relation to the population size, compared to the other selection algorithms.

Regarding the second insight, the use of *single point* crossover gives better results compared with the *arithmetic* one. Looking into both algorithms, whilst the *single point* crossover only rearranges the values from both parents onto the offspring's, the *arithmetic* outputs different coefficient values, which introduce more randomness along the GA algorithm, which can justify the results presented.

Regarding the mutation algorithms used, our results suggest that none of the algorithms used (*swap mutation* and *inversion mutation*) have an apparent effect on final fitness results. The low probability of occurrence of mutation in the generated individuals (20%) certainly influences these outcomes.

6. Future Work

In this project, the goal was to decrease the fitness of our best solution as much as possible, that is, to minimize the RMSE.

With this, we tested and parameterized the applied GA's as well as the various representative components, such as the valid set. The latter has a big impact on the reduction of RMSE associated with our solutions, as seen previously on table 1. Definitely, if we would continue to fine tune the valid set, by testing different range values and also increasing the number of decimal places, a reduction of the RMSE would be achieved.

Considering an overall RMSE of 44, it means the average error is around 2.32, as show below. Therefore, we might conclude the results are fairly good.

$$\widehat{(y_i} - y_i) = \frac{\sqrt{44^2 * 359}}{359} = 2.32$$

Another relevant test that could be performed is the exchange between the crossover and mutation probability occurrence, as suggested in Leonardo's booklet.

Regarding future implementations of the multilinear regression model trained, it would be relevant to study the correlation between the theorical individual and the target function trained. That's because if the inputs are outside of the trained range values, there's the probability that further distant we are from the training set, bigger will be error between the predicted and the real target. If the correlation is studied, and is confirmed that both functions are negatively correlated, as shown on the figure 4, then a warning shall be raised advising the model shall only applied on the range of the values trained. Otherwise, an average error above 2.32 is possible.

Regarding possible implementations to consider, it would be interesting to include a possible metric that evaluates the correlation between the regression lines. This would be a potentially relevant option, since, at the moment, we are only considering the error between the target function and a certain individual from the population. Therefore, although it is likely that we will find reduced errors in our dataset (considered a good solution), this same individual may be uncorrelated with the target function being, in fact, a bad solution when faced with new input values, not represented in the original dataset. This same idea can be visualized in the image below.

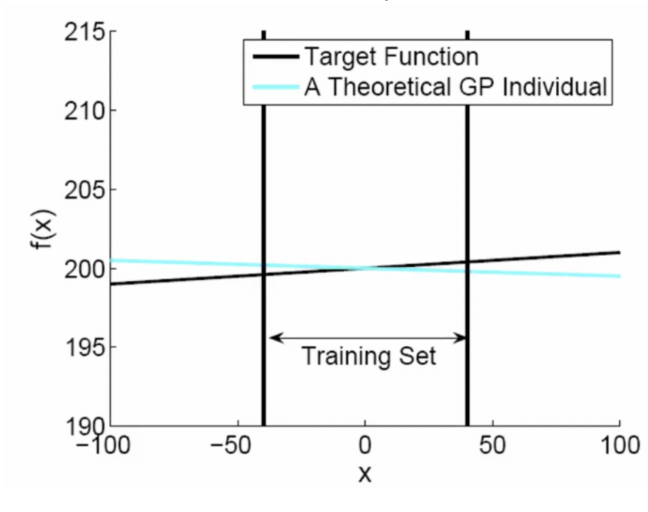

Figure 4: Possible Correlation with a specific GP Individual

Project Division: Regarding the division of tasks, all group members had similar participation in all tested and implemented methodologies.

References

[1] – Mishra, Divyanshu – “Regression: An Explanation of Regression Metrics and What can Go Wrong” – Towards Data Science – Dec 6, 2019 – Available at: https://towardsdatascience.com/regression-an-explanation-of-regression-metrics-and-what-can-go-wrong-a39a9793d914

[2] – Vanneschi Leonardo –“Computational Intelligence for Optimization” – 2022

[3] - Mebrahtu, Hadush, Deep, Kusum – *“Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem”* - International Journal of Combinatorial Optimization Problems and Informatics- 2011, 2(3), 1-23. Available at: https://www.redalyc.org/articulo.oa?id=265219635002