GRAMMATICAL EVOLUTION FOR THE PREDICTION OF HYPOGLYCEMIA IN DIABETES

Gramáticas Evolutivas para la predicción de hipoglucemias en diabetes

MARINA DE LA CRUZ LÓPEZ

Trabajo Fin de Grado en Ingeniería Informática
Facultad de Informática, Universidad Complutense de Madrid

Mayo de 2022

dirigido por

Carlos Cervigón Rückauer
Departamento Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Resumen

Los pacientes con diabetes deben controlar correctamente su nivel de azúcar en sangre para evitar complicaciones. Una de estas complicaciones es la hipoglucemia o nivel bajo de azúcar en sangre, que ocurre cuando la concentración de glucosa en la sangre cae por debajo de cierto umbral. Un episodio de hipoglucemia debe corregirse antes de que se vuelva dañino y puede ser una situación muy angustiosa para el paciente.

El objetivo principal de este estudio es programar los algoritmos de Gramáticas Evolutivas Estructuradas y Gramáticas Evolutivas Estructuradas Dinámicas, y utilizarlos para generar modelos de predicción de episodios hipoglucémicos en pacientes con diabetes.

Los algoritmos se utilizarán para obtener un modelo de caja blanca formado por sentencias if-then-else que, dados unos datos de entrada, compuestos por el nivel de la glucosa en sangre y las lecturas de datos de ejercicio de los pacientes de las 2 horas anteriores, optimiza una relación lógica entre estas variables. La fórmula resultante se usa para determinar si el paciente va a tener un episodio de hipoglucemia en un plazo de 30, 60, 90 y 120 minutos.

Palabras clave

Diabetes, Gramáticas Evolutiva, Algoritmos Genéticos, Aprendizaje Automático, Predicción de Glucosa, Gramáticas Evolutivas Estructuradas, Gramáticas Evolutivas Estructuradas Dinámicas

Abstract

Diabetic patients have to manage their blood sugar correctly to prevent complications. One such complication is hypoglycemia or low blood sugar, which occurs when the blood glucose concentration goes below a certain threshold. A hypoglycemic episode needs to be rectified before it becomes harmful and can be a very distressing situation for the patient.

The main goal of this study is to program Structured Grammatical Evolution and Dynamic Structured Grammatical Evolution algorithms and use them to generate models for the prediction of hypoglycemic episodes in patients with diabetes.

The algorithms will be used to obtain a white-box model made up of if-then-else statements that given some input data, comprised of the blood glucose and exercise readings of the patients from the previous 2 hours, optimizes a logical relation between these variables. The resulting formula will be able to determine if the patient is going to have a hypoglycemic episode in a 30, 60, 90 and 120 minutes prediction horizon.

Keywords

Diabetes, Grammatical Evolution, Genetic Algorithms, Machine Learning, Glucose Prediction, Structured Grammatical Evolution, Dynamic Structured Grammatical Evolution.

Contents

Acknowledgments

Firstly, I would like to thank my family for supporting me during these past years, my classmates and friends who have studied alongside me and all the teachers that have been a part of my education. An special thanks to Carlos Cervigón Rückauer and J. Ignacio Hidalgo for aiding and mentoring me through the creation of this project. I am also thankful to Jorge Alvarado for compiling the data and to all the patients whose data was used to perform this study.

A final thanks to the Faculty of Computer Science of the UCM that has made all of this possible, and to the Adaptive and Bioinspired System Group for giving me the opportunity to add to this great project.

Chapter 1

Introduction

1.1 Introduction

Diabetes is an illness characterized by high blood sugar levels, these levels are caused by insulin absorption deficiency that makes the body unable to keep the blood glucose levels within a normal range. This can be a result of either the body not producing enough insulin or the cells not being able to use it properly. There are two main types of diabetes.

- Type 1 diabetes, the pancreas is not able to produce enough insulin to control blood sugar levels as a result of the loss of the beta cells of the pancreas that produce it, most commonly due to an autoimmune response from the body. Type 1 patients need to inject insulin to keep their normal blood sugar level.

- Type 2 diabetes, is generally caused by a high insulin resistance that forces the pancreas to make more insulin than normal to maintain healthy blood glucose levels, as a result of this the pancreas might eventually stop generating enough insulin and end in a state similar to that of type 1 diabetes, although its development takes longer and can be stopped or reversed in some cases through a lifestyle change early on in the illness development. In this case, depending on the evolution of the illness there are different treatment options.

Keeping blood sugar levels within a certain range is imperative to maintain health and prevent complications that can result from a bad blood glucose management, therefore it's imperative for a person with diabetes to have good control at all times.

For our study, we will be centering ourselves in Type 1 insulin dependant patients as, by the nature of the insulin injections, and other factors, they have more blood sugar variability and will be at risk to suffer some of the states described thusly.

1.2 Hypoglycemia

Hypoglycemia, or low blood sugar, is a state that can happen most commonly in people with diabetes. It is due to an excess in the active insulin that results in too much of the blood sugar being absorbed. It is considered hypoglycemia once the blood sugar level is less than 70 milligrams per deciliter (mg/dL), or 3.9 millimoles per liter (mmol/L). The body has glucose homeostasis mechanisms that prevent hypoglycemia from occurring in most cases, however people with diabetes have a sub-optimal glucagon response[4] and other counter-regulatory hormones of insulin, making it easier to enter a hypoglycemic state. Having blood sugar level below a certain threshold can seriously affect brain functions and produces a nervous response in the body, some of its symptoms include, sweating, dizziness, hunger, shakiness or trembling. If not treated it can develop into severe hypoglycemia and result in the person passing out, or even death in extreme cases.

Hypoglycemic episodes should be prevented for various reasons:

- The range of values considered is relatively small and therefore the earlier a potential hypoglycemia is caught the easier it is to prevent it from developing further.
- Having a lot of hypoglycemic episodes makes the body get used to them so the symptoms that characterize them become less apparent, this makes the person less likely to recognize them early on without measuring their blood sugar.
- An episode might lead to an over-compensation and result in hyperglycemia or high blood sugar later on.

The main issue regarding being able to prevent hypoglycemia events is that an excess in the insulin injected is not always easy to realize, because it depends on the body's insulin sensitivity, which can vary during the day depending on different factors.

One of the causes of variation in insulin sensitivity is the practice of exercise, which enhances it. It increases the ability of the body to absorb glucose and transform it into energy with less amount of insulin. This can be very helpful for a person with diabetes, and as such, it is recommended to perform regular exercise to stabilize glucose values. The less insulin a person has to inject to obtain the same result, the more stable their blood glucose will be, and it also prevents hyperglycemia episodes or high blood sugar.

However, increasing insulin sensitivity means that the person may be at risk of hypoglycemic episodes from past insulin injections and future ones if they are not calibrated correctly in accordance with the amount of exercise performed. This increased risk is one of the reasons why people with diabetes are often scared to perform exercise, even though the positives heavily outweigh the negatives. Since the symptoms of hypoglycemia are so noticeable, the thought of entering into a hypoglycemic state can cause a psychological reaction in the person that makes them afraid of performing activities that might cause a hypoglycemic state, even if this is counteractive to their well-being.

For this study, we will use exercise (or activeness) data and glucose values to predict future hypoglycemia episodes at different time horizons.

1.3 Objective

In order to predict future hypoglycemic events, we will use Structured Grammatical Evolution, both the original SGE and Dynamic SGE, to obtain a white-box model composed of if-then-else statements that perform a classification of future hypoglycemia episodes. These if-then-else statements will have a series of input variables whose values determine what class we obtain as a result. The variables considered will be the glucose readings of the last two hours in five-minute intervals, obtained by means of a continuous glucose monitor

(CGM), and the exercise readings of the last 2 hours in 5-minute intervals. From this, we will consider the number of steps, heart rate, and calories burned, these values can be tracked by using a smartwatch or smart band. All the input variables can be obtained without any need for user input, making them easier to get from patients. It is important to emphasize that we will not take into account the insulin injections and carbohydrate intake, the only information we are considering is the reaction that can be seen in the glucose variation. Not including these values can have negative results, especially in postprandial glucose readings, where they affect a lot of the resulting glucose values.

In terms of the prediction, we will consider 4 different prediction horizons $k \in [30, 60, 90, 120]$ minutes with two different classification methods. We will also compare the results of our algorithm with the results obtained with other classification algorithms.

The results obtained depend on the model and patient, but as a brief summary, we are able to achieve around 90% to 95% average Recall in the 30-minute predictions, 85% to 90% average Recall in the 60-minute predictions, 80% to 90% average Recall in the 90-minute predictions and 70% to 80% average Recall in the 120-minute predictions.

The rest of the paper is structured as such: Chapter 2: Related Work, Chapter 3: Methodology followed, Chapter 4: Patient data used for the prediction, Chapter 5 :Results obtained and conclusions.

Chapter 2

Related Work

As mentioned above, hypoglycemia control is very important for patients with diabetes, so many techniques have been used to predict such hypoglycemia events. In this section, we will review some related work.

2.1 Introduction

According to the American Diabetes Association, hypoglycemia is the limit to the level of insulin that a patient can inject to control their blood sugar. Reducing the mean blood sugar level is imperative to prevent micro-vascular complications such as retinopathy, nephropathy, and neuropathy, however having blood sugar level below a certain level can seriously affect brain functions and produces the nervous responses described previously. A continuous hypoglycemia state can produce permanent brain damage, although this doesn't happen often. If this was not the case, glycemic control would be a lot simpler than what it currently is, since a patient must maintain glucose levels within a range and glucose variability is also a lot higher on average for a person with diabetes.[6]

There are different levels of hypoglycemia that range from mild hypoglycemia to severe hypoglycemia. However, even though they are mentioned in a lot of the studies shown below they are never differentiated in the prediction. This can be due to the fact that Continuous Glucose Monitors have a significant error threshold with lower glucose values in comparison to capillary measurements.

2.2 Symbolic Prediction using Genetic programming

Using genetic programming (GP), there have been different studies that perform symbolic predictions for future glucose values. Instead of classifying the data, genetic programming has been used to generate a real glucose value. This type of prediction is not focused on hypoglycemia but on predicting real glucose values. The methodology used will be very similar to ours, in the sense that, what we are obtaining is a mathematical formula which is our final model. This formula is then applied to all the user data to predict a real value. In this category we have studies that use Genetic Programming ([8,10,13–15,18,27]), and Grammatical Evolution (GE):[11],[12]. Probabilistic fitting in evolutionary optimized models has also been successfully applied[5]. This fitting has been added to palliate the effects of the 5% error that can be found in the glucose readings obtained from any kind of CGM. Instead of trying to fit our model to the reading from the CGM, we try to fit it to the 95% confidence range, which considers the error from the sensor whilst trying to fit it to the real data as well as possible.

2.2.1 Structured Grammatical Evolution

In terms of Structured Grammatical Evolution, which is our proposal for performing prediction, we can find studies showing that SGE performs better than the traditional Grammatical Evolution (GE)[11]. Dynamic Structured Grammatical Evolution has already been used for glucose prediction with good results in Lourenco et al[16]. Where they perform a symbolic prediction of real glucose values using past glucose values, insulin injections, and carbohydrate intakes. This study shows that DSGE does not only give better results on average but that the standard deviation between the results is much smaller than with a traditional GE approach.

2.3 Classification

In this study, we are interested in detecting hypoglycemic episodes in glycemic prediction through classification.

Different machine learning techniques have been used for hypoglycemia prediction, and they are summarized in[20]. We will mention briefly those that perform hypoglycemic classification in a similar way to what we are trying to do in this study. They use previous data values to try and predict future hypoglycemic occurrences, which may or may not have certain constraints.

2.3.1 Variables and prediction horizon

The input variables used for the model data are different in the different studies. Most of them use previous glucose values, either standalone or together with other data. The additional data can be insulin and carbohydrates, or like in our study exercise data, breath examples, etc. Other models do not use glucose values at all and perform the prediction only with electronic health record (EHR), Electrocardiogram (ECG), galvanic skin responses (GSR), electroencephalogram (EEG), clinical notes, secret messages, breath samples, and body temperature, however, these are in the minority.

The time window considered for the prediction also differs from study to study, some make shorts term predictions, 30 to 60 minutes, and others predict over a longer period of time (240 minutes to 6 hours). Generally, it will depend on when the prediction is being made, for example, if the study performs a nocturnal hypoglycemic prediction then the horizon will be 6 hours.

2.3.2 Machine learning algorithms

In terms of the type of machine learning algorithms used, most of the studies make use of Artificial Neural networks, but other techniques include Decision Trees, Regression, Kernel, Reinforced Learning, and Estimating Probability Distribution.

Artificial Neural Networks

In artificial Neural Networks, we have a set of studies that use similar data to what we are using as inputs, in a lot of cases the ANNs are used to perform specifically the prediction of nocturnal hypoglycemic episodes, these include: Bertachi et al[2], which performs predictions of nocturnal hypoglycemia's using an MLP and an SVM, it also uses the data from a CGM monitor and an activity tracker to perform the predictions, and yields results of 78.75% and 82.15% of sensitivity and specificity respectively. Vehi et al.[26] uses glucose information, insulin, carbohydrates, and exercise information which is then sent to a multilayer perceptron network that makes the classification. Others in the same vein include: San et al.[24], whose data subjects were type1 diabetic children, and Bertachi et al. "Prediction of blood glucose levels and nocturnal hypoglycemia using physiological models and artificial neural networks".

Others are focused on hypoglycemic prediction on short time horizons, such as Mosquera-Lopez et al[19], which uses Recurrent Neural Networks to predict adverse glycemic states (both hypoglycemia and hyperglycemia) in a 30-minute prediction time frame.

Decision Trees and Random Forest

Decision Trees are one of the most widely used classification techniques as they are highly interpretable, yielding a comprehensive white-box model as a result. Some studies use normal DTs to perform the classification such as Ranvier et al.[22], but most of them use a Random Forest method, where they generate a lot of decision trees and use the majority vote to make decisions. This yields better performance at the expense of some interpretability.

There are a lot of studies that use Random Forest to perform hypoglycemic classification, some are once more focused on nocturnal hypoglycemias, Güemes et al.[9], postprandial hypoglycemia's, Seo et al and hypoglycemia's experienced during the performance of aerobic exercise, Reddy et al[23]. To highlight the study by Dave et al.[7] that uses RF to perform prediction over a 30 and 60 minutes time horizons using only glucose data, the sensitivity values obtained were pretty high, obtaining values around 90% during the day and 94% at night.

Support Vector machines

Support Vector machines have also been used for this type of classification problem and reported good results. As before we have studies trying to predict nocturnal hypoglycemia, Mosquera-Lopez et al.[1] and Güemes et al.[9] and postprandial hypoglycemia, Vehi et al.[26] and Oviedo et al.[21] within the 240 min after a meal.

Logistic Regression

Logistic Regression is the final method we will comment on for the performance of the classification, although other methods have been also applied we can highlight the study by Dave et al.[7], that we spoke of in the RF, which also uses Logistic Regression to perform a prediction on 30 and 60 minute horizon, yielding values of sensitivity from 95% to 91%.

Conclusions

In general, from all the studies listed that perform data classification, we have determined that a high percentage of them perform the prediction at specific points in the glucose development, such as nocturnal or postprandial predictions, very few of them try to generalize a model that could work at any time.

Chapter 3

Methodology

In this chapter, we will explain how the classification method we have developed works. We will explain the implementation of both Structured Grammatical Evolution types and other operators needed to use them in an evolutionary algorithm. Operators that are not discussed in this section, such as selection, have not been implemented specifically for SGE and to perform the tests we have used the ones already available in the JECO library.

3.1 Grammatical Evolution

Grammatical Evolution (GE) is a variation of Genetic Programming (GP) that uses a grammar to generate the phenotype from the genotype of the individuals. The genotype is made up of a list of numbers, each one of which will generate a production from a rule made up of terminal and/or non-terminal symbols of the grammar.

The individual phenotype will be formed by the terminals of the grammar and have the structure defined by it.

The list that conforms the genotype is of fixed length, determined by the user. If we finish decoding the list and our individual's phenotype is still incomplete, we can go back to the start of the list and repeat the process for an n amount of times, called wraps, which are also defined by the user. If after this our individual is still incomplete we stop, give it a bad fitness, and do not compute it, the individual is considered invalid.

Each number on the lists could generate any of the productions forming the phenotype until there are no more possible expansions, and it has been completely created, to be able to generalize this process the number in the list go from 0 to a codon upper bound, for example, 256, and the modulus of the total amount of productions for one rule is the operation used to decode the result when generating the phenotype.

This type of evolution has a couple of issues. On one hand, since each allele depends on the previous one a small change in one can completely change the final result, making mutation and crossover a potentially destructive process. On the other, as we use a modulus operator to determine the next rule we could potentially change an allele, but the result generated might be the same one, it depends greatly on the number of productions a rule has, therefore mutating certain alleles might return the same result. Both these issues have an impact on the overall evolution of the result and are what the Structured Grammatical Evolution is trying to improve.

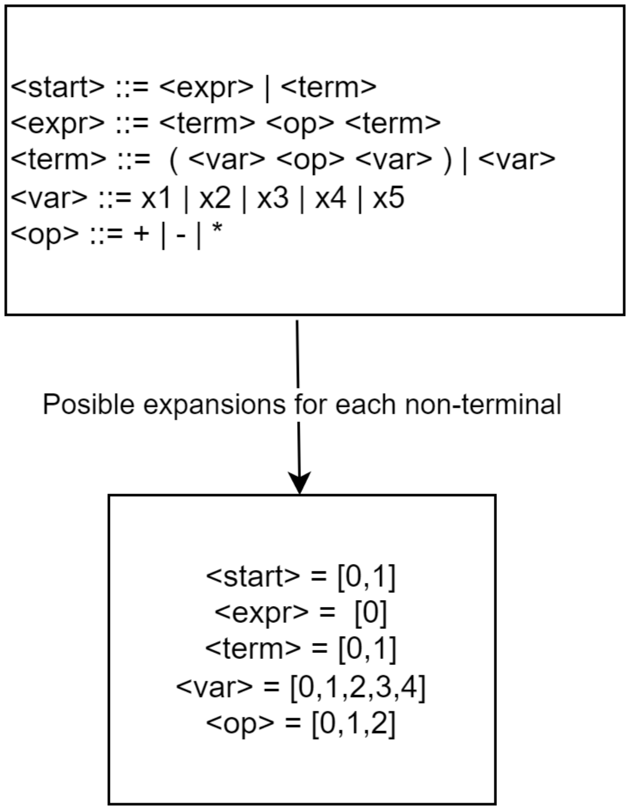
However, the ability to change the problem by changing the grammar rather than editing the code is a great feature of Grammatical Evolution (GE) because it makes it very flexible as it can work with any problem for which we have a Grammar created, without needing to perform any changes to the implementation. With different grammar files we can change the way our individuals will be structured, the input variables we want to consider, and how complex we want the final solutions to be.

3.2 Structured Grammatical Evolution

Although GE has been successfully applied to different problems, it presents some drawbacks derived from the process of decoding the solutions. In order to partially solve these concerns, Lourenço et al. proposed Structured Grammatical Evolution[17].

In comparison to Grammatical Evolution, where individuals are made up of a list of numbers of a set length and each number is used to generate the next symbol of the phenotype, in Structured Grammatical Evolution, individuals are made up of a list of lists.

Each internal list represents a non-terminal of the grammar and the numbers contained are the possible expansions of this rule for a certain individual in our population, therefore each internal list's length can be at most the maximum number of references to that rule. Each of the numbers contained in the internal list represents a production for the rule in question, therefore the numbers can go from 0 to C_{n-1} with C_n equal to the maximum amount of productions of a non-terminal, as shown in Figure 3.1.



<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::= (<var> <op> <var>) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
Posible expansions for each non-terminal
<start> = [0,1]
<expr> = [0]
<term> = [0,1]
<var> = [0,1,2,3,4]
<op> = [0,1,2]

Figure 3.1: *Possible alleles for each rule*

When decoding an individual we will have a list of indexes, instead of just one as we had in Grammatical Evolution (GE). Every time we have to decode a non-terminal, we will go to the list of productions for this non-terminal and get the next element of the list we have not consumed up to that point, given to us by the index value of that list of non-terminals, as shown in Figure 3.2.

With this, the decoding process will be: If the next symbol is a terminal, we add it to our phenotype and continue with the next element. If the next symbol is a non-terminal then we get the corresponding production based on what the individual's genotype states and repeat the process for all the symbols of the production until we don't have more symbols.

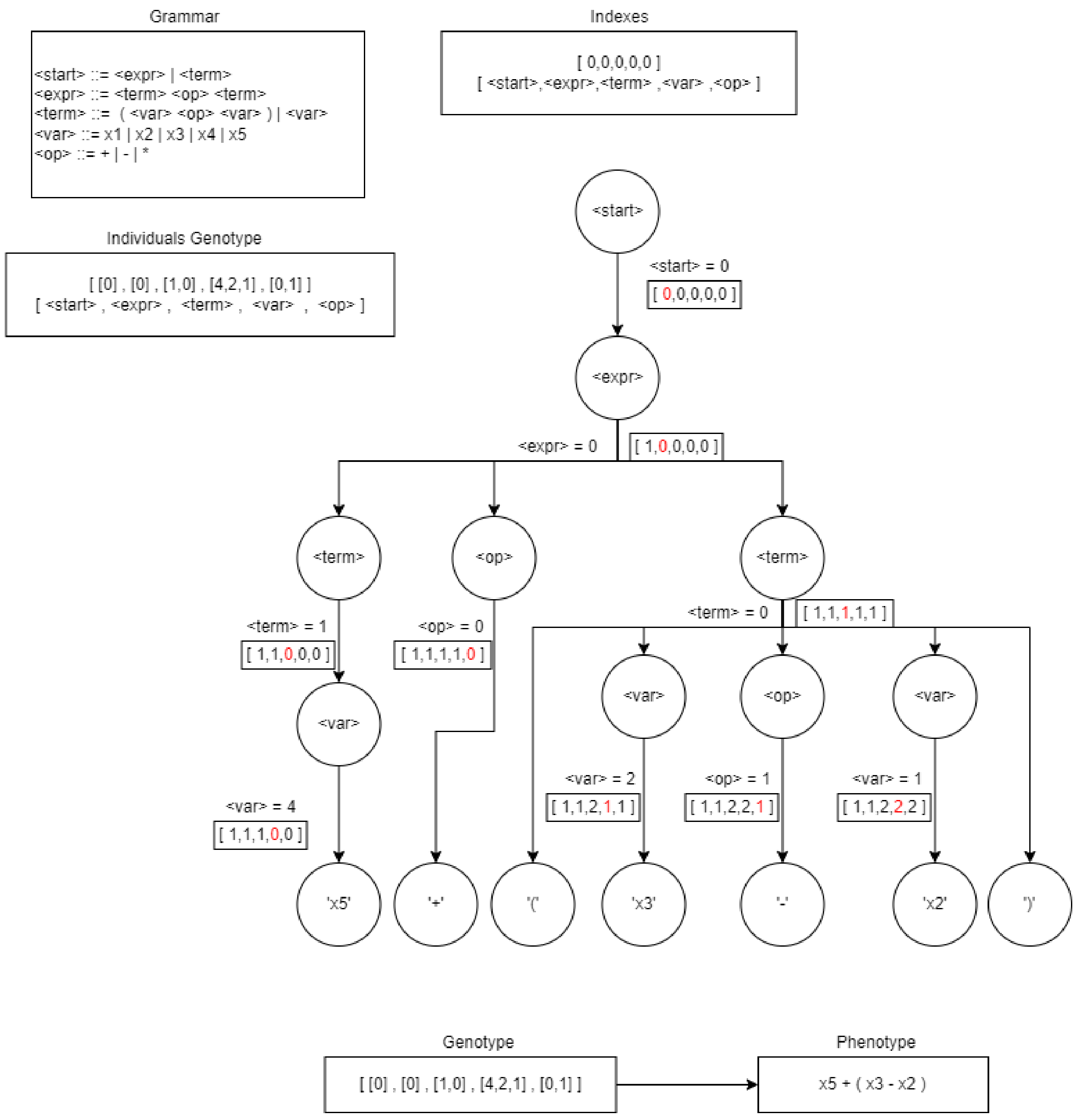


Figure 3.2: *Genotype to phenotype*

3.2.1 Static Structured Grammatical Evolution

The original version of Structured Grammatical Evolution uses static internal lists, whose lengths will not vary during the evolution, to represent the different rules. We have called the original version Static Structured Grammatical Evolution or Static SGE to differentiate it from the dynamic version and highlight its main difference.

To ensure the precondition of having static lists each internal list is generated completely to its maximum possible length, and the maximum number of references for each non-terminal element of the grammar, even if said expansions are not being used by an individual.

In Figure 3.3 can be seen an example of this process, for each non-terminal of the grammar we need to compute the upper bound of times that a certain rule can be called from another one. The first rule will always be called once, as is it the entry point of the phenotype construction.

This mechanism of counting references forces us to eliminate recursion from the grammar file since having recursion in our grammar means that there could exist an infinite number of references for the recursive rule. We have performed this by setting a maximum recursive depth, defined by the user, and transforming the recursive rule into a set of rules that mimic the recursion up to the set depth, with depth = 0 equivalent to eliminating the recursive production and only keeping the non-recursive ones.

```
<expr> ::= <expr> <op> <expr>
```

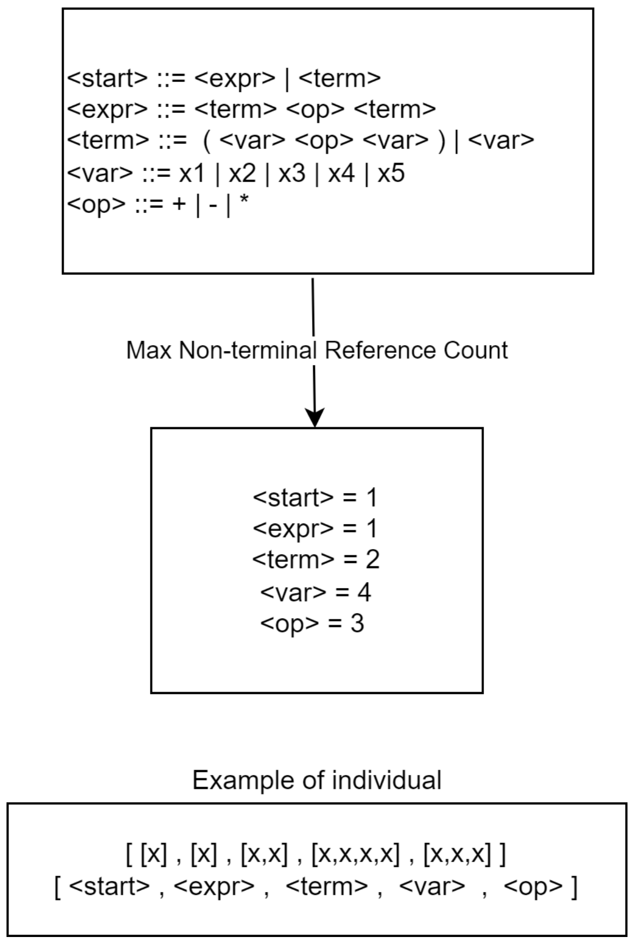
As an example, for depth 3 this rule will be transformed into:

```
<expr>  ::= <expr0> <op> <expr0> | <term> <op> <term>

<expr0> ::= <expr1> <op> <expr1> | <term> <op> <term>

<expr1> ::= <expr2> <op> <expr2> | <term> <op> <term>

<expr2> ::= <term> <op> <term>
```



```
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::=  ( <var> <op> <var> ) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
```

Max Non-terminal Reference Count
<start> = 1
<expr> = 1
<term> = 2
<var> = 4
<op> = 3
Example of individual
[[x] , [x] , [x,x] , [x,x,x,x] , [x,x,x]]
[<start> , <expr> , <term> , <var> , <op>]

Figure 3.3: *Counting the maximum amount of references for a non-terminal*

We change the original recursive rule to call to the newly created non-recursive rules, each depth number calls to the next one. The final one will eliminate all the recursive productions and only keep the ones that have non-recursive symbols, the newly made rules will be added to the loaded grammar and afterward, the references are counted.

Considerations of Static SGE

In static SGE the individual's genotype will be very long since we have to generate the upper bound of possible alleles generated. As a result, the more recursive depth is added by the user the longer the individual, as we will have to add as many extra lists as non-terminals generated through the depth, each one with at least 1 element. This will affect the performance of the evolution over them, making Static SGE slower than the dynamic version.

Generating non-recursive rules in this fashion will change the structure of the final individual in comparison to the Dynamic version, this affects both mutation and crossover, making them not equivalent. In the case of crossover, it increments the number of elements that can be interchanged between the individuals and also allows a more specific exchange, although there is a higher chance that the non-terminal list exchanged won't be used to generate the phenotype. In the case of the mutation it only affects if we are performing a mutation on the level of the lists and not the alleles, because we have more lists we can mutate (because we have more non-terminals) we will mutate more alleles per individual.

3.2.2 Dynamic Structured Grammatical Evolution

The original Structured Grammatical Evolution or Static SGE is able to solve the main issues with Grammatical Evolution, although it adds a new one due to the need for grammar prepossessing and deleting recursion. Making our solutions to be of the maximum length ensures that we will never go over the set bound, however, this length will be unnecessary in most cases, as most solutions will not need the maximum length. Paired with recursion, it becomes very unscalable. As a result, a dynamic approach to the list generation was implemented called Dynamic Structured Grammatical Evolution that would fix this issue by making the genotype able to grow dynamically.

In Dynamic SGE the lists are generated dynamically, meaning, when the individuals are generated we generate the lists up to what is needed to complete the individual's phenotype (all the generated individuals must be valid from creation). This adds a level of complexity in comparison to the Static version as an individual might become invalid after we have performed crossover and mutation over them. To avoid generating invalid individuals when the phenotype is being computed we can add extra alleles to a certain non-terminal list as needed to complete the individual phenotype and be able to calculate its fitness.

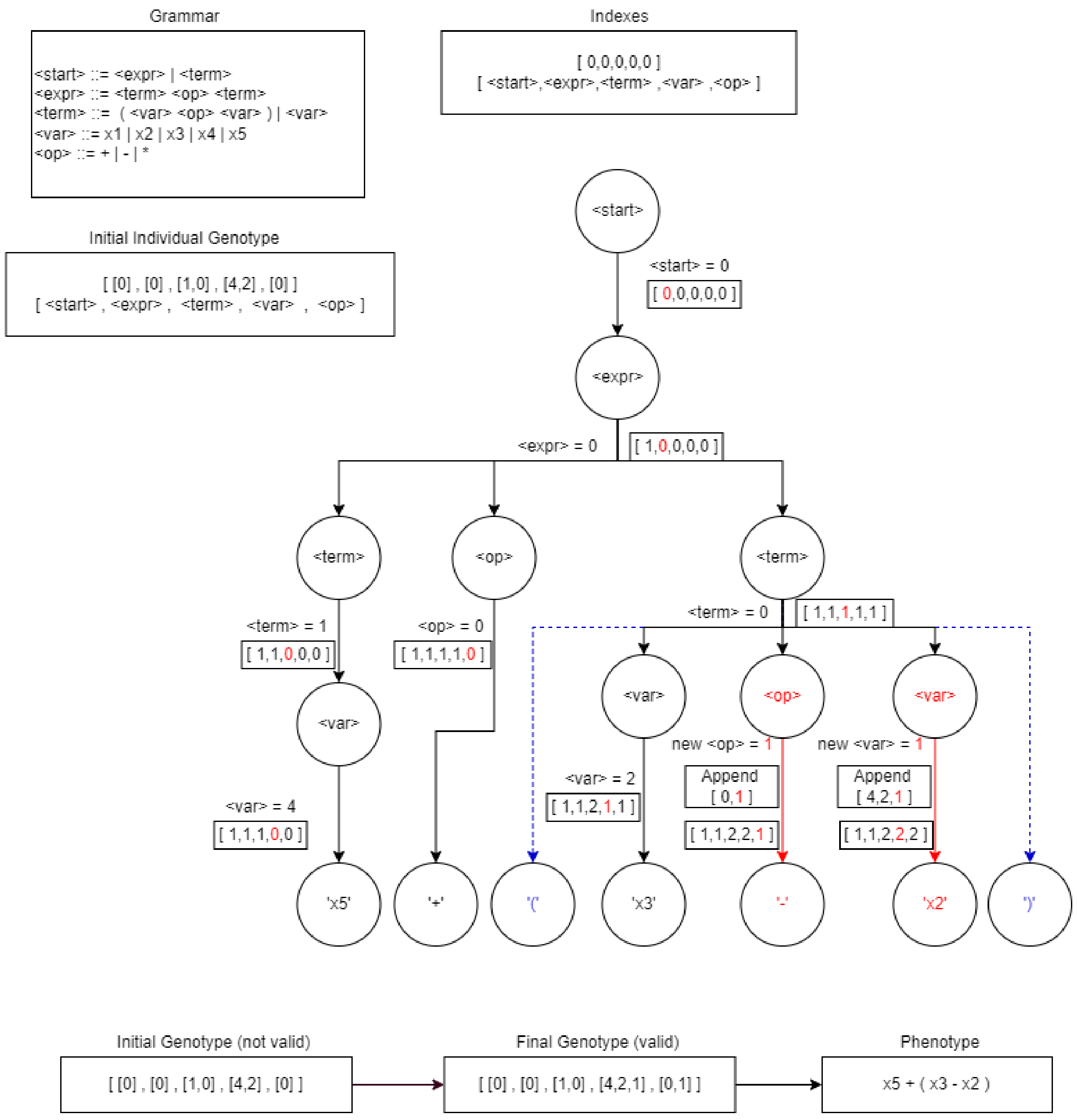


Figure 3.4: *Genotype to phenotype in DSGE when individual is invalid*

As seen in Fig. 3.4, the individual after crossover and mutation is invalid, therefore when generating the phenotype as we get to the second $< op >$ obtained from expanding $< term >$ there is no possible element to expand, we will have to generate a random allele, in the example case, we generate allele 1 and append it to the end of the list of $< op >$.

The same thing happens in the next non-terminal $< var >$ for which we also don't have an expansion, we generate a new value and append it. At the end of the genotype to phenotype conversion, the individual has changed to become valid. The index computing remains the same as in Static SGE.

As with the static version recursion poses an issue since we could potentially create an individual that never completes itself, to stop this from happening and from having individuals that are extremely big, we add a maximum depth. In the original implementation,[16], the maximum depth was implemented through maximum tree depth, which limited how long an individual can be. It was enforced only when generating extra alleles in the phenotype creation, if the depth of the tree became bigger or equal to the number set by the user the algorithm could only generate non-recursive expansions and not recursive ones. Using this implementation a branch of the tree might become longer than the maximum tree depth, since we have to generate valid individuals, however, as a result of not generating recursive rules the maximum length is still limited.

In Fig. 3.5 there is a graphical representation of how the maximum depth is enforced in the original implementation. We have changed the grammar so that the $< expr >$ rule is now recursive. When we reach the second $< expr >$ production, we don't have any more alleles, so we need to dynamically generate them to complete the individual. Since the current depth of this branch is equal to the maximum depth set, we only have one option in the possible productions we can generate: $< term >$ as it is the only non-recursive production of the rule.

The fact that a branch can become longer than the maximum depth makes this implementation dependent on the grammar structure, if the recursive rules appear further down in the grammar tree it will have less probability of performing recursion than up the grammar tree. Due to this, we have chosen to program another implementation that makes it equivalent to the static SGE.

In this alternate implementation, we have changed the maximum tree depth by a recur-

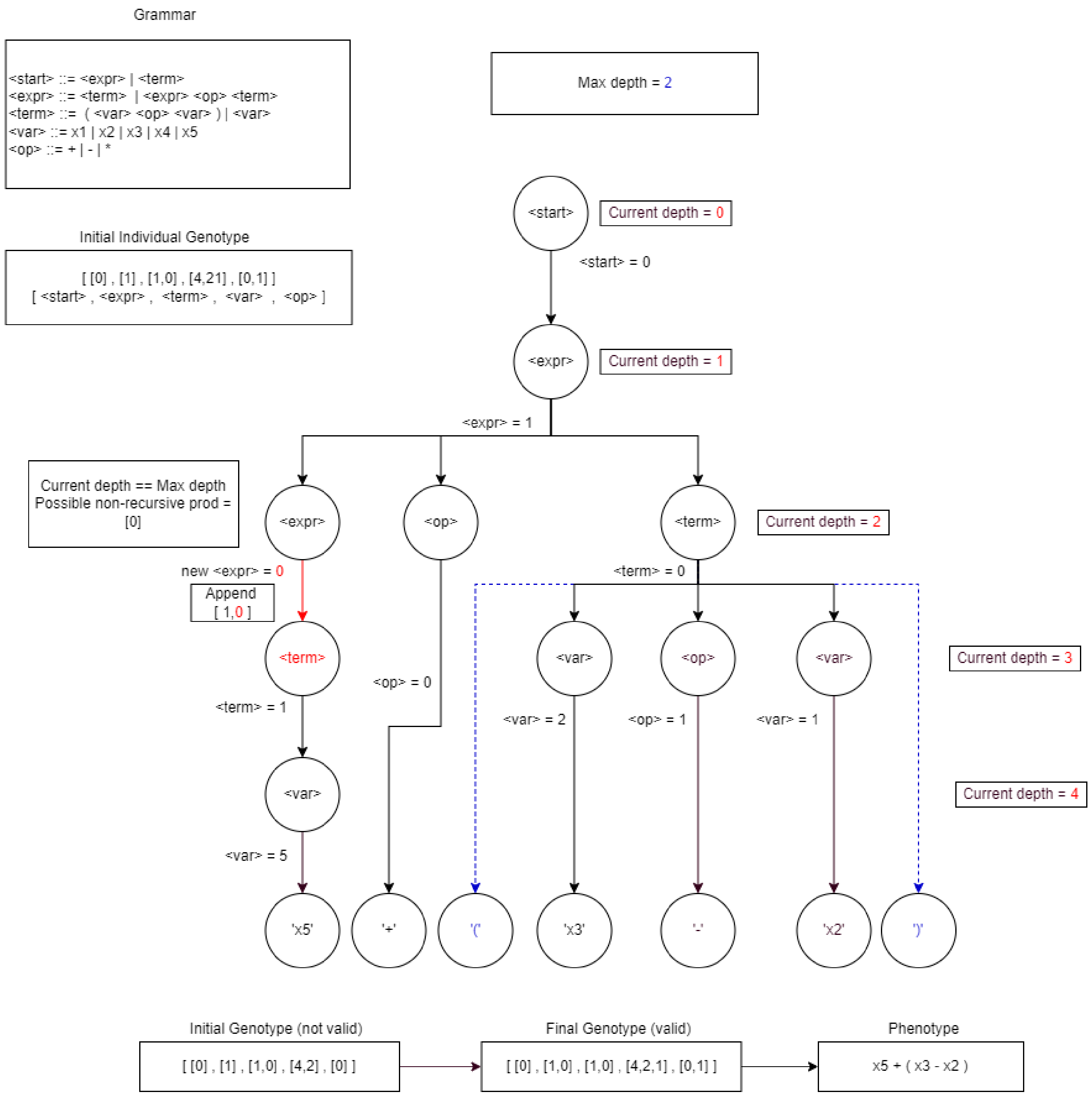


Figure 3.5: *Genotype to phenotype in DSGE showing limited recursive depth*

sive depth just as what we had in the Static version, each recursive rule can be expanded n times, completely independent of the non-recursive rules and the rest of the grammar structure.

Another change we have added is an option for more bloating control of the individuals.

In the original version, the tree depth is only enforced when dealing with recursive rules and when we are generating extra alleles, as by definition the individuals have to be valid. This process is performed in the generation of the phenotype, since neither the crossover nor mutation operators have any information about the individual's state, and it would be very costly to compute the phenotype every time a change is performed in the genotype. In theory, we could expand the individuals by generating recursive expansion through mutation and crossover and then generate non-terminals in the phenotype, to control this random growth so that it stays within the limits we have set, we will change the recursive alleles that go over the maximum length to non-recursive ones.

The bloating control can be activated or deactivated through a parameter, although in general, it's better to consider it as we make the individuals smaller and the search space more manageable.

Optional Parameters

Additionally, we also have two other optional parameters that control the initial creation process of the population. We have spoken of the maximum tree/recursive depth that is enforced when computing the phenotype, this depth must also be added to the initial population to ensure that tree growth is controlled, this value can be the same as the normal tree depth (if not assigned), or a different value assigned by the programmer.

Similarly, the minimum depth can also be assigned, to ensure that the initial population has a decent size. In this case, the depth is a recursive depth, so if a recursive rule is generated then this rule will be expanded a minimum of n times. We have set it this way because we can't possibly know how many recursive rules an individual is going to generate, but we can ensure that if it generates one, then it will generate other recursive productions. If the value is not assigned by the programmer, it is set to 0 by default. The main positive to having an initial population of minimum size is that a minimum number of alleles will initially go through the evolutionary process, if we don't set it then the evolution can take longer because the alleles will need to be generated dynamically after evolution in the phenotype creation.

In terms of preferences, the maximum initial depth is always enforced first, so if the current depth of the individual is bigger than the maximum initial depth we do not generate recursive expansions even if the minimum initial recursive depth is less than the one set by the programmer.

Considerations of Dynamic SGE

The individuals are generally simpler than in the original SGE, because we are not computing the maximum length of the individual, the average length will be much smaller. We don't need to perform any pre-processing on the grammar to determine the maximum length of the lists, nor edit the grammar to eliminate recursion. We have fewer lists to perform crossover over and alleles to mutate and the performance of these operations will have more effect on the final phenotype.

However, it has more complexity in the creation of the phenotype and the individuals can suffer from bloating contrary to the static version because they grow dynamically. To prevent bloating from occurring, we have to add bloating control to the individuals, that we didn't need in the original SGE.

3.2.3 Depth

As stated for both algorithms we have to set a maximum depth that stops them from growing too large, this value is defined by the user. The larger the value, the more freedom we give to the models (phenotypes) to grow as needed, but also the larger the search space will become. This means that the evolution can get lost easier and more generations will need to take place for the problem to converge to an optimum value. The performance will also be affected by this value, especially in the static version of SGE where the genotype grows exponentially. The longer, the individuals become, the more time it will take for the evolution to be performed. Therefore, when possible, smaller depth numbers should be considered as long as they allow an optimal solution to be found.

3.3 Operators

3.3.1 Mutation operator

We have programmed and tested two different mutation operations, both of which consist of changing the value of one of the alleles by another value inside the range of possible values (number of productions) randomly. We have set it so that the change can only be performed on alleles that have more than one possible value and if the change is performed the resulting allele will always have a different value than the one it had previously. this way we fix the issue from Grammatical Evolution where even if an allele mutates, the resulting phenotype could end up being the same as before the change, due to having to perform the mod operator.

Both operators have a version that works with static arrays for the Static SGE and dynamic Lists for the Dynamic SGE, but the process is the same.

Integer Flip Mutation on the list

The first mutation operator we have is called IntegerFlipMutationList in SSGE and Basic-MutationVariableList in DSGE, is the one recommended in the literature. In it, we perform a change of the individual on the level of the inside lists, which corresponds to a non-terminal production from the grammar. For each non-terminal, we can mutate at most one allele chosen randomly from all of the ones in the list. This change is performed for all the internal lists in the individual if the mutation probability is accepted. This process is shown in Fig. 3.6

Integer Flip Mutation on all alleles

The second mutation operator implemented we have called IntegerFlipMutationListAll in SSGE and BasicMutationVariableList in DSGE. The mutation is performed for each individual allele, independently of whether the allele belongs to one list or another. This mutation would be equivalent to the IntegerFlipMutation of Grammatical Evolution, except that we

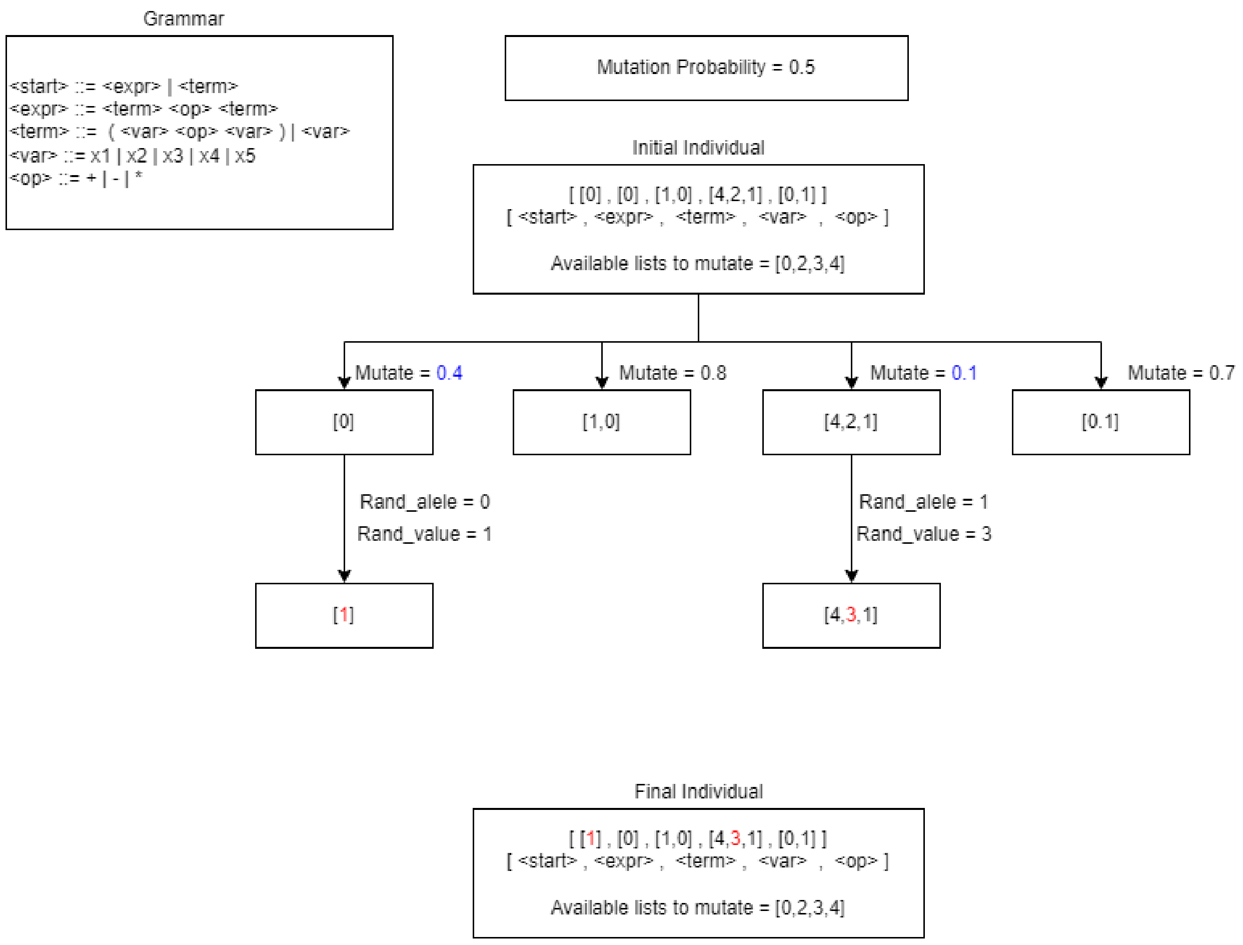

Grammar
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::= (<var> <op> <var>) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
Mutation Probability = 0.5
Initial Individual
[[0] , [0] , [1,0] , [4,2,1] , [0,1]]
[<start> , <expr> , <term> , <var> , <op>]
Available lists to mutate = [0,2,3,4]
Mutate = 0.4
Mutate = 0.8
Mutate = 0.1
Mutate = 0.7
[0]
[1,0]
[4,2,1]
[0,1]
Rand_alele = 0
Rand_value = 1
Rand_alele = 1
Rand_value = 3
[1]
[4,3,1]
Final Individual
[[1] , [0] , [1,0] , [4,3,1] , [0,1]]
[<start> , <expr> , <term> , <var> , <op>]
Available lists to mutate = [0,2,3,4]

Figure 3.6: *Mutation Diagram for IntegerFlipMutationList*

have to consider the range that the alleles can take on each list to make the mutation. This process is shown in Fig. 3.7

3.3.2 Crossover operator

We have programmed and tested two different crossover operators, both of which are performed at the level of the internal lists. The internal values are not edited, the information remains unchanged, we only perform an interchange of the lists for a non-terminal between two individuals. The crossover operators can be used by both the Static SGE and Dynamic SGE.

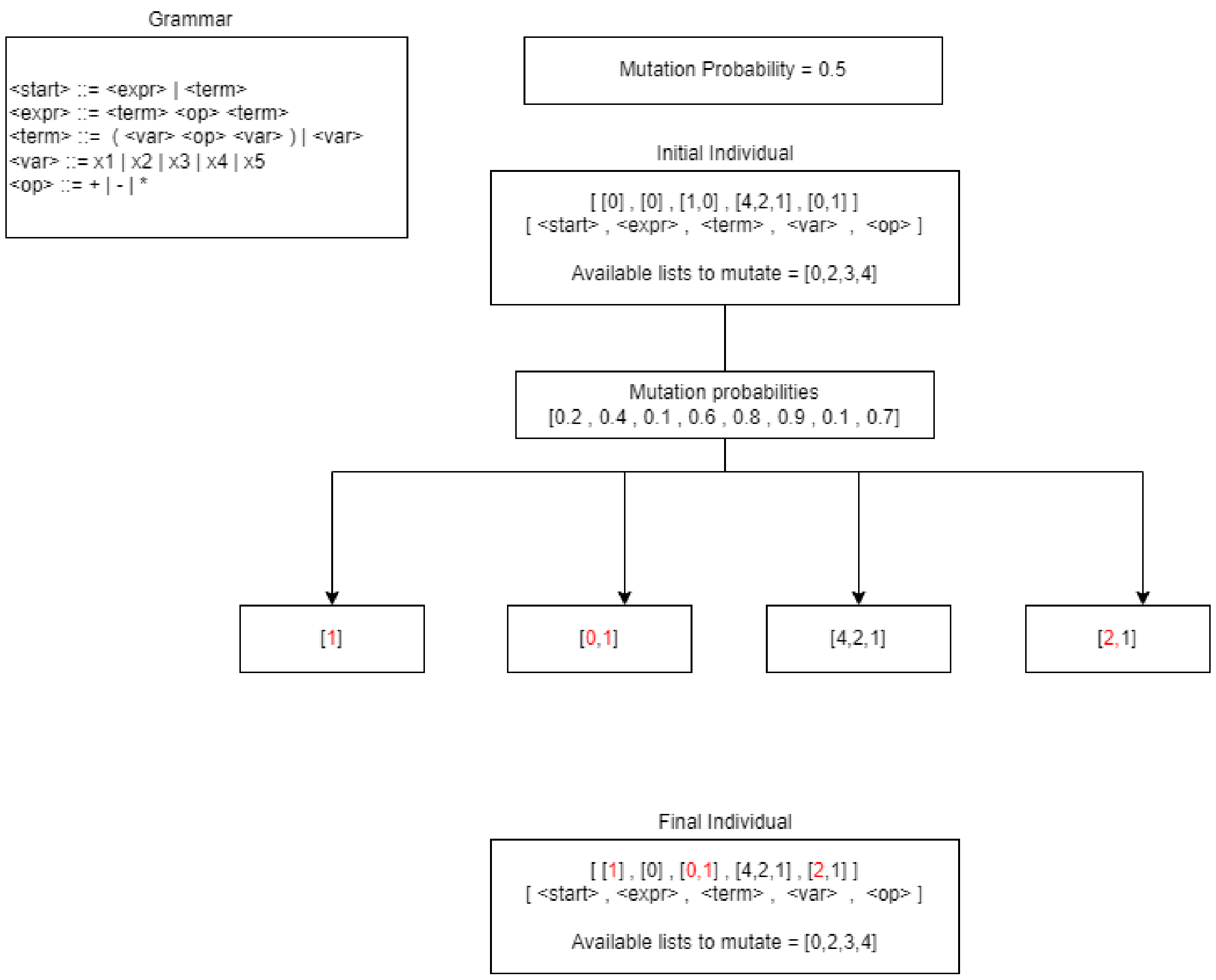

Grammar
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::= (<var> <op> <var>) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
Mutation Probability = 0.5
Initial Individual
[[0] , [0] , [1,0] , [4,2,1] , [0,1]]
[<start> , <expr> , <term> , <var> , <op>]
Available lists to mutate = [0,2,3,4]
Mutation probabilities
[0.2 , 0.4 , 0.1 , 0.6 , 0.8 , 0.9 , 0.1 , 0.7]
[1]
[0,1]
[4,2,1]
[2,1]
Final Individual
[[1] , [0] , [0,1] , [4,2,1] , [2,1]]
[<start> , <expr> , <term> , <var> , <op>]
Available lists to mutate = [0,2,3,4]

Figure 3.7: *Mutation Diagram for IntegerFlipMutationListAll*

Uniform Crossover

This is the crossover operator defined in the literature[17]. For each internal list representing a non-terminal, we perform an interchange of the lists of both individuals if it enters a certain probability of change. This interchange probability can be defined by the user. The offspring will be the result of a random exchange of non-terminal lists from their two parents. This process is shown in Fig. 3.8

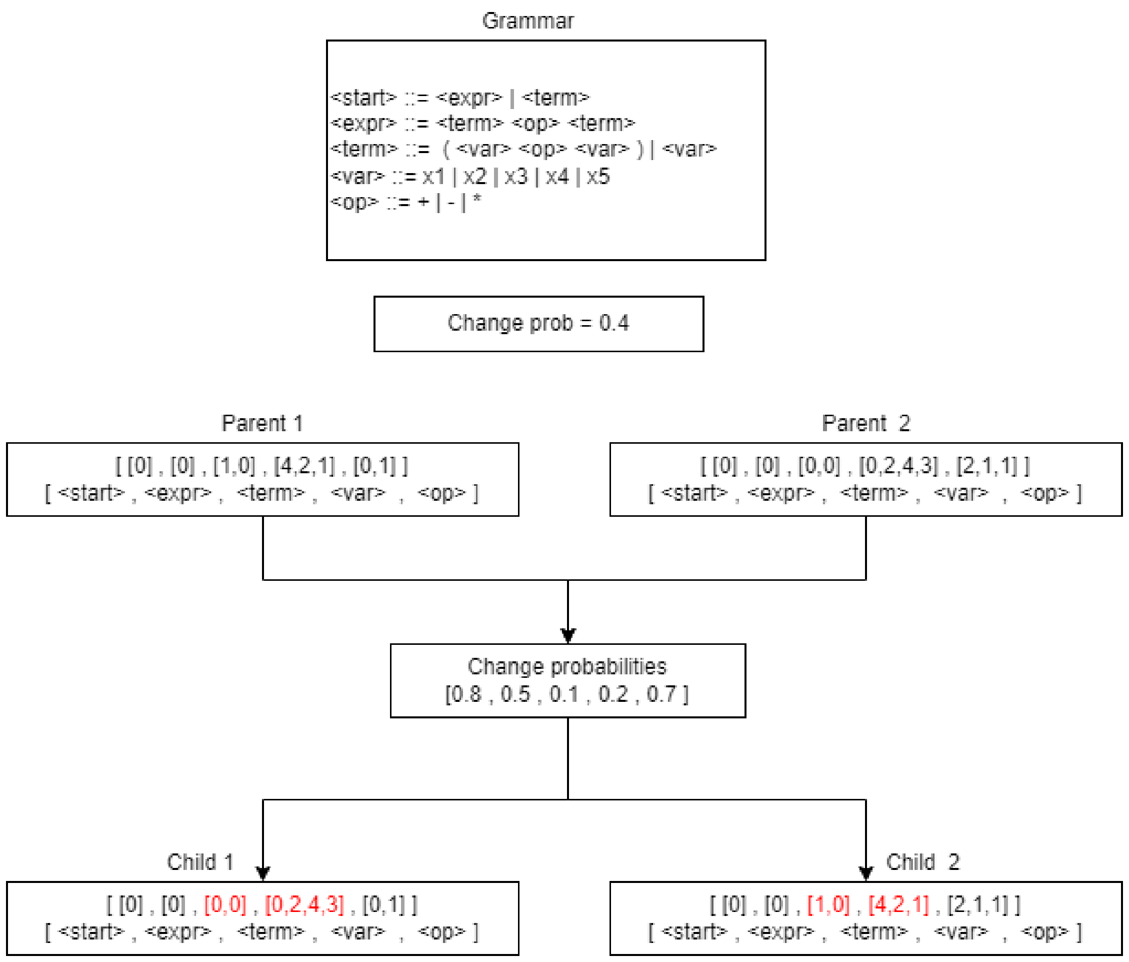

Grammar
<start> ::= <expr> | <term>
<expr> ::= <term> <op> <term>
<term> ::= (<var> <op> <var>) | <var>
<var> ::= x1 | x2 | x3 | x4 | x5
<op> ::= + | - | *
Change prob = 0.4
Parent 1
[[0] , [0] , [1,0] , [4,2,1] , [0,1]]
[<start> , <expr> , <term> , <var> , <op>]
Parent 2
[[0] , [0] , [0,0] , [0,2,4,3] , [2,1,1]]
[<start> , <expr> , <term> , <var> , <op>]
Change probabilities
[0.8 , 0.5 , 0.1 , 0.2 , 0.7]
Child 1
[[0] , [0] , [0,0] , [0,2,4,3] , [0,1]]
[<start> , <expr> , <term> , <var> , <op>]
Child 2
[[0] , [0] , [1,0] , [4,2,1] , [2,1,1]]
[<start> , <expr> , <term> , <var> , <op>]

Figure 3.8: *Uniform Crossover Diagram*

Tree Crossover

This operator has been made as an alteration of the Uniform crossover, it takes into account the structure of the grammar to perform the exchange between the lists, as such the resulting children's phenotype will be more similar to the parents, than what we obtain with a random list interchange.

First, it chooses a non-terminal production and interchanges those two lists in the children, then it gets all the lists of non-terminals that are produced from a certain rule and interchanges those with a certain probability, similar to the one we had in the Uniform crossover. We repeat the process for the non-terminal lists that we have interchanged until we reach a terminal, or we don't change any more lists. This process is shown in Fig. 3.9

It works a bit better than the Uniform crossover in some cases, especially for the static list structured grammatical evolution, where adding recursion generates new rules to the grammar. When performing a random interchange of lists, the children will look less than the parents the more depth added in creation, because more intermediate non-terminals can be exchanged. It maintains the structure of the parents better in the children, adding less variation to the offspring.

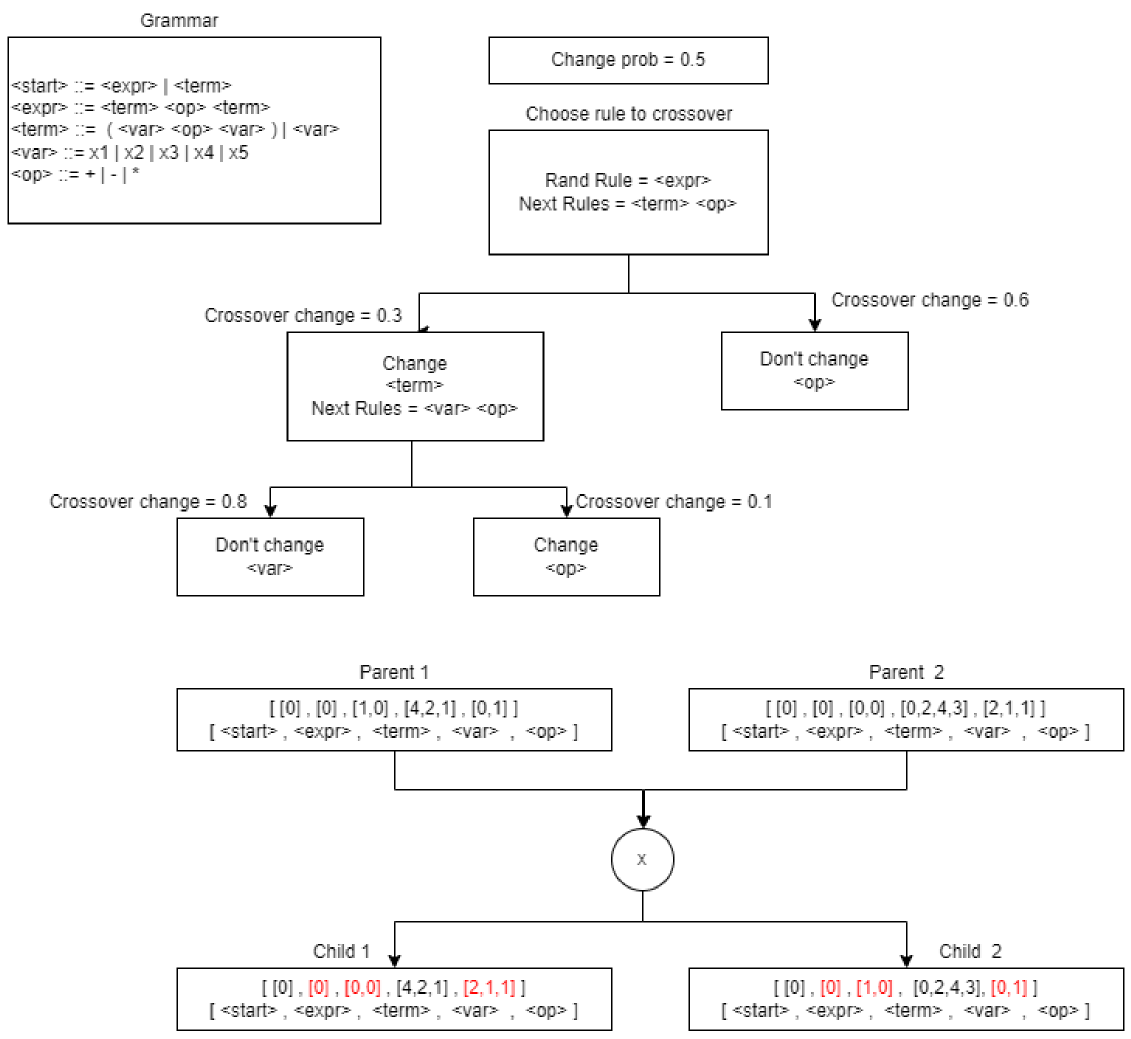


Figure 3.9: *Tree Crossover Diagram*

3.4 Grammar

As we have stated before, the genotype to phenotype conversion is performed by means of a grammar file that determines the structure of the phenotype of the individual. In SGE, compared to normal GE, the grammar structure also affects the individual internally.

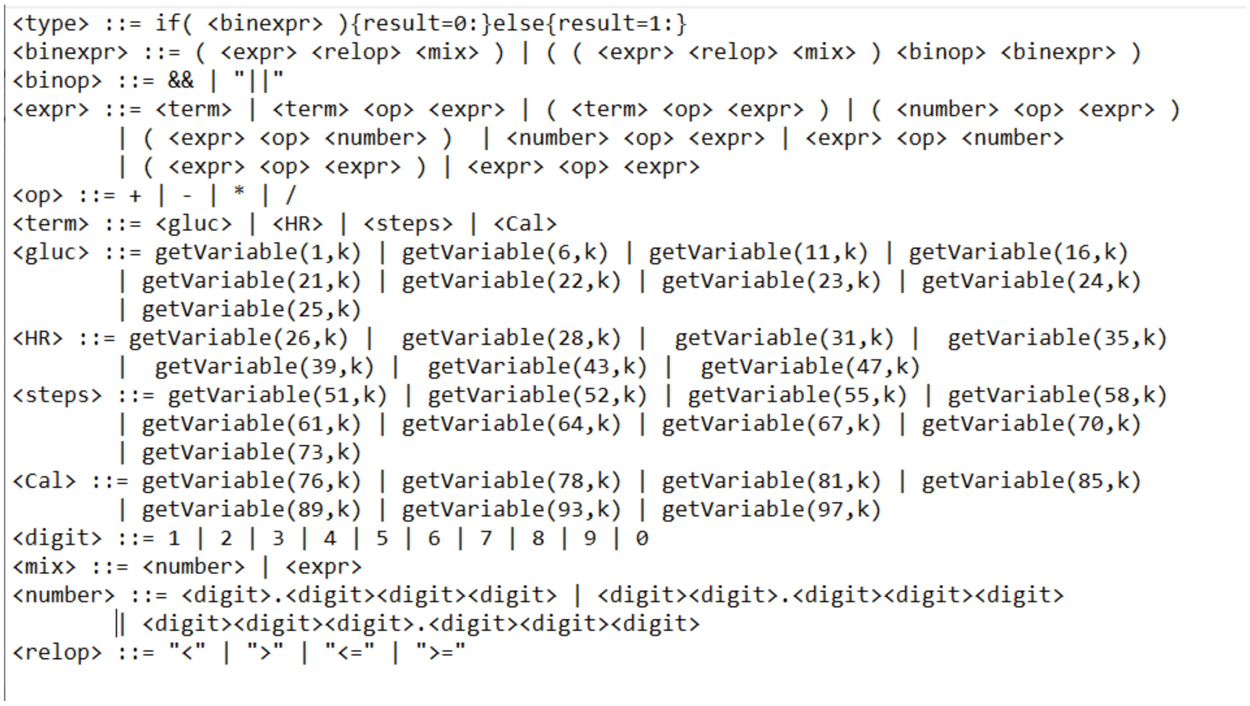
The grammar will be the one that states the structure of our individuals in terms of the number of lists, which corresponds to the number of rules of the grammar, and the maximum length that each list can have, which corresponds to the number of references made to a certain rule or non-terminal expansion. The classification will also be performed through the grammar file in the genotype to phenotype conversion, it will allow us to change the number of classes just by editing the grammar file.

For our problem, the grammar will be used to transform the list of lists into a sequence of if-then-else statements that return a class. The classes have to also be defined in the input data to compare the result from the if-then-else to the actual value of the input example.

The if-then-else statements use the input data and some constants to create conditions, if all the conditions are met then we consider that the example belongs to that specific class. Since we are returning the statements as our models, the conditions can be studied to observe what input parameters the model has considered over others. This white-box approach makes it so that we can give a medical explanation to the results, or discard a model that we consider is not correct using knowledge from the problem specification.

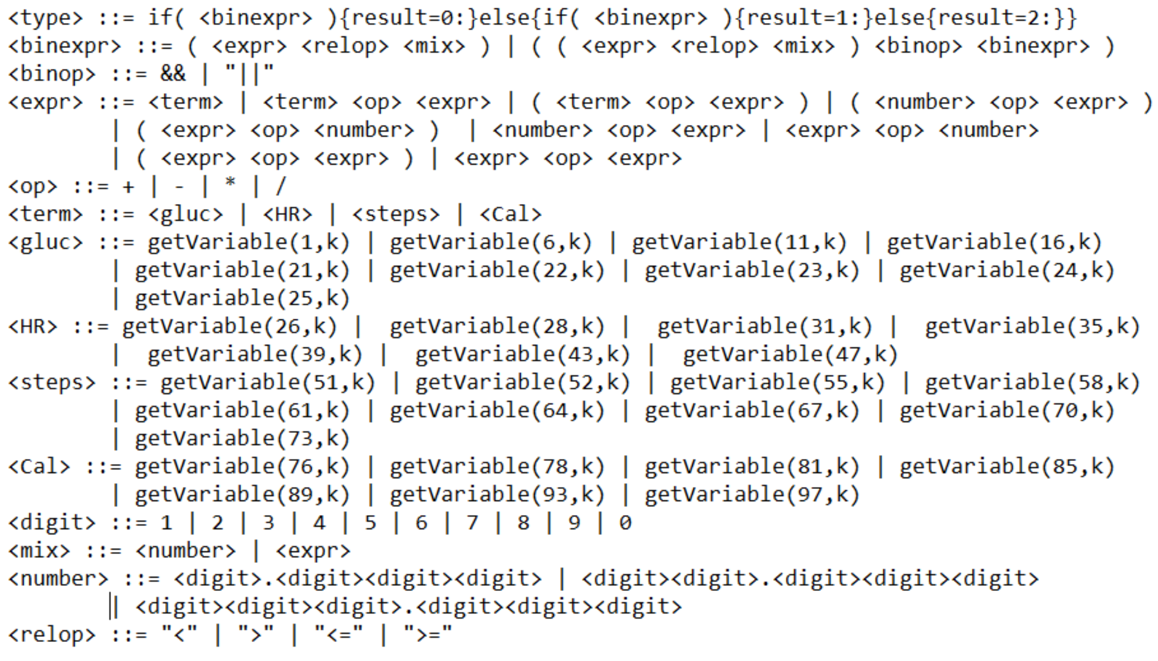
We have tested two different types of grammars depending on the number of classes we have to classify, one for 2 classes 3.10 and another for 3 classes 3.11, their structure is the same and their only difference is the rule $< type >$ that generates a construction for 2 classes or one for 3 classes.

In terms of the structure of the conditions that they can generate they are as follows: The $< type >$ rule is the entrance of the program and the one that will wield the final result, it generates $< binexpr >$ rules that create the logical conditions. The $< binexpr >$



```
<type> ::= if( <binexpr> ){result=0:}else{result=1:}
<binexpr> ::= ( <expr> <relop> <mix> ) | ( ( <expr> <relop> <mix> ) <binop> <binexpr> )
<binop> ::= && | "||"
<expr> ::= <term> | <term> <op> <expr> | ( <term> <op> <expr> ) | ( <number> <op> <expr> )
        | ( <expr> <op> <number> )  | <number> <op> <expr> | <expr> <op> <number>
        | ( <expr> <op> <expr> ) | <expr> <op> <expr>
<op> ::= + | - | * | /
<term> ::= <gluc> | <HR> | <steps> | <Cal>
<gluc> ::= getVariable(1,k) | getVariable(6,k) | getVariable(11,k) | getVariable(16,k)
        | getVariable(21,k) | getVariable(22,k) | getVariable(23,k) | getVariable(24,k)
        | getVariable(25,k)
<HR> ::= getVariable(26,k) |  getVariable(28,k) |  getVariable(31,k) |  getVariable(35,k)
        |  getVariable(39,k) |  getVariable(43,k) |  getVariable(47,k)
<steps> ::= getVariable(51,k) | getVariable(52,k) | getVariable(55,k) | getVariable(58,k)
        | getVariable(61,k) | getVariable(64,k) | getVariable(67,k) | getVariable(70,k)
        | getVariable(73,k)
<Cal> ::= getVariable(76,k) | getVariable(78,k) | getVariable(81,k) | getVariable(85,k)
        | getVariable(89,k) | getVariable(93,k) | getVariable(97,k)
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<mix> ::= <number> | <expr>
<number> ::= <digit>.<digit><digit><digit> | <digit><digit>.<digit><digit><digit>
        || <digit><digit><digit>.<digit><digit><digit>
<relop> ::= "<" | ">" | "<=" | ">="
```

Figure 3.10: *Example Grammar 2 Classes*



```
<type> ::= if( <binexpr> ){result=0:}else{if( <binexpr> ){result=1:}else{result=2:}}
<binexpr> ::= ( <expr> <relop> <mix> ) | ( ( <expr> <relop> <mix> ) <binop> <binexpr> )
<binop> ::= && | "||"
<expr> ::= <term> | <term> <op> <expr> | ( <term> <op> <expr> ) | ( <number> <op> <expr> )
        | ( <expr> <op> <number> )  | <number> <op> <expr> | <expr> <op> <number>
        | ( <expr> <op> <expr> ) | <expr> <op> <expr>
<op> ::= + | - | * | /
<term> ::= <gluc> | <HR> | <steps> | <Cal>
<gluc> ::= getVariable(1,k) | getVariable(6,k) | getVariable(11,k) | getVariable(16,k)
        | getVariable(21,k) | getVariable(22,k) | getVariable(23,k) | getVariable(24,k)
        | getVariable(25,k)
<HR> ::= getVariable(26,k) |  getVariable(28,k) |  getVariable(31,k) |  getVariable(35,k)
        |  getVariable(39,k) |  getVariable(43,k) |  getVariable(47,k)
<steps> ::= getVariable(51,k) | getVariable(52,k) | getVariable(55,k) | getVariable(58,k)
        | getVariable(61,k) | getVariable(64,k) | getVariable(67,k) | getVariable(70,k)
        | getVariable(73,k)
<Cal> ::= getVariable(76,k) | getVariable(78,k) | getVariable(81,k) | getVariable(85,k)
        | getVariable(89,k) | getVariable(93,k) | getVariable(97,k)
<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<mix> ::= <number> | <expr>
<number> ::= <digit>.<digit><digit><digit> | <digit><digit>.<digit><digit><digit>
        || <digit><digit><digit>.<digit><digit><digit>
<relop> ::= "<" | ">" | "<=" | ">="
```

Figure 3.11: *Example Grammar 3 Classes*

rule is a recursive rule that creates a sequence of conditions fused by "&&" or "||" logical operators. The internal conditions are statements that can be evaluated and compared with each other using the relational operators "<", ">", "≥", "≤". The left-hand side of the statements has the $< expr >$ recursive rule that generates a mathematical operation made up of the input variables, that in our grammar appear as the "getVariable(x,k)" function, and constants that are generated with the $< number >$ rule. The right-hand side is the $< mix >$ rule that can either generate a $< expr >$ or a $< number >$.

Recursion appears in two different places in our grammar, so the max depth we set for each type of algorithm will control the number of conditions that are generated by the individual, by the $< binexpr >$ rule, and the length of the conditions, from the $< expr >$ rule.

The input variables, defined by the "getVariable(x,k)" function, represent each type of variable "x", for each example "k" in our input for training or test.

We will explain the input data structure in the Data section, but in general terms, the variables are defined as such:

- Glucose values of the two hours before the time of prediction t ($x \in (0, 25]$).
- Heart rate values of the patient of the two hours before the time of prediction t ($x \in (25, 50]$).
- Amount of steps performed by the patient on the two hours before the time of prediction t ($x \in (50, 75]$).
- Amount of calories burned by the patient in the two hours before the time of prediction t ($x \in (75, 100]$).

As can be observed in fig 3.10 and fig 3.11 we have not taken into account every single possible input to generate the models, we have handpicked a subset of them to construct the solutions, however, the available inputs that could be used by the grammar are the ones described above. This selection will be discussed in Chapter 4.

3.5 Fitness Function

As with any evolutionary algorithm, we need to establish a fitness function that will determine how good an individual is. Since we are considering a classification problem, we can test different classification metrics and compare their output with the results obtained from the test.

In our problem, the main metric we are going to consider when testing whether a solution is good or not is the sensitivity (or recall) of each class. This refers to the number of elements that have been classified as one class and actually belong to this class. This metric is the most important as we want to be able to recognize all instances of a potential Hypoglycemia over all the cases, more than we want that the ones that we have classified to be correct.

In binary classification we talk about sensitivity (or recall) and specificity since we only consider positives or negatives, these refer to the percentages of true-positives and true-negatives over the total number of positives or negatives in our data.

In multi-class classification, we can just look at the sensitivity of each class, which would be equivalent.

All the fitness functions considered will return a value between 0 and 1 where the higher the result the better the classification. Since we are minimizing in our evolutionary algorithm, we will use $1 - FitnessFunction$ in all cases so that the result that gets closer to 0 is the best result overall.

3.5.1 Multi-class Recall

This fitness function calculates the sensitivity of each class and makes the harmonic mean of all the classes. In binary classification, it would be equivalent to making 2*(sensitivity*specificity)/(sensitivity+specificity). By performing the harmonic mean of the recalls, we make sure that no one class has a much higher recall than the others.

$$Multi-classRecall = \frac{numClasses}{\sum_{i=1}^{numClases} \frac{1}{Recall_i}} \tag{3.1}$$

$$Recall = \frac{TruePositives}{TotalPositives} \tag{3.2}$$

Its main advantage is that it can work with unbalanced data sets because it doesn't matter how many examples we have of one class, each class counts the same as the rest.

3.5.2 $F1_{measure}$

This is a classification metric that makes the harmonic mean of the precision and recall of the class 0.

$$F1_{measure} = \frac{2 * Recall * Precision}{Recall + Precision} \tag{3.3}$$

where *Precision* corresponds to the fraction of elements that we are classifying correctly in the Hypoglycemia class from the the total amount we have classified in this class and *Recall* to the fraction of elements that we are classifying correctly from the total amount of elements in the Hypoglycemia class.

In a multi-class problem, we will calculate the $F1_{measure}$ measure for each class and return the average (macro-average).

3.5.3 Weighted Accuracy

This is a fitness formula proposed in[3] for Evolutionary algorithms in the context of classification. If combines the $F1_{measure}$ with accuracy in a 50% split.

Its main drawback is that because it uses accuracy, we won't be able to use it with unbalanced classes.

$$\mathrm{WA} = 0.5 * Accuracy + 0.5 * F1_{measure} \tag{3.4}$$

It gives good results, better than when we are only considering the F1-measure standalone, which is why we have used it to perform the tests.

3.6 Evolutionary Algorithm

Finally, we have implemented an evolutionary algorithm that performs the process of selection, crossover, and mutation, for each generation. These are made sequentially and are the standard process for a general genetic algorithm. We select 2 individuals using the selection method chosen, we perform crossover, then mutation, and then we add them to the list of children. The two main differences between our algorithm and a general evolutionary algorithm is the replacement policy and the regeneration of the population.

In terms of replacement, for each generation, we will get the best individuals between the parents and children, and those are the ones that will compose the population of the next generation. This makes the evolution a very elitist process and might end up in the population converging too quickly and not evolving further than a local optimum, in order to palliate this from happening we have added a regeneration of a section of the population. If the population fitness ever converges to a number a percentage of the population, set by the user, will be regenerated from scratch to add new genes to the evolution and hopefully make the problem get over the local optimum.

Chapter 4

Data

4.1 Introduction

As stated before the goal of our system is to be able to predict hypoglycemic episodes at a certain time, our data will only include glucose readings, number of steps, heart rate, and calories burned over the previous two hours from the time of prediction t measured in 5-minute intervals. The glucose readings have been obtained by means of a Continuous Glucose Monitor (CGM) FreeStyleLibre, and the rest of the data is obtained through a smartwatch Fitbit Ionic, making all the necessary data for our model obtainable without needing user input.

4.2 Description

The data used in this study has been obtained from 11 patients with type 1 diabetes, whose descriptions appear in 4.1. We will restrict the data to a maximum of 2 weeks worth of observations per patient.

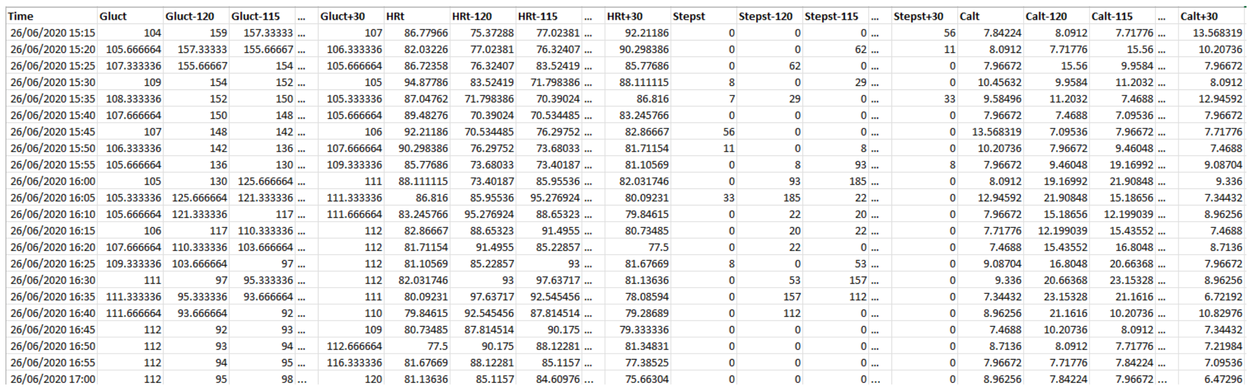
From the 11 patients, we have 4 males and 7 females, whose ages range from 20 years old to 56. They follow 2 different treatment methods, 3 patients control their blood sugar through multiple doses of insulin and the other 8 do it by means of an insulin pump that administers a continuous subcutaneous stream of insulin. Their HBA1c values represent the average value of blood sugar in the past 8 to 12 weeks and are used as a representation of the glycemic history[25]. The patient's values go from 6.4% to 8.5%, which is transformed to average glucose mg/dl would be from 137 mg/dL to 197 mg/dL on average. In general, a healthy range for a person with diabetes should be 7% or less, however, it's important to keep in mind that this is an average and a low value does not necessarily mean a good control, a person with a low or high HBA1c can still have frequent hypoglycemias and high glucose variability. In terms of their BMI 7 patients are within the normal range (18.5-24.9), 3 are in the overweight category (25-29.9) and one is in the obese range (30 or higher), this is important to take into account in terms of their blood glucose control because an excess in weight gain decreases insulin sensitivity and makes glucose control harder. Finally, we can observe that all patients have been diabetic for more than 8 years with the maximum being 36 years, this implies that their leftover insulin production is probably negligible.

ID	Gender	Age	IMC	HBA1c	Treatment	Years DMT1
HUPA001	F	56.3	22.76	8.2	ISCI	15.46
HUPA002	M	48.6	23.82	7.1	ISCI	36.47
HUPA003	F	43.4	18.72	7.3	ISCI	12.45
HUPA004	M	41.2	27.16	7.8	ISCI	8.5
HUPA005	F	20.9	22.57	6.9	ISCI	39.5
HUPA007	M	37.6	30.64	6.6	ISCI	10.1
HUPA011	F	35.0	23.92	7.8	ISCI	27.3
HUPA014	F	50.0	25.39	8.5	MDI	12.9
HUPA015	F	43.1	22.33	6.4	MDI	11.2
HUPA016	F	29.9	26.33	6.5	ISCI	20.1
HUPA027	M	26.4	22.21	7.0	MDI	23.7

Table 4.1: *Characteristics of the participants: ID, Gender (M=Male; F=Female), Age, BMI, HbA1c, Treatment (MDI: Multiples doses of insulin; ISCI. Infusion Subcutaneal continuous of Insulin)Years of evolution of DMT1.*

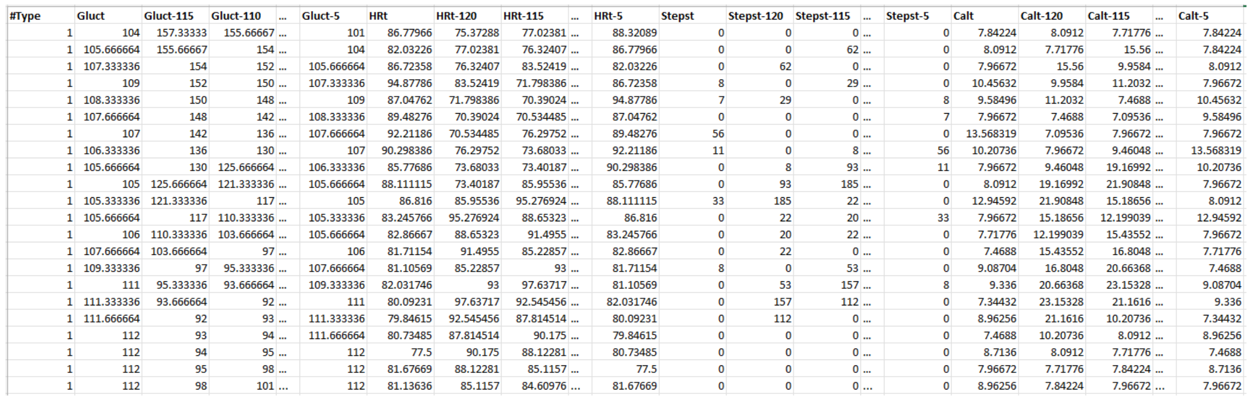
The input data obtained from the patients is structured as follows: Each row contains 101 columns, the first one is the glucose value at the time horizon $H \in \{30, 60, 90, 120\}$ followed by the glucose readings from the previous 2 hours, then the heart rate, followed by the steps and ending in the calories burned. Each one of these types has 25 readings, where the first value is always reading at time t and the next values are the readings from t-120 to t-5 in 5-minute intervals of each type of data.

In this way, for each data point we can access to all the available information by looking at the column values. We will have as many rows as data points for each patient.



Time	Gluct	Gluct-120	Gluct-115	...	Gluct+30	HRt	HRt-120	HRt-115	...	HRt+30	Stepst	Stepst-120	Stepst-115	...	Stepst+30	Calt	Calt-120	Calt-115	...	Calt+30
26/06/2020 15:15	104	159	157.33333	...	107	86.77966	75.37288	77.02381	...	92.21186	0	0	0	...	56	7.84224	8.0912	7.71776	...	13.568319
26/06/2020 15:20	105.666664	157.33333	155.66667	...	106.333336	82.03226	77.02381	76.32407	...	90.298386	0	0	62	...	11	8.0912	7.71776	15.56	...	10.20736
26/06/2020 15:25	107.333336	155.66667	154	...	105.666664	86.72358	76.32407	83.52419	...	85.77686	0	62	0	...	0	7.96672	15.56	9.9584	...	7.96672
26/06/2020 15:30	109	154	152	...	105	94.87786	83.52419	71.798386	...	88.111115	8	0	29	...	0	10.45632	9.9584	11.2032	...	8.0912
26/06/2020 15:35	108.333336	152	150	...	105.333336	87.04762	71.798386	70.39024	...	86.816	7	29	0	...	33	9.58496	11.2032	7.4688	...	12.94592
26/06/2020 15:40	107.666664	150	148	...	105.666664	89.48276	70.39024	70.534485	...	83.245766	0	0	0	...	0	7.96672	7.4688	7.09536	...	7.96672
26/06/2020 15:45	107	148	142	...	106	92.21186	70.534485	76.29752	...	82.86667	56	0	0	...	0	13.568319	7.09536	7.96672	...	7.71776
26/06/2020 15:50	106.333336	142	136	...	107.666664	90.298386	76.29752	73.68033	...	81.71154	11	0	8	...	0	10.20736	7.96672	9.46048	...	7.4688
26/06/2020 15:55	105.666664	136	130	...	109.333336	85.77686	73.68033	73.40187	...	81.10569	0	8	93	...	8	7.96672	9.46048	19.16992	...	9.08704
26/06/2020 16:00	105	130	125.666664	...	111	88.111115	73.40187	85.95536	...	82.031746	0	93	185	...	0	8.0912	19.16992	21.90848	...	9.336
26/06/2020 16:05	105.333336	125.666664	121.333336	...	111.333336	86.816	85.95536	95.276924	...	80.09231	33	185	22	...	0	12.94592	21.90848	15.18656	...	7.34432
26/06/2020 16:10	105.666664	121.333336	117	...	111.666664	83.245766	95.276924	88.65323	...	79.84615	0	22	20	...	0	7.96672	15.18656	12.199039	...	8.96256
26/06/2020 16:15	106	117	110.333336	...	112	82.86667	88.65323	91.4955	...	80.73485	0	20	22	...	0	7.71776	12.199039	15.43552	...	7.4688
26/06/2020 16:20	107.666664	110.333336	103.666664	...	112	81.71154	91.4955	85.22857	...	77.5	0	22	0	...	0	7.4688	15.43552	16.8048	...	8.7136
26/06/2020 16:25	109.333336	103.666664	97	...	112	81.10569	85.22857	93	...	81.67669	8	0	53	...	0	9.08704	16.8048	20.66368	...	7.96672
26/06/2020 16:30	111	97	95.333336	...	112	82.031746	93	97.63717	...	81.13636	0	53	157	...	0	9.336	20.66368	23.15328	...	8.96256
26/06/2020 16:35	111.333336	95.333336	93.666664	...	111	80.09231	97.63717	92.545456	...	78.08594	0	157	112	...	0	7.34432	23.15328	21.1616	...	6.72192
26/06/2020 16:40	111.666664	93.666664	92	...	110	79.84615	92.545456	87.814514	...	79.28689	0	112	0	...	0	8.96256	21.1616	10.20736	...	10.82976
26/06/2020 16:45	112	92	93	...	109	80.73485	87.814514	90.175	...	79.333336	0	0	0	...	0	7.4688	10.20736	8.0912	...	7.34432
26/06/2020 16:50	112	93	94	...	112.666664	77.5	90.175	88.12281	...	81.34831	0	0	0	...	0	8.7136	8.0912	7.71776	...	7.21984
26/06/2020 16:55	112	94	95	...	116.333336	81.67669	88.12281	85.1157	...	77.38525	0	0	0	...	0	7.96672	7.71776	7.84224	...	7.09536
26/06/2020 17:00	112	95	98	...	120	81.13636	85.1157	84.60976	...	75.66304	0	0	0	...	0	8.96256	7.84224	7.96672	...	6.47296

Figure 4.1: *Original Data for t +30*



#Type	Gluct	Gluct-115	Gluct-110	...	Gluct-5	HRt	HRt-120	HRt-115	...	HRt-5	Stepst	Stepst-120	Stepst-115	...	Stepst-5	Calt	Calt-120	Calt-115	...	Calt-5
1	104	157.33333	155.66667	...	101	86.77966	75.37288	77.02381	...	88.32089	0	0	0	...	0	7.84224	8.0912	7.71776	...	7.84224
1	105.666664	155.66667	154	...	104	82.03226	77.02381	76.32407	...	86.77966	0	0	62	...	0	8.0912	7.71776	15.56	...	7.84224
1	107.333336	154	152	...	105.666664	86.72358	76.32407	83.52419	...	82.03226	0	62	0	...	0	7.96672	15.56	9.9584	...	8.0912
1	109	152	150	...	107.333336	94.87786	83.52419	71.798386	...	86.72358	8	0	29	...	0	10.45632	9.9584	11.2032	...	7.96672
1	108.333336	150	148	...	109	87.04762	71.798386	70.39024	...	94.87786	7	29	0	...	8	9.58496	11.2032	7.4688	...	10.45632
1	107.666664	148	142	...	108.333336	89.48276	70.39024	70.534485	...	87.04762	0	0	0	...	7	7.96672	7.4688	7.09536	...	9.58496
1	107	142	136	...	107.666664	92.21186	70.534485	76.29752	...	89.48276	56	0	0	...	0	13.568319	7.09536	7.96672	...	7.96672
1	106.333336	136	130	...	107	90.298386	76.29752	73.68033	...	92.21186	11	0	8	...	56	10.20736	7.96672	9.46048	...	13.568319
1	105.666664	130	125.666664	...	106.333336	85.77686	73.68033	73.40187	...	90.298386	0	8	93	...	11	7.96672	9.46048	19.16992	...	10.20736
1	105	125.666664	121.333336	...	105.666664	88.111115	73.40187	85.95536	...	85.77686	0	93	185	...	0	8.0912	19.16992	21.90848	...	7.96672
1	105.333336	121.333336	117	...	105	86.816	85.95536	95.276924	...	88.111115	33	185	22	...	0	12.94592	21.90848	15.18656	...	8.0912
1	105.666664	117	110.333336	...	105.333336	83.245766	95.276924	88.65323	...	86.816	0	22	20	...	33	7.96672	15.18656	12.199039	...	12.94592
1	106	110.333336	103.666664	...	105.666664	82.86667	88.65323	91.4955	...	83.245766	0	20	22	...	0	7.71776	12.199039	15.43552	...	7.96672
1	107.666664	103.666664	97	...	106	81.71154	91.4955	85.22857	...	82.86667	0	22	0	...	0	7.4688	15.43552	16.8048	...	7.71776
1	109.333336	97	95.333336	...	107.666664	81.10569	85.22857	93	...	81.71154	8	0	53	...	0	9.08704	16.8048	20.66368	...	7.4688
1	111	95.333336	93.666664	...	109.333336	82.031746	93	97.63717	...	81.10569	0	53	157	...	8	9.336	20.66368	23.15328	...	9.08704
1	111.333336	93.666664	92	...	111	80.09231	97.63717	92.545456	...	82.031746	0	157	112	...	0	7.34432	23.15328	21.1616	...	9.336
1	111.666664	92	93	...	111.333336	79.84615	92.545456	87.814514	...	80.09231	0	112	0	...	0	8.96256	21.1616	10.20736	...	7.34432
1	112	93	94	...	111.666664	80.73485	87.814514	90.175	...	79.84615	0	0	0	...	0	7.4688	10.20736	8.0912	...	8.96256
1	112	94	95	...	112	77.5	90.175	88.12281	...	80.73485	0	0	0	...	0	8.7136	8.0912	7.71776	...	7.4688
1	112	95	98	...	112	81.67669	88.12281	85.1157	...	77.5	0	0	0	...	0	7.96672	7.71776	7.84224	...	8.7136
1	112	98	101	...	112	81.13636	85.1157	84.60976	...	81.67669	0	0	0	...	0	8.96256	7.84224	7.96672	...	7.96672

Figure 4.2: *Edited Data*

In Fig. 4.1 we can observe a fragment of the original data structure that we then process so that our algorithm can work with it, it includes the values from t-120 to t+30 as in this example this is the value for prediction, separated in 5-minute intervals, the first column is the time at which the data is taken. It allows for a What-If scenario, where we can use the t to t+30 data as if we knew it, but we will not be using it.

In Fig. 4.2 we see the final data structure that is sent to the algorithm, the first column will be the prediction, in our case a class, and we have eliminated all the columns from t+5 to t+30 for each variable type since the prediction is made agnostic, we have no information past the actual time t.

4.2.1 Classifying data

From the raw data obtained from the patients, we will transform the glucose value at t+k k in [30,60,90,120] to the class that we will later predict. The class division will depend on what we are predicting over, we will consider 2 different divisions.

- 2 Classes.

  We make a simple division for hypoglycemia and non-hypoglycemia.

  - Class 0 (Hypoglycemia) : gluct+k $<$ 70
  - Class 1 (Non-hypoglycemia) : gluct+k $>$ 70

  With the 2 class model described the main issue is that if a value is not Hypoglycemia but is close to, it is treated the same as a value that is really far from being hypoglycemia, this is the issue with making a classification of discrete values as you lose a lot of the context in return for having a general overview.

- 3 Classes.

  To take into account the intermediate values that might not be classified correctly but are close to Hypoglycemia a vice versa.

  - Class 0 (Hypoglycemia) : gluct+k $<$ 70
  - Class 1 (Low Non-hypoglycemia) : gluct+k $>=$ 70 and gluct+k $<$ 90
  - Class 2 (High Non-hypoglycemia) : gluct+k $>=$ 90

  Using this 3-class model will result in the classes having less accuracy since whilst the Hypoglycemic classification has a medical basis dividing in gluct+k = 90 is rather arbitrary. However, having this intermediate class is much better from the patient's point of view as it adds a buffer between Hypoglycemic and Non-hypoglycemic values, so the patient is aware that the value is in a grey zone and must be careful in case it becomes a Hypoglycemia, it also adjusts to the 5% error band that can be expected from the CGM. Moreover, the value in which the division is made can be moved, since it is only a metric to illustrate low blood glucose but not yet hypoglycemia.

In the result obtained with the 3-class classification, we will be interested in seeing how well it is able to differentiate Class 0 and Class 2 from each other, as that is the main objective.

4.3 Data Preprocessing

Firstly, we have performed some data processing to give us information on the data we have obtained from the patients. From this, we can tune our algorithm and improve its performance.

If we take all the data from all the patients and divide it into hypoglycemia and not hypoglycemia, we can observe the relation of each data type to the variables at the instant t, the most recent value before prediction.

Fig 4.3 and Fig 4.4 show the relation between the 4 variables at t and are colored by their class at t+30 and t+120 respectively, from this we are able to discern that besides the glucose value none of the others are discernibly directly related to the class at instant t, which is the closest temporal value. The glucose value will also inevitably become less related the bigger the time horizon we are considering, this can be seen if we compare the graphs for glucose for t+30 and t+120.

Secondly, we have set the data so that it is in a form our program can process and perform the evolution over.

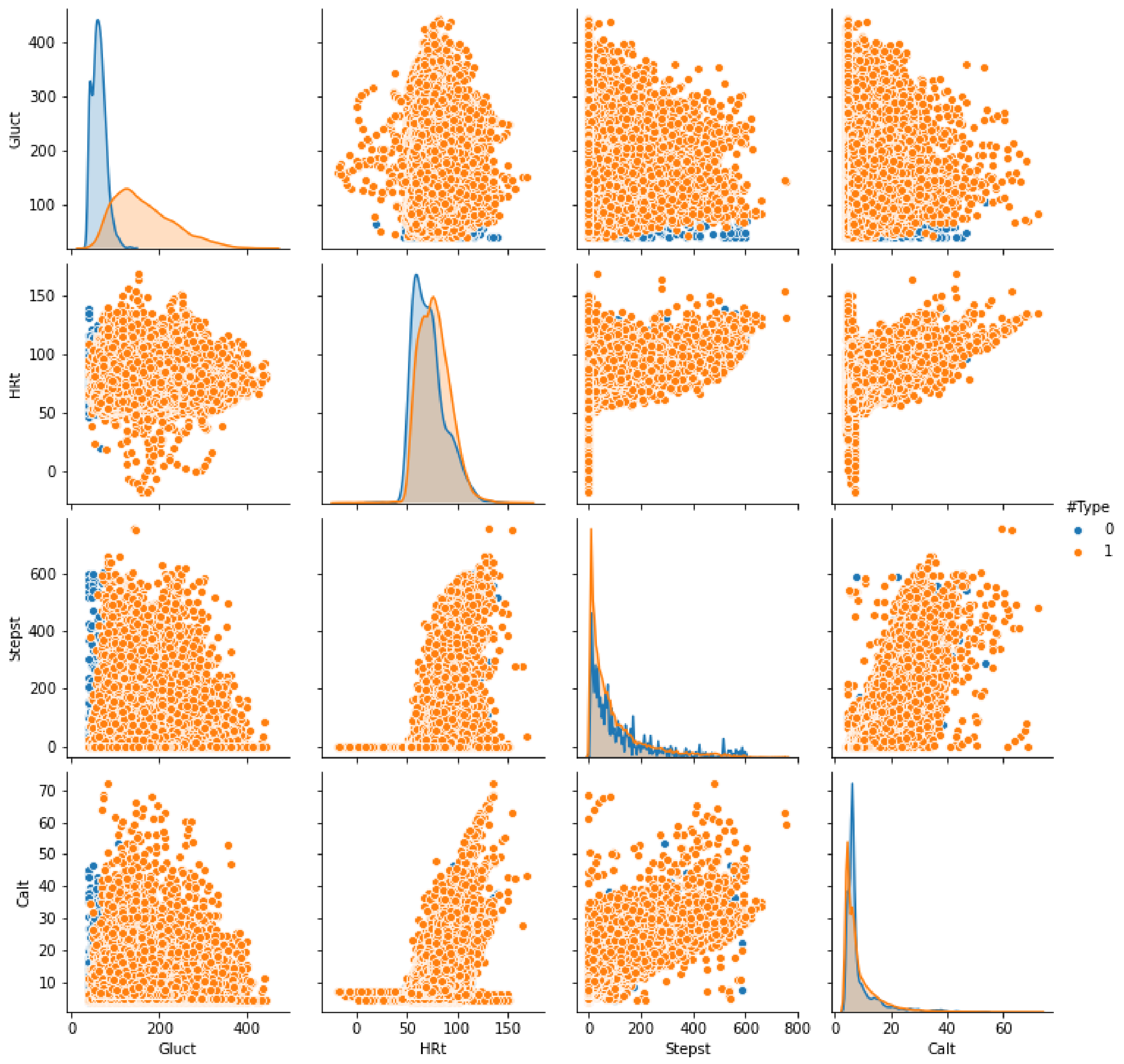


Figure 4.3: *Pair plot for the 4 variables in t colored by the class (0: Hypoglycemia, 1: Non-hypoglycemia) in t+30*

4.3.1 Correlations

Even though we have 100 columns of data that we can use to perform the classification, in our grammar files, as seen previously, we can choose what columns to consider for the classification. This will reduce the number of terminal productions for certain rules and improve the algorithm's development because we will have to take fewer values into account.

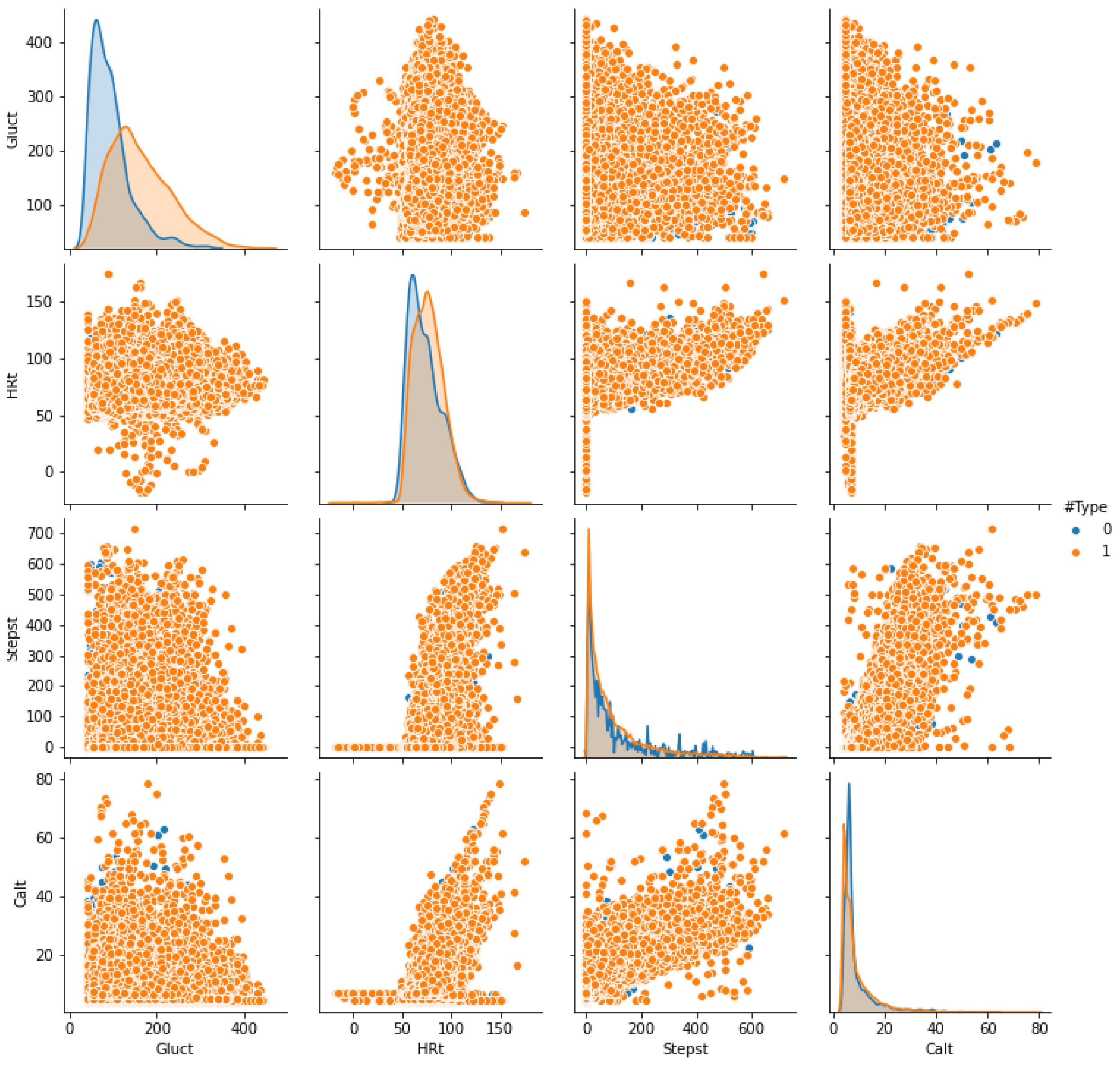


Figure 4.4: *Pair plot for the 4 variables in t colored by the class (0: Hypoglycemia, 1: Non-hypoglycemia) in t+120*

On the flip side, we might be eliminating useful data, so we must be careful when choosing what columns to use. To perform a reduction in the number of input variables that we are considering, we have first observed the correlation values between the columns of the input data of the same type.

To compute the correlation values, we have fused the data of all the patients considered and computed the correlation on the mix. In general, the correlations will be different for different patients, but we will try to use the same input variables for all of them to generate the models, which is why we compute it over the combination of all the data. If we were to use different grammars for different patients, the correlation values should be calculated separately.

From the results, we need to choose a subset of data that represents the variable's development over the 2 hours considered. We also should include the variables at t, which represents the time of prediction, since it is the newest value, and consider how many variables we want to include of each type. We will compute what columns have a lower correlation between themselves and use these for the algorithm, as they will represent the input variable's range without having too much-repeated information.

From the original 100 variables, we have reduced to just 32 of those 100, which are the ones used in the grammar files previously discussed. All the results obtained will use only these 32 variables, later we will study which variables are the ones that have been used the most to obtain the best models.

Glucose

From the input columns, the one that has a higher linear correlation in the values over time is the glucose reading. This makes sense since glucose variation in 5-minute intervals is generally not very high. The glucose readings are also the only variables that have a high linear correlation between themselves and the class for Glucose (t+k) [k : 30,60,90,120] minutes, in comparison to the other input variables. The glucose variation also represents implicitly the carbohydrates and insulin injected, whose values we are not being taken into account when making the models, but that are very important as they are directly responsible for glucose changes.

For the glucose, we will consider 5 values, Glucose (t), Glucose (t-25), Glucose (t-50), Glucose (t-75), and Glucose (t-100), that correspond to the Glucose value at the instant t, time of prediction, minus k minutes. Their correlations can be seen in Fig. 4.5. We will also include all the intermediate glucose values from Glucose (t) to Glucose (t-25) for a total of 9 glucose variables. The newest values will not add much information from the overall set, but they will show the newest tendency.

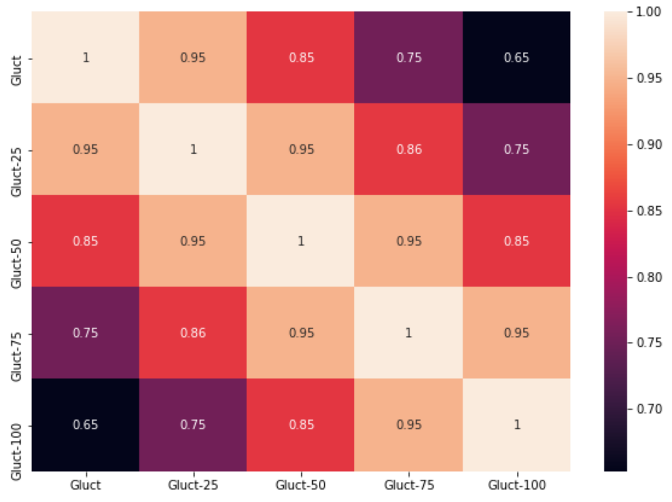
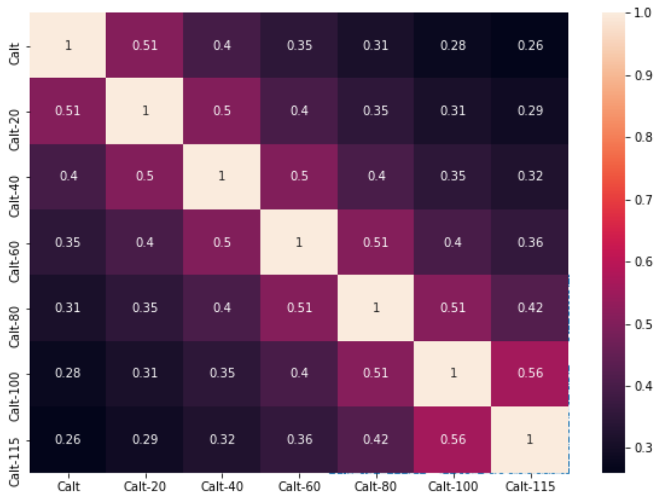


Figure 4.5: *Correlations of the 5 glucose values considered that are less correlated (not including the final 4)*

Heart rate

Similar to the glucose the heart rate readings also have a high correlation between themselves, but lower than the glucose correlations. From the heart-rate values we will get 7 values: Heart-Rate (t), Heart-Rate (t-20), Heart-Rate (t-40), Heart-Rate (t-60), Heart-Rate (t-80), Heart-Rate (t-100) and Heart-Rate (t-115), that correspond to the Heart-Rate value at instant t, time of prediction, minus k minutes. Their correlations can be seen in

Fig. 4.6



	Calt	Calt-20	Calt-40	Calt-60	Calt-80	Calt-100	Calt-115
Calt	1	0.51	0.4	0.35	0.31	0.28	0.26
Calt-20	0.51	1	0.5	0.4	0.35	0.31	0.29
Calt-40	0.4	0.5	1	0.5	0.4	0.35	0.32
Calt-60	0.35	0.4	0.5	1	0.51	0.4	0.36
Calt-80	0.31	0.35	0.4	0.51	1	0.51	0.42
Calt-100	0.28	0.31	0.35	0.4	0.51	1	0.56
Calt-115	0.26	0.29	0.32	0.36	0.42	0.56	1

Figure 4.6: *Correlations of the 7 Heart rate values considered*

Steps

This is the input variable with the least amount of correlation between itself, which makes sense as the number of steps taken from one 5-minute interval to another can vary a lot. For our problem we are interested in it describing 'activeness' since the most common exercise people do in their daily life is walking or running we have taken it into account. For the variables used we have chosen Steps (t), Steps (t-15), Steps (t-30), Steps (t-45), Steps (t-60), Steps (t-75), Steps (t-90), Steps (t-105), Steps (t-120), that correspond to the amount of steps taken at instant t, time of prediction, minus k minutes. Their correlation values can be seen in Fig. 4.7

Calories burned

The correlation of the number of calories burned in 5-minute intervals is lower than the Glucose or Heart rate, but higher than the correlation over the number of steps. This variable represents a sort of combination of the previous two, alongside a basal value that is constant.

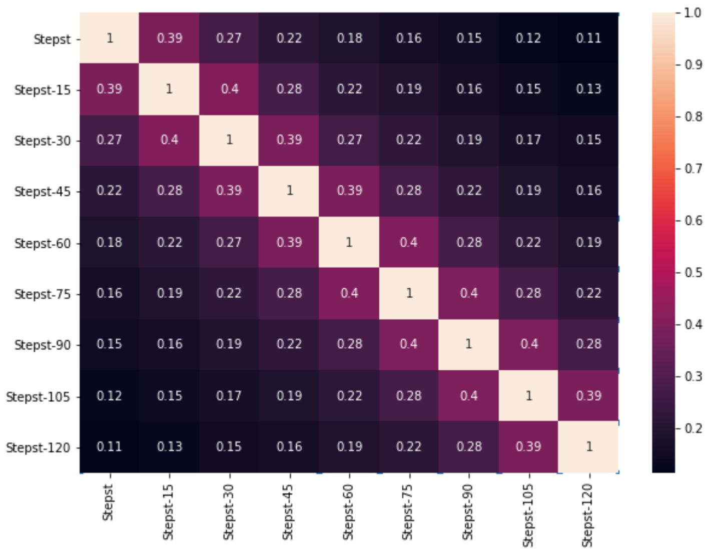


Figure 4.7: *Correlations of the 9 Steps values considered*

Although the specific formula used to compute the value depends on the Smartwatch used. The variables included are Calories (t), Calories (t-20), Calories (t-40), Calories (t-60), Calories (t-80), Calories (t-100), and Calories (t-115), for a total of 7 variables, that correspond to the number of calories burned at instant t, time of prediction, minus k minutes. Their correlations can be seen in Fig. 4.8

4.3.2 Class balancing

After performing the classification, we will obtain a data set where the classes are very unbalanced, since the recommended percentage of hypoglycemia values is under 4% of the total values. To help the algorithm learn properly and quicker, we should perform class balancing, so that there are approximately the same amount of examples in each class.

For balancing the different classes we can either do under-sampling; where we get all the examples of the class with fewer instances and a random sample of the other classes so that the amount of instances of each class is approximately the same, or oversampling; where we generate extra samples of the classes with the least amount of instances synthetically.

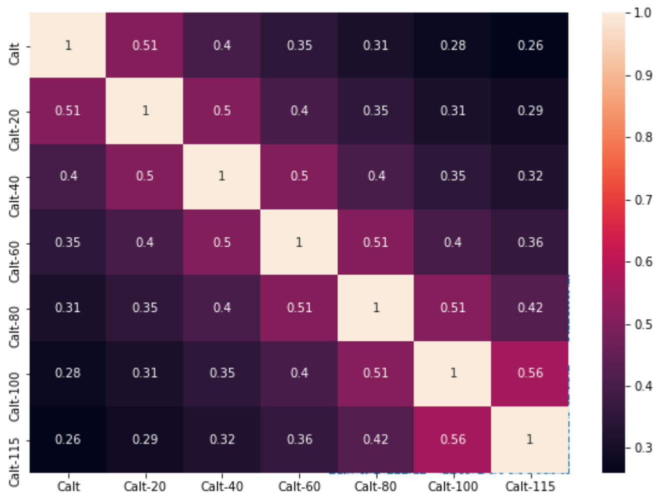


Figure 4.8: *Correlations of the 7 burned calories values considered*

For obtaining the results shown in the next chapter we have chosen to perform an undersampling of the data, as in terms of performance it is the cheapest option and the results obtained from previous tests don't show a huge advantage in performing an oversampling of the data. The balancing will only be performed over the training set.

For the 3.1 function discussed previously the data doesn't need to be balanced to work, however, it is recommended since the final results are better with balanced training data.

Chapter 5

Results

In this chapter, we will show and explain some results obtained using the methodology commented previously with the data of the patients discussed. The data has been divided into training and test with 70% and 30% of the total data respectively, maintaining the original data distribution of the classes. The fitness function we have used to perform the prediction, from the ones described, is the Weighted Accuracy, although any of them could be used. As a result of this decision the training data must be balanced to obtain good results, in general even if we use the Recall fitness function, which can deal with an unbalanced data set, we will obtain the best results after performing the data balancing. The data balancing chosen was random under-balancing.

The test data will maintain the original distribution of the classes. This ensures that the data we are testing over is the same data that will be encountered in a real-life scenario.

5.1 Experimental Set-up

To obtain the results from the evolutionary algorithm, we have set its parameters as such for all the tests:

Static Structured Grammatical Evolution

- Replacement policy: Best individuals from parents and children with a 10% regeneration percentage

- 100 population size
- 500 generations for t+30 and t+60 and 800 generations for t+90 and t+120
- Recursive Depth: 3
- Tournament selection operator with pressure 2
- Tree crossover operator
    - 70% crossover probability
    - 60% interchange probability
- Basic mutation of all alleles
    - 5% probability

Dynamic Structured Grammatical Evolution

- Replacement policy: Best individuals from parents and children with a 10% regeneration percentage
- 100 population size
- 500 generations for t+30 and t+60 and 800 generations for t+90 and t+120
- Maximum Tree Depth and Initial Maximum Tree Depth: 7
- Initial Minimum Recursive Depth: 0
- Tournament selection operator with pressure 2
- Uniform crossover operator
    - 70% crossover probability
    - 20% interchange probability

- Basic mutation of all alleles
  - 5% probability

We have set the parameters to be equivalent for both algorithm types, the two main differences are the depths: wherein the Structured Grammatical Evolution as we have two recursive rules we set depth 3 for each of them and in Dynamic Grammatical Evolution we set maximum tree depth to 7 that considers both recursive rules plus the initial rule. And the crossover operator, where we tested the tree crossover for the Static SGE, since it works better because we have more rules and the Uniform crossover for the Dynamic SGE.

For each set of data, we have executed 30 times the algorithm and obtained the mean, median, and standard deviation of the best result of each execution. With the best element of those 30 generations, we obtain the recall value on the Test data-set which will represent how well the model is able to classify the data of each class.

5.1.1 General and Individual Models

For each of the 11 patients, we are able to generate an individual model, that only uses their data for training, but it can also be useful to create a general model, that has been trained with the data of all the patients and test this model on each patient. Even though an individual approach will give us better results in almost all cases, using a general model can be useful if we want to predict a patient that we don't have a lot of data on, or at all. How well this model works will greatly depend on the specific patient and whether the data from other patients can be generalized to them or not.

We will not make such distinctions in this study, however, we will observe that some patients consistently give worse results in the general models than others, whose score is higher.

5.2 2 Classes

The data will only be divided into two classes, Hypoglycemia (Hypo) = 0 and Non-hypoglycemia (No Hypo) = 1, for each prediction horizon we will obtain how well the algorithm classifies the different input parameters and compare the recall of the general and individual models. The grammar used will be the one shown previously in Fig. 3.10.

This process is repeated for the 4 prediction horizons, [30, 60, 90, 120] minutes. For each prediction horizon, we get the Mean, Median, Std. Dev. and best fitness of the 30 executions, this information is shown in the tables; t+30: Tab. 5.1 , t+60: Tab. 5.4 , t+90: Tab. 5.7 and t+120: Tab. 5.10. For the model with the smallest fitness value of the 30 executions (both individual and general) we obtain the Recall of the classes, this is shown in the tables; t+30: Tab. 5.2 , t+60. Tab. 5.5 , t+90: Tab. 5.8 , t+120: Tab. 5.11.

For each prediction horizon, we have also tested other machine learning algorithms from SkLearn to compare how the best-obtained model fares in comparison with other Machine Learning algorithms, this can be seen in; t+30: Tab. 5.3 , t+60: Tab. 5.6, t+90: Tab. 5.9 and t+120: Tab. 5.12. The number shown is the average Recall of each class for all the patients by performing the training and test over the data from all the patients since we just wanted a general view of the comparison.

5.2.1 30 Minutes

Table 5.1: *Statistics: Mean, Median, Standard deviation and Best Fitness for all the models we have trained*

Patient	Algorithm Type	Mean	Median	Std. Dev.	Best Fitness
HUPA001	Static	0.0197	0.0227	0.0091	0.0
HUPA001	Dynamic	0.0186	0.0227	0.009	0.0
HUPA002	Static	0.101	0.1031	0.0085	0.0791
HUPA002	Dynamic	0.1004	0.1002	0.0077	0.0839
HUPA003	Static	0.0833	0.0829	0.0053	0.0733
HUPA003	Dynamic	0.0832	0.0838	0.0059	0.0684
HUPA004	Static	0.0443	0.0472	0.0081	0.0304
HUPA004	Dynamic	0.0459	0.0493	0.0086	0.0279
HUPA005	Static	0.0732	0.0759	0.01	0.0522
HUPA005	Dynamic	0.069	0.0642	0.0085	0.0578
HUPA007	Static	0.0559	0.0612	0.0118	0.0324
HUPA007	Dynamic	0.053	0.0566	0.0122	0.0302
HUPA011	Static	0.0709	0.0703	0.0103	0.0510
HUPA011	Dynamic	0.0693	0.0694	0.0097	0.0510
HUPA014	Static	0.0556	0.0567	0.0086	0.0343
HUPA014	Dynamic	0.0507	0.0517	0.0098	0.0303
HUPA015	Static	0.0664	0.0721	0.0134	0.0321
HUPA015	Dynamic	0.0643	0.0679	0.0129	0.0413
HUPA016	Static	0.096	0.0976	0.008	0.0739
HUPA016	Dynamic	0.0931	0.0933	0.0098	0.0769
HUPA027	Static	0.0776	0.0787	0.0032	0.0694
HUPA027	Dynamic	0.0764	0.0761	0.0034	0.0674
All	Static	0.0975	0.0984	0.008	0.0820
All	Dynamic	0.0945	0.0930	0.0083	0.0791

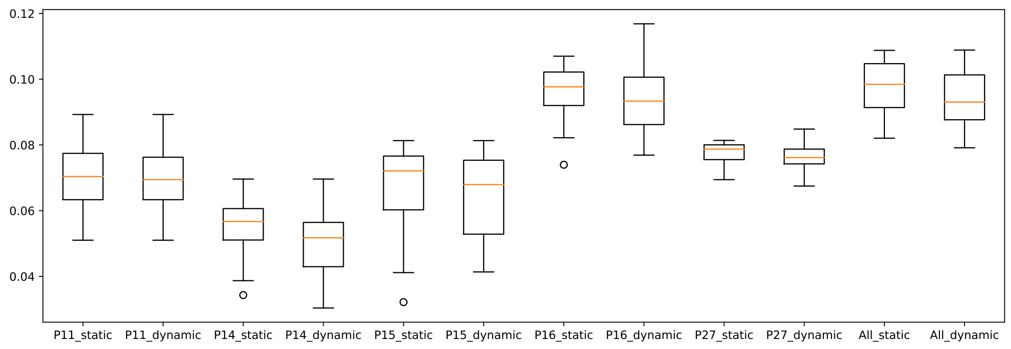


Figure 5.1: *Box plot of the fitness t+30*

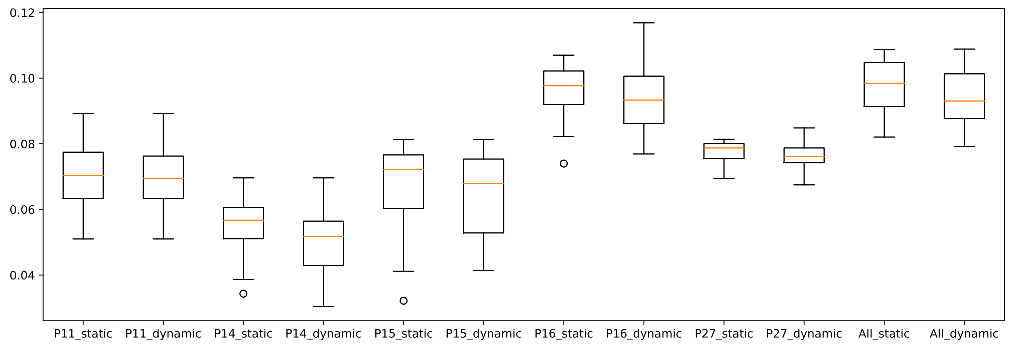


Figure 5.2: *Box plot of the fitness t+30*

Table 5.2: *Comparison of the recall of the Individual and General model for each patient in t + 30 minutes*

Patient	**Algorithm**	**Individual Models**		**General Models**	
		Recall Hypo	Recall No Hypo	Recall Hypo	Recall No Hypo
HUPA001	Static	0.9412	0.9504	0.8824	0.9488
HUPA001	Dynamic	1.0	0.9289	0.9412	0.8329
HUPA002	Static	0.88	0.9266	0.9511	0.8435
HUPA002	Dynamic	0.8756	0.9169	0.9556	0.8698
HUPA003	Static	0.9125	0.9107	0.9250	0.8896
HUPA003	Dynamic	0.9375	0.8935	0.9500	0.9002
HUPA004	Static	0.9785	0.966	1.0	0.959
HUPA004	Dynamic	1.0	0.9426	0.9785	0.966
HUPA005	Static	0.9706	0.865	0.7353	0.9109
HUPA005	Dynamic	0.9706	0.865	0.7647	0.9298
HUPA007	Static	0.9394	0.9513	0.9899	0.9217
HUPA007	Dynamic	0.9394	0.9351	0.9697	0.9255
HUPA011	Static	0.9677	0.848	0.9677	0.9263
HUPA011	Dynamic	0.9677	0.9317	0.9677	0.8903
HUPA014	Static	1.0	0.9222	0.9787	0.957
HUPA014	Dynamic	0.9574	0.9597	0.9362	0.9689
HUPA015	Static	0.9535	0.9392	0.9767	0.9309
HUPA015	Dynamic	0.9535	0.9373	0.9535	0.9244
HUPA016	Static	0.9711	0.8906	0.9075	0.903
HUPA016	Dynamic	0.9306	0.8999	0.9480	0.9133
HUPA027	Static	0.9510	0.9194	0.9301	0.9044
HUPA027	Dynamic	0.9650	0.8969	0.9161	0.9203

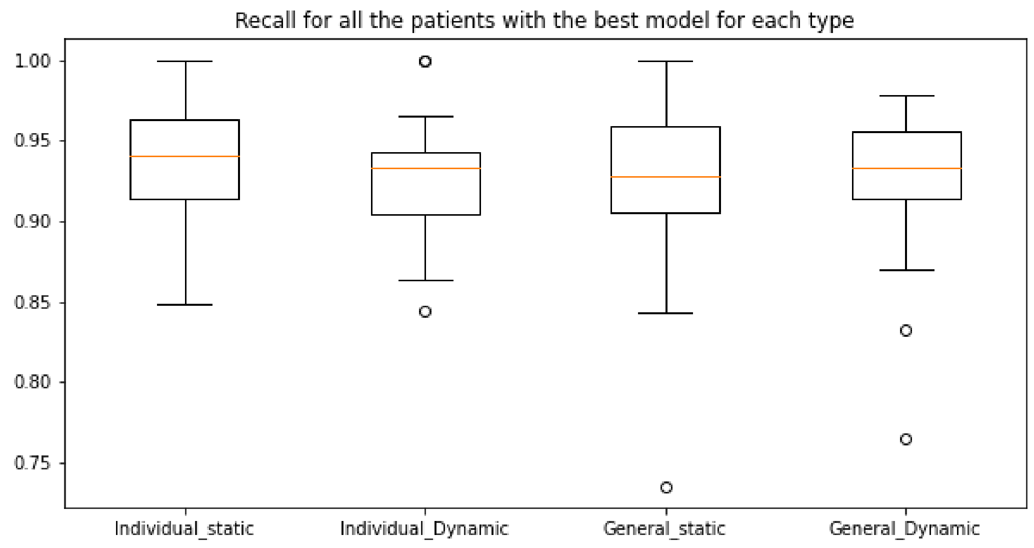


Figure 5.3: *Average recall for the best solution of all patients in t+30*

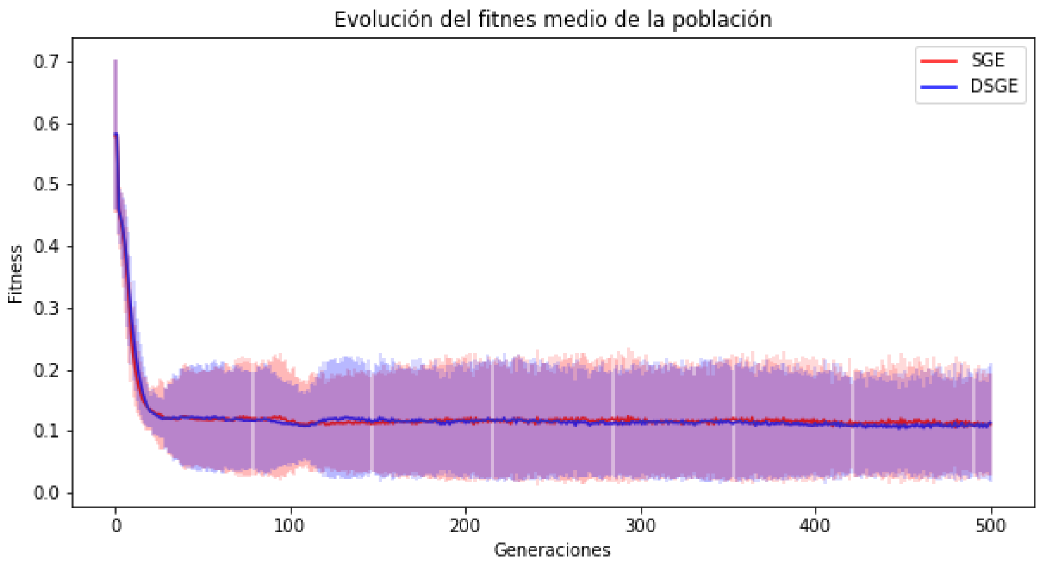


Figure 5.4: *Evolution of the average population fitness in t+30*

As can be seen in Fig. 5.3 the average recall of both classes for all patients is between 90% and 95%, and both the general and individual models yield similar results on average. Individually, the only patient that sees a big drop in the Recall values is HUPA005, where using data from the other patients does not seem to be effective for the prediction.

Comparison with other Machine Learning Algorithms

Table 5.3: *Comparison of the average Recall for our algorithm with other ML algorithms for t+30*

Algorithm	**Recall Hypo**	**Recall No Hypo**
Gradient boosting	0.9532	0.9293
Gradient boosting with Cross val	0.9532	0.9293
Logistic Regression	0.9187	0.8951
Random forest classifier	0.9837	0.9287
Random forest Regresor	0.9868	0.9307
Gaussian Naive Bayes	0.8720	0.7893
Bernoulli Naive Bayes	0.6020	0.5304
Decision Tree Classifier (6 depth)	0.9431	0.9182
Ada Boost Classifier	0.9604	0.9246
Bagging Classifier	0.9868	0.9305
Extra Trees Classifier	0.9868	0.9290
SGDClassifier	0.9573	0.7875
MLPClassifier	0.9390	0.8748
Dynamic SGE	0.9542	0.9188
Static SGE	0.9514	0.9254

5.2.2 60 Minutes

Table 5.4: *Statistics: Mean, Median, Standard deviation and Best Fitness for all the models we have trained*

Patient	Algorithm Type	Mean	Median	Std. Dev.	Best Fitness
HUPA001	Static	0.0133	0.0123	0.0092	0.0
HUPA001	Dynamic	0.0117	0.0123	0.0085	0.0
HUPA002	Static	0.1656	0.1642	0.0086	0.1514
HUPA002	Dynamic	0.1646	0.1662	0.0089	0.1445
HUPA003	Static	0.1402	0.1411	0.0111	0.1033
HUPA003	Dynamic	0.1385	0.1380	0.0112	0.1205
HUPA004	Static	0.0889	0.0828	0.0168	0.0630
HUPA004	Dynamic	0.0884	0.0878	0.0135	0.0588
HUPA005	Static	0.1911	0.1942	0.0211	0.1490
HUPA005	Dynamic	0.188	0.1937	0.0197	0.1470
HUPA007	Static	0.1238	0.1245	0.0095	0.1028
HUPA007	Dynamic	0.1218	0.1259	0.0123	0.0925
HUPA011	Static	0.109	0.1099	0.0176	0.0815
HUPA011	Dynamic	0.1086	0.1069	0.0131	0.0886
HUPA014	Static	0.0970	0.1	0.0145	0.0482
HUPA014	Dynamic	0.0971	0.0984	0.0121	0.0572
HUPA015	Static	0.1327	0.1330	0.014	0.0969
HUPA015	Dynamic	0.1304	0.1302	0.0137	0.0999
HUPA016	Static	0.1722	0.1710	0.0129	0.1448
HUPA016	Dynamic	0.1692	0.1724	0.0111	0.1434
HUPA027	Static	0.1206	0.1203	0.0048	0.1101
HUPA027	Dynamic	0.1205	0.1202	0.0045	0.1113
All	Static	0.1709	0.1716	0.0119	0.1511
All	Dynamic	0.1692	0.1662	0.0105	0.1574

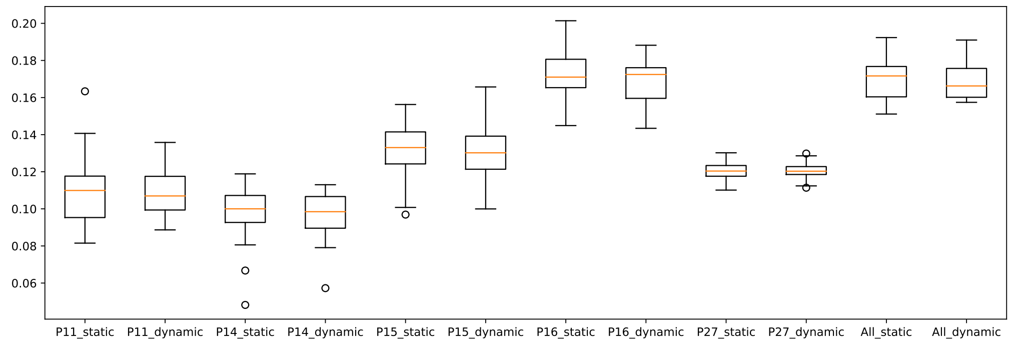


Figure 5.5: *Box plot of the fitness t+60*

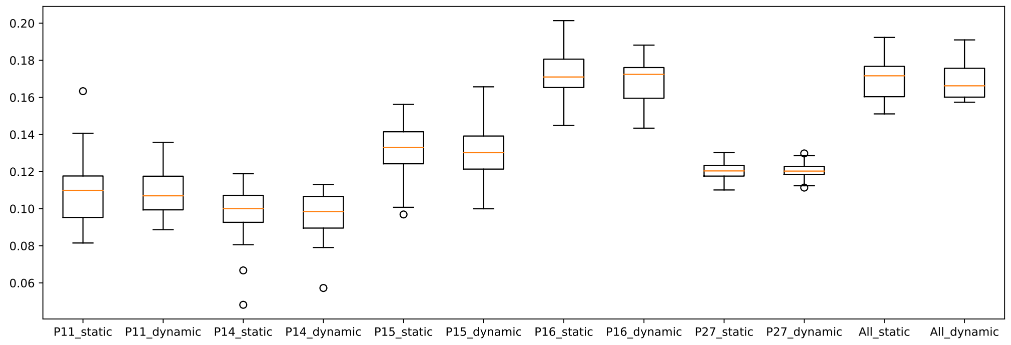


Figure 5.6: *Box plot of the fitness t+60*

Table 5.5: *Comparison of the recall of the Individual and General model for each patient in t + 60 minutes*

Patient	**Algorithm**	**Individual Models**		**General Models**	
		Recall Hypo	Recall No Hypo	Recall Hypo	Recall No Hypo
HUPA001	Static	1.0	0.9144	0.9375	0.9136
HUPA001	Dynamic	1.0	0.9072	0.9375	0.8896
HUPA002	Static	0.9063	0.7781	0.9152	0.7087
HUPA002	Dynamic	0.8616	0.8169	0.9375	0.6879
HUPA003	Static	0.8500	0.875	0.9375	0.8077
HUPA003	Dynamic	0.8250	0.8894	0.9250	0.7769
HUPA004	Static	0.9355	0.9354	0.9462	0.9167
HUPA004	Dynamic	0.9677	0.9401	0.9785	0.9143
HUPA005	Static	0.7941	0.8242	0.7353	0.8783
HUPA005	Dynamic	0.9118	0.8142	0.7353	0.8638
HUPA007	Static	0.9394	0.8547	0.9394	0.8642
HUPA007	Dynamic	0.9495	0.8805	0.9495	0.8298
HUPA011	Static	1.0	0.7874	0.8065	0.8847
HUPA011	Dynamic	0.9355	0.8099	0.8387	0.8423
HUPA014	Static	0.9574	0.8625	0.8298	0.9056
HUPA014	Dynamic	0.9362	0.8708	0.8936	0.8973
HUPA015	Static	0.9302	0.7638	0.8372	0.8708
HUPA015	Dynamic	0.814	0.8109	0.8605	0.8441
HUPA016	Static	0.896	0.7942	0.8555	0.7901
HUPA016	Dynamic	0.9364	0.7570	0.8555	0.7715
HUPA027	Static	0.9172	0.8376	0.9172	0.7944
HUPA027	Dynamic	0.9103	0.8347	0.9172	0.7803

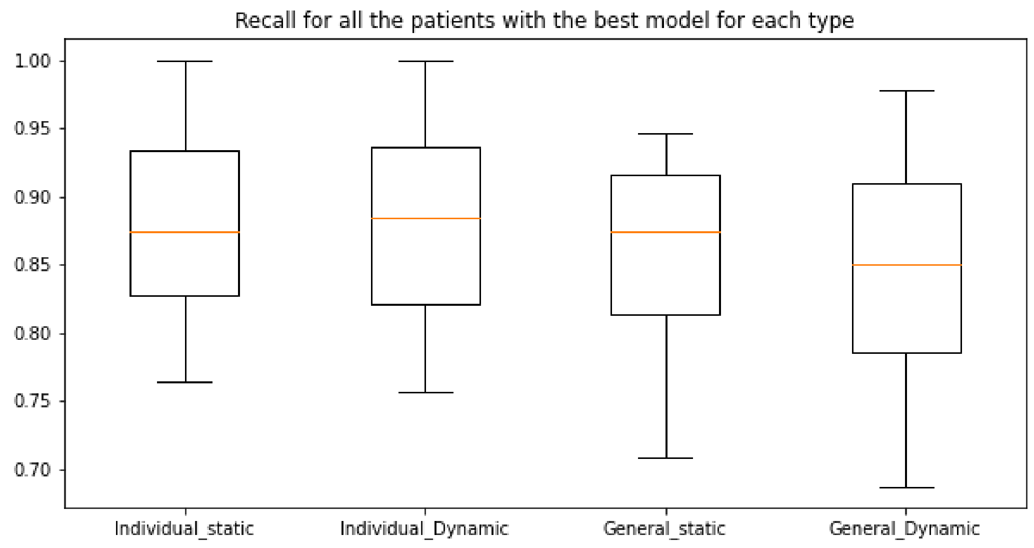


Figure 5.7: *Average recall for the best solution of all patients in t+60*

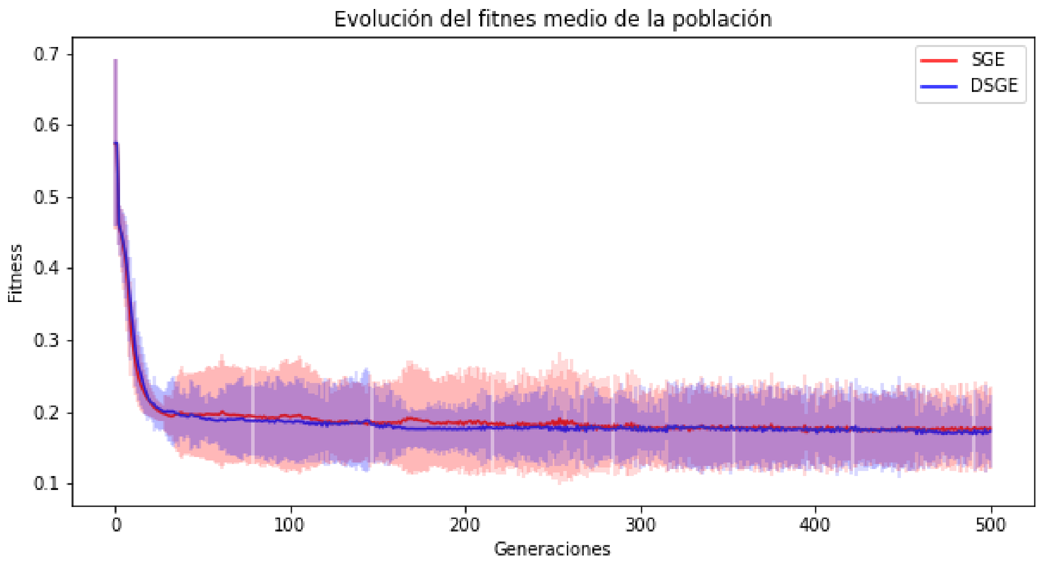


Figure 5.8: *Evolution of the average population fitness in t+60*

For t+60 we can observe that the general model has a slightly less average Recall for both classes than the individual model, but in general, the difference is not significant. Once more HUPA005 is the patient that yields worse results in the general model. The Recall values are between 0.85 and 0.9 which can be seen in Fig. 5.7.

Comparison with other Machine Learning algorithms

Table 5.6: *Comparison of the average Recall for our algorithm with other ML algorithms for t+60*

Algorithm	Recall Hypo	Recall No Hypo
Gradient boosting	0.9025	0.8595
Gradient boosting with Cross val	0.9025	0.8595
Logistic Regression	0.8832	0.8235
Random forest classifier	0.9695	0.8838
Random forest Regresor	0.9705	0.8814
Gaussian Naive Bayes	0.8659	0.6732
Bernoulli Naive Bayes	0.5857	0.5312
Decision Tree Classifier (6 depth)	0.9279	0.8038
Ada Boost Classifier	0.9421	0.8464
Bagging Classifier	0.9695	0.8782
Extra Trees Classifier	0.9746	0.8942
SGDClassifier	0.9675	0.5947
MLPClassifier	0.9147	0.7789
Dynamic SGE	0.9046	0.8218
Static SGE	0.8954	0.8536

5.2.3 90 Minutes

Table 5.7: *Statistics: Mean, Median, Standard deviation and Best Fitness for all the models we have trained*

Patient	Algorithm Type	Mean	Median	Std. Dev.	Best Fitness
HUPA001	Static	0.0617	0.0618	0.024	0.0245
HUPA001	Dynamic	0.0587	0.0505	0.0203	0.0245
HUPA002	Static	0.2285	0.2297	0.0104	0.2108
HUPA002	Dynamic	0.2249	0.2233	0.0094	0.2108
HUPA003	Static	0.1705	0.1707	0.0086	0.1549
HUPA003	Dynamic	0.1701	0.1757	0.0135	0.1449
HUPA004	Static	0.1383	0.1434	0.0232	0.0934
HUPA004	Dynamic	0.1223	0.1168	0.0207	0.0885
HUPA005	Static	0.1751	0.1780	0.0188	0.1426
HUPA005	Dynamic	0.1740	0.1742	0.0134	0.1281
HUPA007	Static	0.1756	0.1755	0.0142	0.1487
HUPA007	Dynamic	0.1738	0.1763	0.0124	0.1424
HUPA011	Static	0.1228	0.1241	0.019	0.0694
HUPA011	Dynamic	0.1203	0.1206	0.0166	0.0775
HUPA014	Static	0.1512	0.1535	0.0117	0.1125
HUPA014	Dynamic	0.1544	0.1563	0.0115	0.1292
HUPA015	Static	0.1892	0.1923	0.0178	0.1473
HUPA015	Dynamic	0.1868	0.1857	0.0186	0.1317
HUPA016	Static	0.2168	0.2126	0.0156	0.1941
HUPA016	Dynamic	0.2150	0.2150	0.0134	0.1890
HUPA027	Static	0.1778	0.1766	0.0099	0.1560
HUPA027	Dynamic	0.1777	0.1781	0.0089	0.1493
All	Static	0.2339	0.2334	0.0087	0.2172
All	Dynamic	0.2305	0.2280	0.0092	0.2148

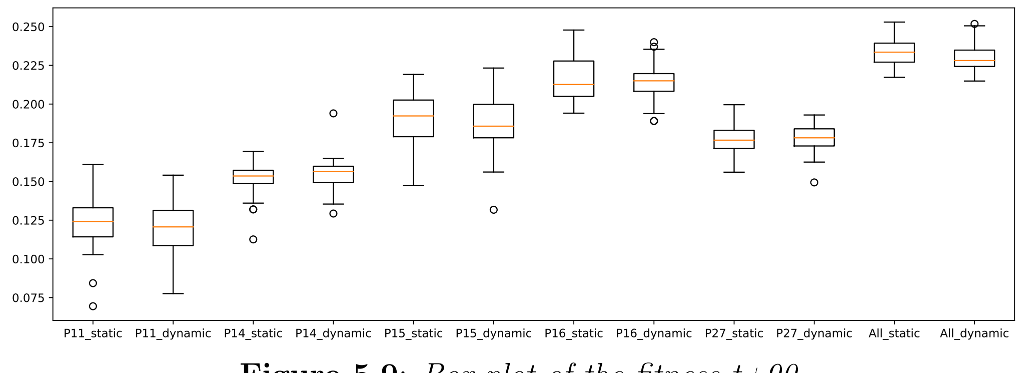


Figure 5.9: *Box plot of the fitness t+90*

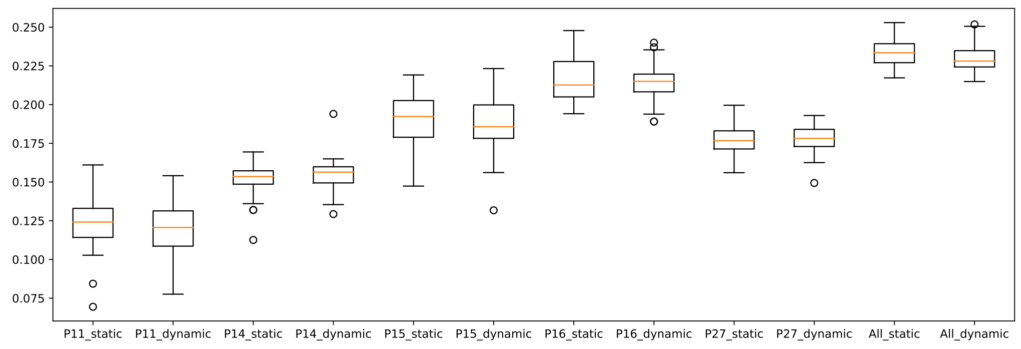


Figure 5.10: *Box plot of the fitness t+90*

Table 5.8: *Comparison of the recall of the Individual and General model for each patient in t + 90 minutes*

Patient	**Algorithm**	**Individual Models**		**General Models**	
		Recall Hypo	Recall No Hypo	Recall Hypo	Recall No Hypo
HUPA001	Static	0.9375	0.8702	0.8750	0.8494
HUPA001	Dynamic	0.875	0.8309	0.8750	0.7877
HUPA002	Static	0.8739	0.6741	0.9009	0.6699
HUPA002	Dynamic	0.8604	0.7337	0.9054	0.6699
HUPA003	Static	0.825	0.8508	0.7875	0.7247
HUPA003	Dynamic	0.7875	0.8422	0.8250	0.7113
HUPA004	Static	0.8387	0.9353	0.8495	0.8776
HUPA004	Dynamic	0.8495	0.9106	0.9032	0.8341
HUPA005	Static	0.7647	0.7669	0.5882	0.7696
HUPA005	Dynamic	0.8235	0.6784	0.5294	0.7660
HUPA007	Static	0.8081	0.7969	0.7879	0.7711
HUPA007	Dynamic	0.8687	0.7778	0.8687	0.7366
HUPA011	Static	0.9677	0.7211	0.7419	0.8114
HUPA011	Dynamic	0.9032	0.7987	0.6452	0.7861
HUPA014	Static	0.8936	0.8431	0.8511	0.8385
HUPA014	Dynamic	0.8723	0.8450	0.7234	0.8523
HUPA015	Static	0.9024	0.7011	0.7805	0.7915
HUPA015	Dynamic	0.9268	0.7814	0.7805	0.7934
HUPA016	Static	0.8035	0.7772	0.8613	0.6974
HUPA016	Dynamic	0.8555	0.7119	0.7977	0.7161
HUPA027	Static	0.8844	0.7865	0.9184	0.7103
HUPA027	Dynamic	0.8980	0.8109	0.9048	0.7093

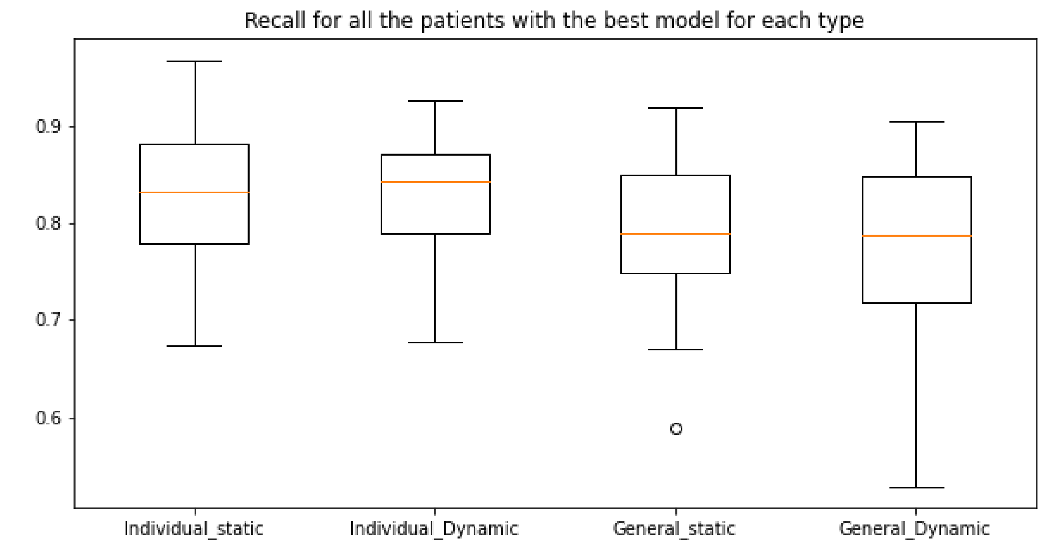


Figure 5.11: *Average recall for the best solution of all patients in t+90*

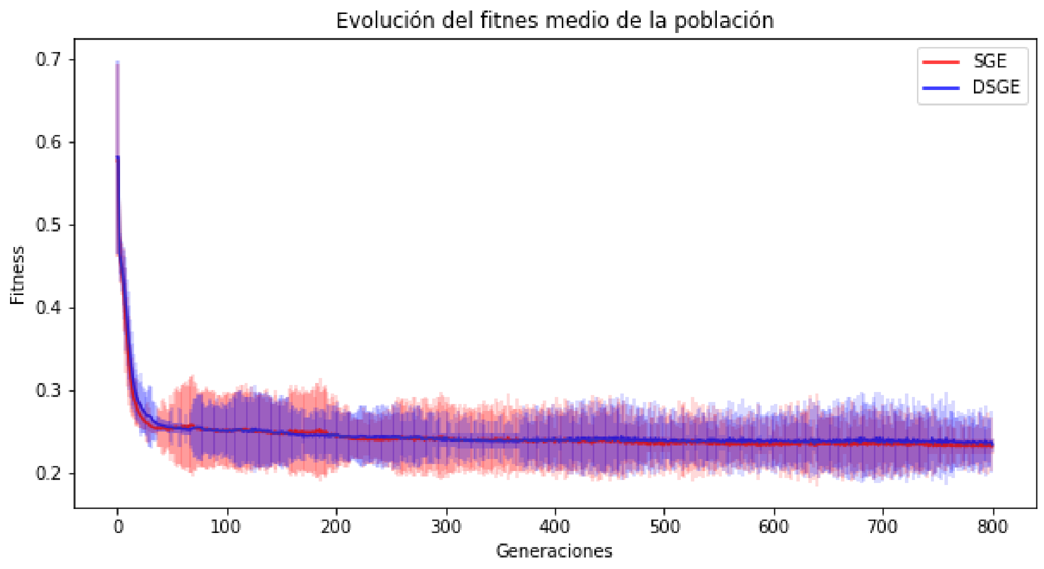


Figure 5.12: *Evolution of the average population fitness in t+90*

For t+90 we can observe that once more the general model has less average Recall for both classes than the individual model, this is more noticeable than for t+60, but the difference is still not statistically significant. The average Recall values are between 80% and 90% as shown in Fig. 5.11. Lower than before but a significant percentage.

Comparison with other Machine Learning algorithms

Table 5.9: *Comparison of the average Recall for our algorithm with other ML algorithms for t+90*

Algorithm	**Recall Hypo**	**Recall No Hypo**
Gradient boosting	0.8209	0.8218
Gradient boosting with Cross val	0.8036	0.8202
Logistic Regression	0.8260	0.7541
Random forest classifier	0.9633	0.8544
Random forest Regresor	0.9562	0.8491
Gaussian Naive Bayes	0.7924	0.5963
Bernoulli Naive Bayes	0.5857	0.5312
Decision Tree Classifier (6 depth)	0.7924	0.8013
Ada Boost Classifier	0.9186	0.8051
Bagging Classifier	0.9532	0.8446
Extra Trees Classifier	0.9593	0.8807
SGDClassifier	0.6998	0.8275
MLPClassifier	0.8769	0.7055
Dynamic SGE	0.8654	0.7928
Static SGE	0.8630	0.7930

5.2.4 120 Minutes

Table 5.10: *Statistics: Mean, Median, Standard deviation and Best Fitness for all the models we have trained*

Patient	Algorithm Type	Mean	Median	Std. Dev.	Best Fitness
HUPA001	Static	0.0788	0.0737	0.0261	0.0252
HUPA001	Dynamic	0.0774	0.0801	0.0325	0.0125
HUPA002	Static	0.2455	0.2446	0.0080	0.2264
HUPA002	Dynamic	0.2416	0.2393	0.0085	0.2274
HUPA003	Static	0.1942	0.1956	0.0167	0.1535
HUPA003	Dynamic	0.1925	0.1888	0.0136	0.1646
HUPA004	Static	0.1845	0.1841	0.0159	0.1553
HUPA004	Dynamic	0.1882	0.1844	0.0156	0.1541
HUPA005	Static	0.2006	0.1974	0.0186	0.1611
HUPA005	Dynamic	0.1972	0.1916	0.0153	0.1588
HUPA007	Static	0.2456	0.2535	0.0218	0.1951
HUPA007	Dynamic	0.2339	0.2358	0.0194	0.1875
HUPA011	Static	0.1414	0.1363	0.0190	0.1127
HUPA011	Dynamic	0.1374	0.1363	0.0148	0.1102
HUPA014	Static	0.1991	0.1938	0.0176	0.1639
HUPA014	Dynamic	0.1967	0.1930	0.0193	0.1632
HUPA015	Static	0.2078	0.2070	0.0220	0.1549
HUPA015	Dynamic	0.2021	0.2041	0.0238	0.1542
HUPA016	Static	0.2395	0.2378	0.0121	0.2170
HUPA016	Dynamic	0.2424	0.2398	0.0161	0.2160
HUPA027	Static	0.2209	0.2219	0.0117	0.2021
HUPA027	Dynamic	0.2206	0.2192	0.0085	0.2072
All	Static	0.2840	0.2833	0.0100	0.2662
All	Dynamic	0.2829	0.2820	0.0072	0.2678

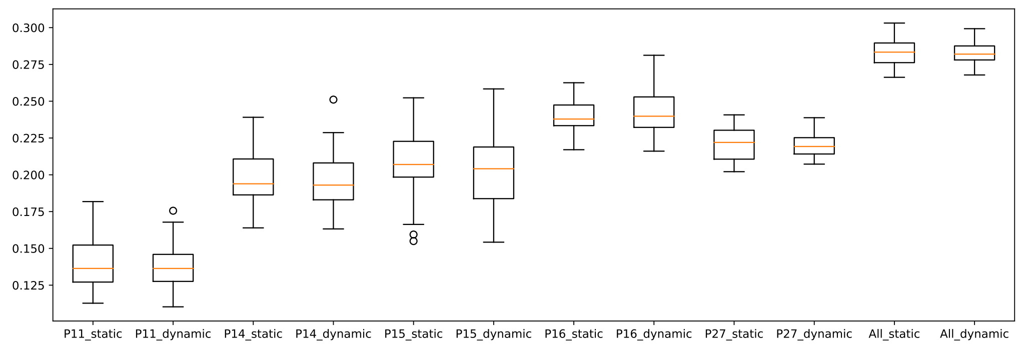


Figure 5.13: *Box plot of the fitness t+120*

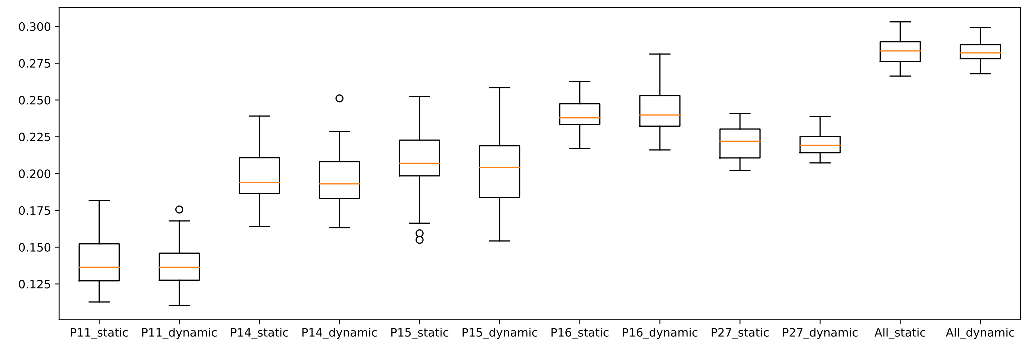


Figure 5.14: *Box plot of the fitness t+120*

Table 5.11: *Comparison of the recall of the Individual and General model for each patient in t + 120 minutes*

Patient	**Algorithm**	**Individual Models**		**General Models**	
		Recall Hypo	Recall No Hypo	Recall Hypo	Recall No Hypo
HUPA001	Static	0.75	0.749	0.8125	0.6512
HUPA001	Dynamic	0.8125	0.7498	0.7500	0.7057
HUPA002	Static	0.8364	0.6893	0.8273	0.6574
HUPA002	Dynamic	0.7955	0.7060	0.8364	0.6630
HUPA003	Static	0.7875	0.7907	0.6625	0.6220
HUPA003	Dynamic	0.7625	0.8457	0.6625	0.7059
HUPA004	Static	0.8280	0.8196	0.7849	0.8160
HUPA004	Dynamic	0.8602	0.8443	0.7527	0.8408
HUPA005	Static	0.7647	0.7167	0.8235	0.5439
HUPA005	Dynamic	0.7059	0.7964	0.6176	0.6742
HUPA007	Static	0.7374	0.7553	0.6465	0.8052
HUPA007	Dynamic	0.6566	0.8666	0.6970	0.7850
HUPA011	Static	0.8387	0.8083	0.7742	0.7269
HUPA011	Dynamic	0.8710	0.7577	0.7097	0.7568
HUPA014	Static	0.8511	0.7711	0.6170	0.7381
HUPA014	Dynamic	0.8511	0.7693	0.6154	0.7426
HUPA015	Static	0.8974	0.6384	0.6154	0.7177
HUPA015	Dynamic	0.7949	0.6190	0.3846	0.7196
HUPA016	Static	0.7977	0.7095	0.8844	0.5902
HUPA016	Dynamic	0.8439	0.6784	0.7746	0.6992
HUPA027	Static	0.8725	0.7125	0.8859	0.6437
HUPA027	Dynamic	0.8188	0.7804	0.8255	0.7257

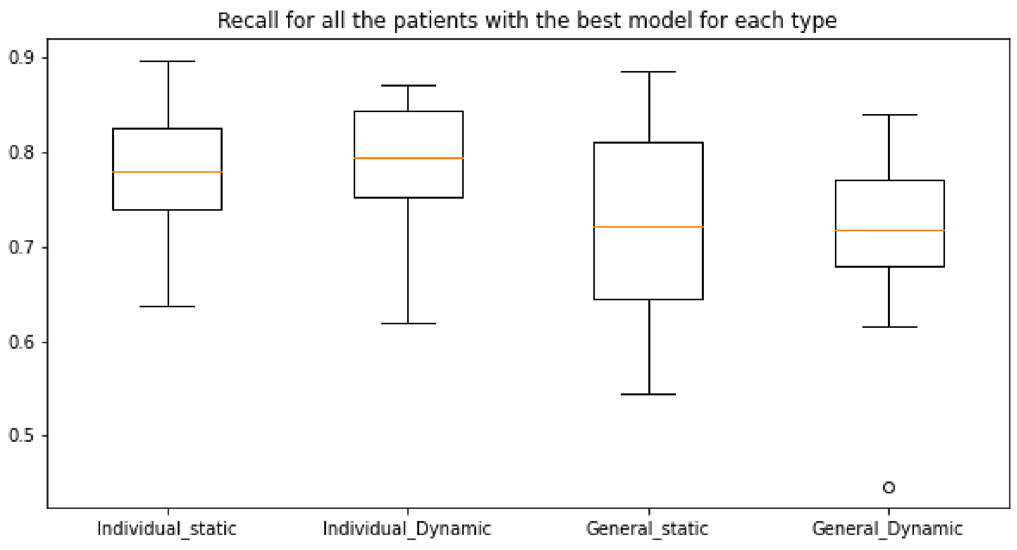


Figure 5.15: *Average recall for the best solution of all patients in t+120*

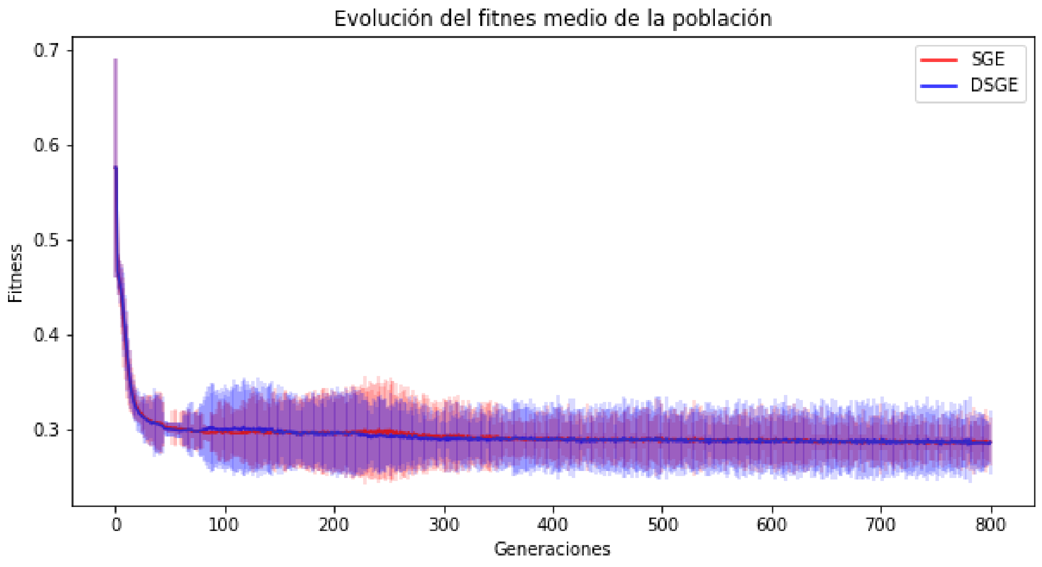


Figure 5.16: *Evolution of the average population fitness in t+30*

In t+120 is where we observe the algorithm returning the worst results, it is the hardest time frame to predict as we don't have any data of the previous 2 hours. The comparison between the individual and general model show the biggest difference. The average Recall obtained for this prediction is between 70% and 80% as shown in Fig. 5.15.

Comparison with other Machine Learning algorithms

Table 5.12: *Comparison of the average Recall for our algorithm with other ML algorithms for t+120*

Algorithm	**Recall Hypo**	**Recall No Hypo**
Gradient boosting	0.7573	0.7995
Gradient boosting with Cross val	0.7390	0.7940
Logistic Regression	0.7889	0.6984
Random forest classifier	0.9347	0.8575
Random forest Regresor	0.9317	0.8503
Gaussian Naive Bayes	0.7624	0.5600
Bernoulli Naive Bayes	0.5857	0.5312
Decision Tree Classifier (6 depth)	0.7533	0.7520
Ada Boost Classifier	0.9133	0.7756
Bagging Classifier	0.9317	0.8056
Extra Trees Classifier	0.9531	0.8811
SGDClassifier	0.7900	0.5823
MLPClassifier	0.9255	0.4865
Dynamic SGE	0.7975	0.7648
Static SGE	0.8149	0.7418

5.2.5 Results for 2 classes

As can be seen in the BoxPlots included for t +30: Fig. 5.1, Fig, 5.2, t+60: Fig. 5.5, Fig. 5.6, t+90: Fig. 5.9, Fig. 5.10 and t+120: Fig. 5.13, Fig. 5.14, there seems to be no statistical difference between the performance of the Static or original SGE and the Dynamic SGE, however the Dynamic SGE gets slightly better average results for almost all patients. The recall results over the test dataset are also very similar with both versions of the algorithm, the slight difference between both shows no statistical significance.

In terms of the general model, we can observe that it works relatively well for all patients with the exception of HUPA005, where it gives lower results than its individual model, this trend is also seen in HUPA014 and HUPA015 when considering bigger prediction horizons (t+90 and t+120). However, the results obtained in the other patients show that a general model can be possible when a specific subset of patients is considered.

In comparison with the other ML learning algorithms considered our algorithms do not do as well as some of them, especially Ada Boost, Bagging Classifier, Random Forest, and Extra Trees Classifier. All of them give excellent results of the recall, especially in the Hypoglycemia class for all the time frames considered, where both our algorithms start to lose Recall percentage in bigger prediction horizons. The main advantage of our algorithm is that the final model is an expression that can be easily understood, studied, and manually edited by an expert after obtaining it.

The results obtained will have a structure similar to the following one, with different constants, variables and relational/logical operators:

```
if(((getVariable(1,k)<=103.465)||(((getVariable(47,k)-(getVariable(28,k)*
getVariable(43,k)/getVariable(31,k)))>=(getVariable(1,k)/6.184))&&
(((3.956/(getVariable(21,k)+94.433))*406.226)<=getVariable(35,k)))))
{result=0:}else{result=1:}
```

Which is equivalent to this expression with the actual names of the variables:

```
if((( Glucose(t) <=103.465)||((HR(t-20) -(HR(t-115) * HR
(t-40)/HR(t-100)))>=(Glucose(t)/6.184))&&
(((3.956/(Gluc(t-25) +94.433))*406.226)<=HR(t-80)))))
{result=0:}else{result=1:}
```

5.2.6 Used variables

The used variables depend on the solution, the time frame we are considering for the prediction, and the patient, however, we can study in wide terms what variables are most used by the best solutions of each run. In the table below 5.13 and in Fig. 5.17 we have all the variables ordered by their number and with the corresponding name for t + [30,60,90,120], where the total number is the total amount of solutions obtained, and we only count one appearance per solution. Firstly, we are diving between the Static or original SGE and the Dynamic SGE. Clearly, by far, the variable that the solutions use most is Glucose (t), this makes sense as is it the most recent glucose value before the prediction, the other variables at t, HeartRate (t), Cal (t), and Steps (t) are also used in a higher proportion than other variables in their respective types, although the difference is much smaller than for the Glucose (t).

In both algorithms, the second and third most used variables, appearing in more than 30% of the solutions, are HeartRate (t) and Calories Burned (t) respectively. After this the order of the variables changes slightly for both algorithms, however, the actual percentages are not too different as they are all between 20% and 30%. The two less used variables in both algorithms are Glucose (t-10) and Steps (t-60) but in a different order for each one. The glucose value might be a result of duplicated information as we have added the newer values, but the algorithms don't seem to use them a lot in the solutions.

In general, besides the Glucose(t) all the other variables are used between 18.7%, 19.7%, and 31%, 33% for each algorithm.

We cannot see a big difference between the variables used in both algorithms, so we are doing to choose one, DSGE, and observe the variable distribution for the different prediction horizons, Tab. 5.14 and Fig. 5.18. In this case, more variation in the variables use is noticeable, especially for t+120 where the amount of instances of Glucose (t) is a lot less than in the other prediction horizons. Nonetheless, Glucose (t) is still, by far, the most used variable, and the rest follow a similar trend to what we explained previously.

Table 5.13: *Percentage of variable involvement in the final solutions for all executions of 30 runs*

Variable	Name	Static	Dynamic
getVariable(1,k)	Gluc (t)	0.9220	0.9265
getVariable(6,k)	Gluc (t-100)	0.2788	0.2674
getVariable(11,k)	Gluc (t-75)	0.2795	0.2864
getVariable(16,k)	Gluc (t-50)	0.2826	0.3061
getVariable(21,k)	Gluc (t-25)	0.2068	0.225
getVariable(22,k)	Gluc (t-20)	0.2114	0.2061
getVariable(23,k)	Gluc (t-15)	0.2045	0.2212
getVariable(24,k)	Gluc (t-10)	0.1962	0.1977
getVariable(25,k)	Gluc (t-5)	0.2508	0.2591
getVariable(26,k)	HR (t)	0.3197	0.3303
getVariable(28,k)	HR (t-115)	0.2992	0.2894
getVariable(31,k)	HR (t-100)	0.2864	0.2826
getVariable(35,k)	HR (t-80)	0.2818	0.2689
getVariable(39,k)	HR (t-60)	0.2735	0.2727
getVariable(43,k)	HR (t-40)	0.2924	0.2811
getVariable(47,k)	HR (t-20)	0.2818	0.2667
getVariable(51,k)	Steps (t)	0.2856	0.253
getVariable(52,k)	Steps (t-120)	0.2356	0.2455
getVariable(55,k)	Steps (t-105)	0.2129	0.2258
getVariable(58,k)	Steps (t-90)	0.2076	0.2212
getVariable(61,k)	Steps (t-75)	0.2652	0.2614
getVariable(64,k)	Steps (t-60)	0.2174	0.2197
getVariable(67,k)	Steps (t-45)	0.1985	0.1879
getVariable(70,k)	Steps (t-30)	0.2424	0.2735
getVariable(73,k)	Steps (t-15)	0.2765	0.2727
getVariable(76,k)	Cal (t)	0.3106	0.3136
getVariable(78,k)	Cal (t-115)	0.2902	0.2871
getVariable(81,k)	Cal (t-100)	0.2955	0.2902
getVariable(85,k)	Cal (t-80)	0.2735	0.2803
getVariable(89,k)	Cal (t-60)	0.278	0.2583
getVariable(93,k)	Cal (t-40)	0.2886	0.2962
getVariable(97,k)	Cal (t-20)	0.2909	0.2962

Table 5.14: *Distribution of the variables in the solutions divided by the prediction horizon*

Variable	**Name**	**Dynamic t +30**	**Dynamic t +60**	**Dynamic t +90**	**Dynamic t +120**
getVariable(1,k)	Gluc (t)	0.9879	0.9788	0.9485	0.7909
getVariable(6,k)	Gluc (t-100)	0.1758	0.2303	0.3545	0.3091
getVariable(11,k)	Gluc (t-75)	0.2394	0.2424	0.2727	0.3909
getVariable(16,k)	Gluc (t-50)	0.2182	0.2939	0.3636	0.3485
getVariable(21,k)	Gluc (t-25)	0.1788	0.2121	0.2303	0.2788
getVariable(22,k)	Gluc (t-20)	0.1424	0.2333	0.1970	0.2515
getVariable(23,k)	Gluc (t-15)	0.2152	0.1848	0.1970	0.2879
getVariable(24,k)	Gluc (t-10)	0.1758	0.1576	0.2333	0.2242
getVariable(25,k)	Gluc (t-5)	0.203	0.2364	0.2970	0.3000
getVariable(26,k)	HR (t)	0.2879	0.2515	0.3970	0.3848
getVariable(28,k)	HR (t-115)	0.2424	0.2727	0.3394	0.3030
getVariable(31,k)	HR (t-100)	0.2061	0.3485	0.2970	0.2788
getVariable(35,k)	HR (t-80)	0.2545	0.2697	0.2758	0.2758
getVariable(39,k)	HR (t-60)	0.1939	0.2970	0.2970	0.3030
getVariable(43,k)	HR (t-40)	0.2242	0.2606	0.3212	0.3182
getVariable(47,k)	HR (t-20)	0.2152	0.2545	0.3030	0.2939
getVariable(51,k)	Steps (t)	0.2364	0.2394	0.2576	0.2788
getVariable(52,k)	Steps (t-120)	0.1485	0.2485	0.3061	0.2788
getVariable(55,k)	Steps (t-105)	0.1727	0.1848	0.2212	0.3242
getVariable(58,k)	Steps (t-90)	0.1727	0.2152	0.2758	0.2212
getVariable(61,k)	Steps (t-75)	0.2061	0.2636	0.2818	0.2939
getVariable(64,k)	Steps (t-60)	0.1727	0.1576	0.2242	0.3242
getVariable(67,k)	Steps (t-45)	0.1394	0.1394	0.2455	0.2273
getVariable(70,k)	Steps (t-30)	0.1939	0.2848	0.3242	0.2909
getVariable(73,k)	Steps (t-15)	0.1667	0.2485	0.3182	0.3576
getVariable(76,k)	Cal (t)	0.2697	0.2939	0.3394	0.3515
getVariable(78,k)	Cal (t-115)	0.2455	0.2667	0.3394	0.2970
getVariable(81,k)	Cal (t-100)	0.2121	0.3030	0.2909	0.3545
getVariable(85,k)	Cal (t-80)	0.2212	0.2727	0.3485	0.2788
getVariable(89,k)	Cal (t-60)	0.2242	0.2485	0.2848	0.2758
getVariable(93,k)	Cal (t-40)	0.2606	0.2606	0.3727	0.2909
getVariable(97,k)	Cal (t-20)	0.2727	0.2758	0.3242	0.3121

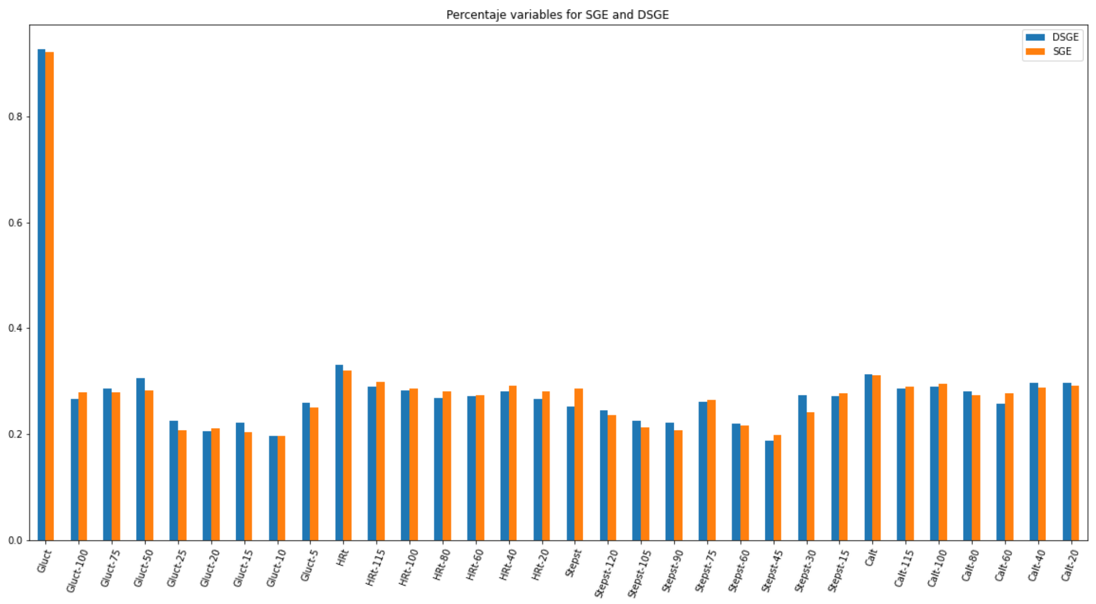


Figure 5.17: *Variable distribution comparison with SGE and DSGE*

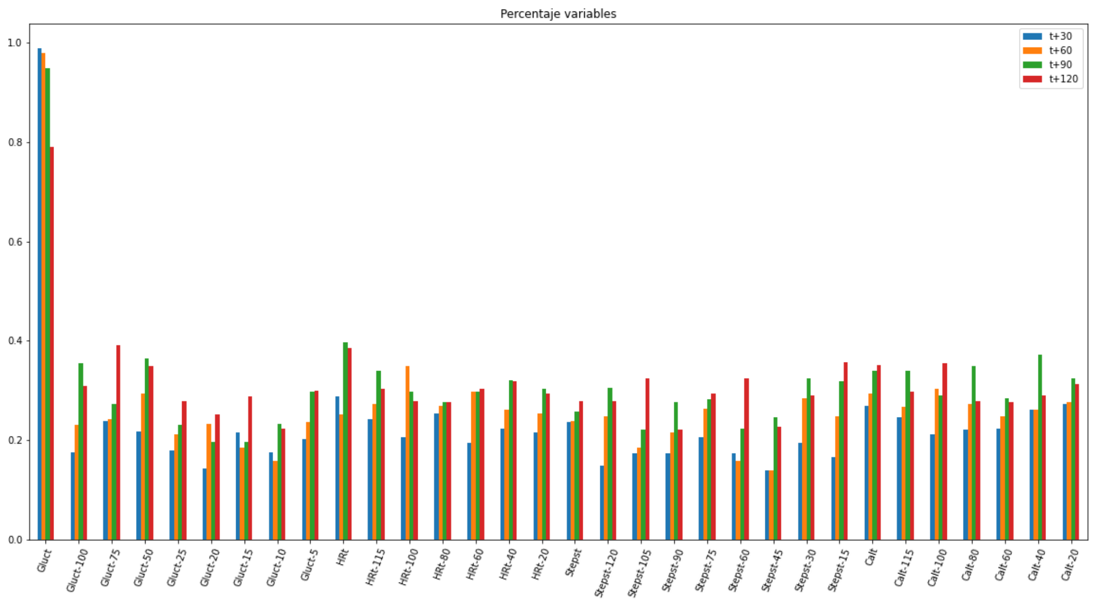


Figure 5.18: *Variable distribution comparison with t+30, t+60, t+90, t+120*

5.3 3 Classes

In this section, we will show the results obtained from performing the division in 3 classes for t+120. Hypoglycemia (Hypo) = 0, Low Non-hypoglycemia (L No-Hypo) = 1 and High Non-hypoglycemia (H No-Hypo) =2. These results can be replicated for all the other prediction horizons, but we have chosen this one since in terms of the prediction it was the hardest one and, as a result, the one with the lowest values of the Recall over the test data. The grammar used is the one shown in Fig. 3.11. We will get the Mean, Median, Std. Dev. and best fitness of the 30 executions performed, this information is shown in the table 5.15. The Recall results appear in the table 5.16 for the individual model and the table 5.17 for the general model. The comparison with other ML algorithms in the table 5.18, once more, the number shown is the average Recall of each class for all the patients by performing the training and test over the data from all the patients.

The main goal of this classification is to see if the hypoglycemia values and Non-hypoglycemia values that the algorithm is predicting wrongly are considered to be really far away from their respective classes or not. By adding a third class, we are creating a buffer that the user can use to take decisions. For a prediction horizon where our Recall is high, it doesn't make much sense to add this extra class to perform a Hypoglycemia prediction, as the added complexity will most likely hinder the prediction.

The column called "Recall Hypo + L No-Hypo" corresponds to the addition of when the value is Hypoglycemia and the algorithm predicts it as either hypoglycemia or Low No-Hypo.

The column called "Recall No-Hypo" corresponds to the values that belong to the High No-Hypo class but are classified into either their right category or the Low No-Hypo class.

The main function of these columns is to observe how many values we are actually classifying wrongly to a point where it could cause issues for the patient in case a correction is performed as a result of the prediction. This is what we are most interested in when performing the classification of 3 classes.

5.3.1 120 minutes

Table 5.15: *Statistics: Mean, Median, Standard deviation and Best Fitness for all the models we have trained*

Patient	Algorithm Type	Mean	Median	Std. Dev.	Best Fitness
HUPA001	Static	0.3824	0.3824	0.0345	0.2980
HUPA001	Dynamic	0.3732	0.3699	0.0334	0.3075
HUPA002	Static	0.4240	0.4262	0.0118	0.3989
HUPA002	Dynamic	0.4215	0.4231	0.0102	0.4005
HUPA003	Static	0.4379	0.4397	0.0200	0.3824
HUPA003	Dynamic	0.4405	0.4420	0.0173	0.4157
HUPA004	Static	0.3714	0.3772	0.0215	0.3254
HUPA004	Dynamic	0.3829	0.3830	0.0219	0.3375
HUPA005	Static	0.4071	0.4036	0.0237	0.3598
HUPA005	Dynamic	0.3973	0.3945	0.0339	0.3300
HUPA007	Static	0.3996	0.3990	0.0127	0.3528
HUPA007	Dynamic	0.4049	0.4028	0.0123	0.3852
HUPA011	Static	0.3472	0.3444	0.0168	0.3121
HUPA011	Dynamic	0.3459	0.3461	0.0157	0.3137
HUPA014	Static	0.3744	0.3784	0.0157	0.3219
HUPA014	Dynamic	0.3694	0.3746	0.0157	0.3347
HUPA015	Static	0.4073	0.4040	0.0314	0.3538
HUPA015	Dynamic	0.3991	0.3983	0.0198	0.3625
HUPA016	Static	0.4257	0.4344	0.0216	0.3877
HUPA016	Dynamic	0.4218	0.4211	0.0191	0.3856
HUPA027	Static	0.4191	0.4183	0.0117	0.3931
HUPA027	Dynamic	0.4178	0.4187	0.0122	0.3833
All	Static	0.4783	0.4772	0.0075	0.4638
All	Dynamic	0.4771	0.4780	0.0096	0.4559

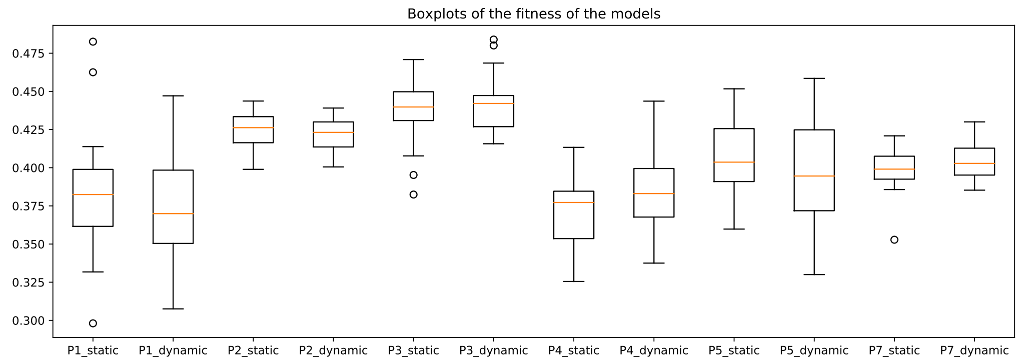


Figure 5.19: *Box plot of the fitness t+120*

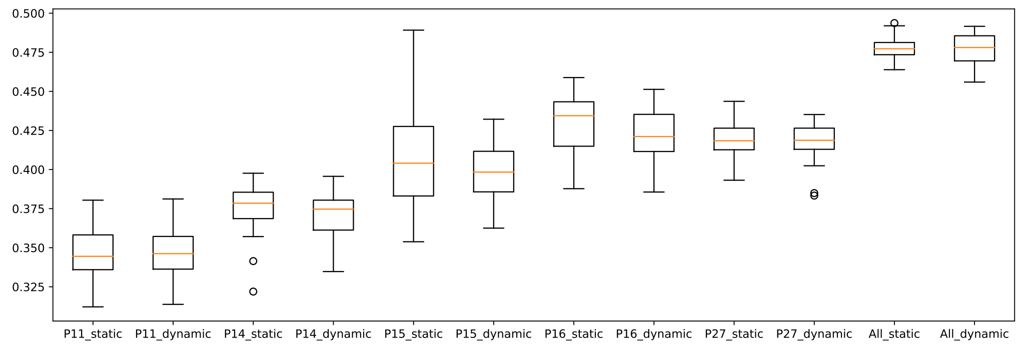


Figure 5.20: *Box plot of the fitness t+120*

Table 5.16: *Recall of the Individual model for each patient*

Patient	**Algorithm**	**Recall Hypo**	**Recall L No-Hypo**	**Recall H No-Hypo**	**Recall Hypo + L No-Hypo**	**Recall No-Hypo**
HUPA001	Static	0.5625	0.6226	0.5034	0.8125	0.8878
HUPA001	Dynamic	0.6250	0.5094	0.4874	0.7500	0.7981
HUPA002	Static	0.600	0.5163	0.6145	0.8909	0.8491
HUPA002	Dynamic	0.5818	0.5598	0.6238	0.8409	0.8529
HUPA003	Static	0.5500	0.3806	0.6361	0.8375	0.8356
HUPA003	Dynamic	0.6750	0.3613	0.5556	0.8625	0.7223
HUPA004	Static	0.7204	0.6591	0.6182	0.9355	0.8533
HUPA004	Dynamic	0.7527	0.5455	0.7388	0.9140	0.8905
HUPA005	Static	0.4118	0.466	0.6447	0.6765	0.8313
HUPA005	Dynamic	0.4412	0.5922	0.5589	0.8236	0.7475
HUPA007	Static	0.5455	0.5315	0.7626	0.6566	0.8700
HUPA007	Dynamic	0.4848	0.5766	0.6778	0.6868	0.8561
HUPA011	Static	0.4839	0.6809	0.6482	0.9678	0.9091
HUPA011	Dynamic	0.5484	0.4574	0.6798	0.8065	0.9387
HUPA014	Static	0.7021	0.5526	0.5629	0.9574	0.7686
HUPA014	Dynamic	0.4681	0.5789	0.5314	0.9149	0.7171
HUPA015	Static	0.4872	0.3111	0.5513	0.7180	0.7968
HUPA015	Dynamic	0.4872	0.4556	0.5010	0.8462	0.8390
HUPA016	Static	0.7688	0.4545	0.542	0.9306	0.7630
HUPA016	Dynamic	0.8266	0.4156	0.4753	0.9422	0.7210
HUPA027	Static	0.5906	0.5667	0.6935	0.8859	0.9228
HUPA027	Dynamic	0.6913	0.5278	0.6776	0.8524	0.8865

Table 5.17: *Recall of the General model for each patient*

Patient	**Algorithm**	**Recall Hypo**	**Recall L No-Hypo**	**Recall H No-Hypo**	**Recall Hypo + L No-Hypo**	**Recall No-Hypo**
HUPA001	Static	0.0	0.4151	0.6759	0.6250	0.9498
HUPA001	Dynamic	0.3125	0.2453	0.7781	0.5625	0.9012
HUPA002	Static	0.6364	0.5054	0.4898	0.9546	0.8306
HUPA002	Dynamic	0.6455	0.4511	0.5289	0.9591	0.8417
HUPA003	Static	0.5125	0.4516	0.5578	0.8417	0.8417
HUPA003	Dynamic	0.3250	0.4452	0.5805	0.8000	0.8605
HUPA004	Static	0.5591	0.2955	0.7612	0.8279	0.9105
HUPA004	Dynamic	0.6452	0.2500	0.7886	0.7850	0.9017
HUPA005	Static	0.4412	0.3786	0.6497	0.6765	0.9052
HUPA005	Dynamic	0.2353	0.5049	0.6966	0.6765	0.9411
HUPA007	Static	0.3535	0.6486	0.7014	0.7070	0.8958
HUPA007	Dynamic	0.5354	0.4595	0.7368	0.7071	0.8861
HUPA011	Static	0.1613	0.6064	0.6492	0.7742	0.9427
HUPA011	Dynamic	0.3226	0.5000	0.7075	0.7742	0.9032
HUPA014	Static	0.4894	0.2368	0.7914	0.6596	0.8952
HUPA014	Dynamic	0.1489	0.2632	0.8143	0.4892	0.9191
HUPA015	Static	0.0513	0.4333	0.6358	0.7436	0.9195
HUPA015	Dynamic	0.4103	0.2444	0.6710	0.7180	0.9104
HUPA016	Static	0.6763	0.3377	0.6000	0.8902	0.7283
HUPA016	Dynamic	0.5318	0.5390	0.6160	0.8671	0.8740
HUPA027	Static	0.6174	0.5444	0.5573	0.9597	0.8853
HUPA027	Dynamic	0.5436	0.7000	0.5823	0.9462	0.9206

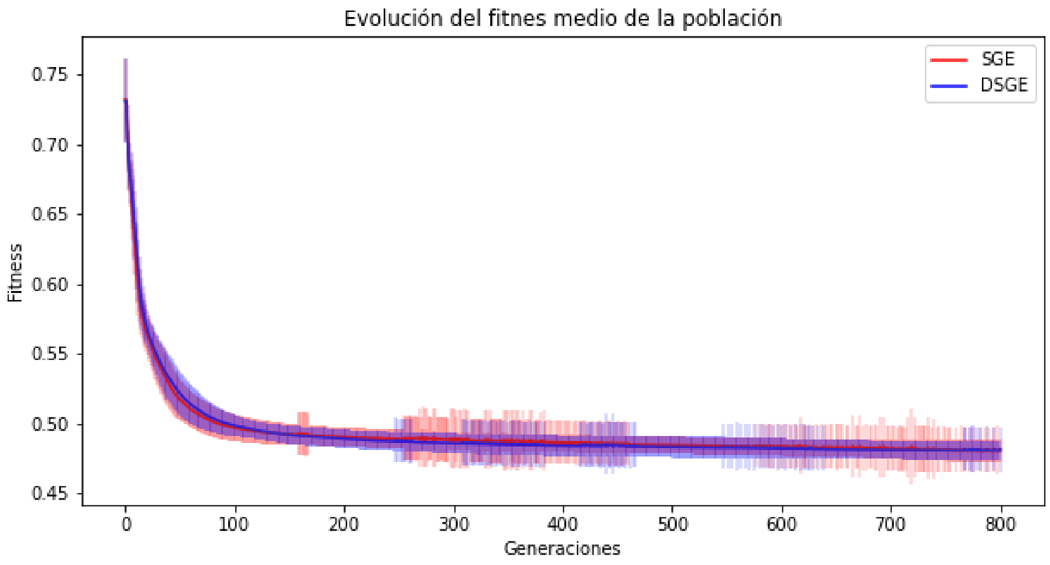


Figure 5.21: *Evolution of the average population fitness in t+120*

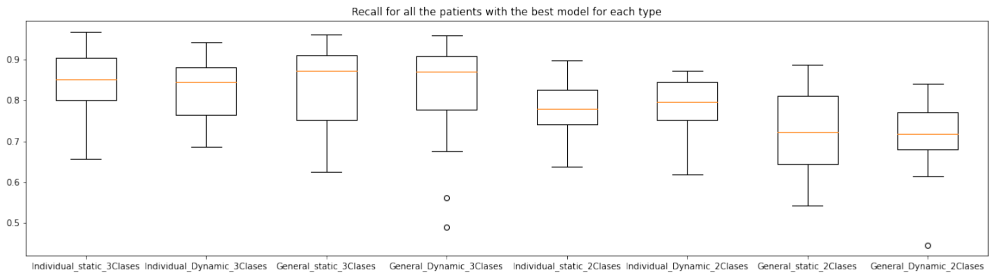


Figure 5.22: *Boxplots of the Recall of the best execution in t+120 for 3 Clases and 2 Clases*

Comparison with other Machine Learning algorithms

Table 5.18: *Comparison of the recall between different ML algorithms*

Algorithm	**Recall Low No-Hypo**	**Recall High No-Hypo**	**Recall High No-Hypo**	**Recall Hypo + Low No-Hypo**	**Recall No-Hypo**
Gradient boosting	0.6676	0.5538	0.6770	0.8718	0.8797
Gradient boosting with Cross val	0.6676	0.5538	0.6770	0.8718	0.8797
Logistic Regression	0.6727	0.2313	0.6693	0.8357	0.7979
Random forest classifier	0.9378	0.8266	0.7371	0.9704	0.8967
Random forest Regresor	0.8012	0.9535	0.4518	0.9999	0.9936
Gaussian Naive Bayes	0.7054	0.1965	0.5004	0.8491	0.7097
Bernoulli Naive Bayes	0.5657	0.2943	0.2710	0.7505	0.5121
Decision Tree Classifier (6 depth)	0.5076	0.5854	0.5587	0.9194	0.8572
Ada Boost Classifier	0.8888	0.7703	0.6184	0.9468	0.8451
Bagging Classifier	0.9296	0.8175	0.7212	0.9596	0.8914
Extra Trees Classifier	0.9551	0.8349	0.7615	0.9551	0.9055
SGD Classifier	0.6564	0.1426	0.7058	0.7746	0.7942
MLP Classifier	0.7940	0.1393	0.6249	0.8327	0.6859
Dynamic SGE	0.5097	0.4701	0.6926	0.8318	0.8996
Static SGE	0.5321	0.4677	0.6512	0.8522	0.8902

5.3.2 Results for 3 Classes

From this experimental example, we don't really obtain much better results than in the case of the 2 classes. The individual class Recalls are much lower since performing a 3-class classification is harder than when we just have 2 classes, the intermediate class is making it harder to predict the hypoglycemia state, as shown in Tab. 5.16 and Tab. 5.17. In terms of the columns of "Recall Hypo + L No-Hypo" and "Recall No-Hypo" we can see a slight improvement in comparison to the Hypoglycemia prediction for the 2 classes. However, we are still making an erroneous prediction since "Recall Hypo + L No-Hypo" corresponds to the addition of the Hypoglycemic values correctly classified and incorrectly classified into the second class, and "Recall No-Hypo" the Non-Hypoglycemic values over 90 mg/dl classified correctly and incorrectly into the second class. This comparison is shown in Fig. 5.22 where we can observe the average Recall values of 3 Classes, considering these previously described columns, and 2 Classes for t+120. The average Recall of the fusion of the classes rises to a value between 80% and 90%.

If we compare our prediction with other ML algorithms we obtain similar results than for the 2 classes, the ensemble methods (Ada Boost, Bagging Classifier, Random Forest, and Extra Trees Classifier) obtain better results on the Recall, and our algorithms give similar results to Logistic Regression or Gradient Boosting.

The solutions obtained have this form:

```
if((((getVariable(35,k)*getVariable(61,k)*getVariable(58,k)-getVariable(31,k
))<=(324.978-getVariable(76,k)*96.752))&&((getVariable(28,k)+getVariable(47,k)
-getVariable(22,k))>(getVariable(73,k)-2.471))))
{result=0:}
else{
if(((getVariable(1,k)*getVariable(89,k))<805.928-(getVariable(89,k)+
getVariable(70,k)))){result=1:}
else{result=2:}}
```

That with the variables names translated would be:

```
if((((HR (t-80) * Steps(t-75) * Steps(t-100) - HR (t-100)
)<=(324.978- Cal (t) *96.752))&&(( HR (t-115) )+ HR(t-20) - Gluc (t-25) )
>(Steps (t-15) -2.471))))
{result=0:}
else{
if((( Gluc(t) * Cal (t-60) )<805.928-( Cal (t-60) +Steps(t-30) ))){result=1:}
else{result=2:}}
```

5.3.3 Used variables

The percentage of the variables being used by both algorithms is still very similar for the two of them as can be seen in the table 5.19 and in Fig. 5.23. The value with the highest percentage is Glucose (t) and the other variables have a used percentage of around 30% to 40%. The less-used variable in both algorithms is Glucose (t-20) with a percentage of 23.6% and 25.7% respectively.

If we compare it to the results obtained for t+120 for just two classes we can see some differences in some variables, such as Steps (t-90) or HR (t-115) that are used much more often in the 3 class version. It can also be seen that the average of the percentages are a bit higher in the 3 class solutions, which is probably due to the fact that the solutions are much longer than for two classes so more variables can be used per solution.

Table 5.19: *Percentage of variable involvement in the final solutions for all executions of 30 runs*

Variable	**Name**	**Static t +120**	**Dynamic t +120**
getVariable(1,k)	Gluc (t)	0.8273	0.7939
getVariable(6,k)	Gluc (t-100)	0.3909	0.4182
getVariable(11,k)	Gluc (t-75)	0.3455	0.3364
getVariable(16,k)	Gluc (t-50)	0.2848	0.3242
getVariable(21,k)	Gluc (t-25)	0.3091	0.2818
getVariable(22,k)	Gluc (t-20)	0.2364	0.2576
getVariable(23,k)	Gluc (t-15)	0.3000	0.2576
getVariable(24,k)	Gluc (t-10)	0.3030	0.3030
getVariable(25,k)	Gluc (t-5)	0.4364	0.4121
getVariable(26,k)	HR (t)	0.3667	0.3758
getVariable(28,k)	HR (t-115)	0.3939	0.4121
getVariable(31,k)	HR (t-100)	0.4030	0.3515
getVariable(35,k)	HR (t-80)	0.3576	0.3303
getVariable(39,k)	HR (t-60)	0.3152	0.2939
getVariable(43,k)	HR (t-40)	0.3061	0.2788
getVariable(47,k)	HR (t-20)	0.3364	0.3000
getVariable(51,k)	Steps (t)	0.4212	0.3848
getVariable(52,k)	Steps (t-120)	0.3485	0.3030
getVariable(55,k)	Steps (t-105)	0.3424	0.2939
getVariable(58,k)	Steps (t-90)	0.3364	0.3576
getVariable(61,k)	Steps (t-75)	0.2667	0.3121
getVariable(64,k)	Steps (t-60)	0.3818	0.3485
getVariable(67,k)	Steps (t-45)	0.3242	0.2939
getVariable(70,k)	Steps (t-30)	0.2727	0.3091
getVariable(73,k)	Steps (t-15)	0.3576	0.3273
getVariable(76,k)	Cal (t)	0.4212	0.3788
getVariable(78,k)	Cal (t-115)	0.3242	0.3545
getVariable(81,k)	Cal (t-100)	0.3879	0.3182
getVariable(85,k)	Cal (t-80)	0.3394	0.2970
getVariable(89,k)	Cal (t-60)	0.3333	0.3212
getVariable(93,k)	Cal (t-40)	0.3576	0.3515
getVariable(97,k)	Cal (t-20)	0.3424	0.3152

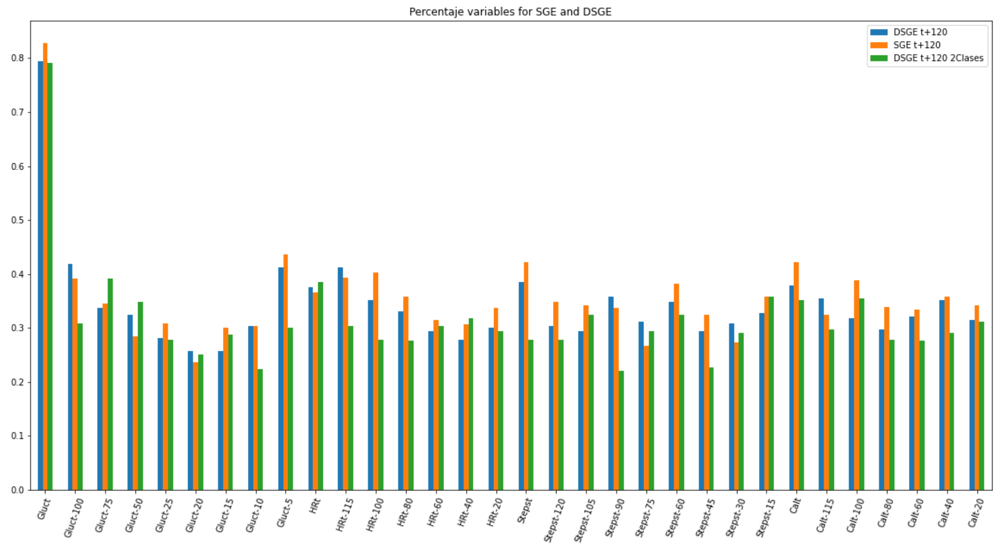


Figure 5.23: *Variable distribution for t+120 for SGE and DSGE*

5.4 Future Work

This study can be expanded to try to predict other states, not only hypoglycemia, such as hyperglycemia. This would be performed by using other data elements such as carbohydrates, insulin, stress data, etc. Additionally, different class divisions can be made depending on what each patient needs, for example, some patients don't correct their glucose value until it drops to 65 mg/dl, or they choose to correct it when it reaches higher values.

The algorithms developed should be tested on more patients over a longer period of time, for the project we have only included data of 11 of the 38 patients we currently have at our disposal, in the near future we will make a study that includes all of the patients. We should also study what conditions can be met to create a general model with what type of patients. In our study we have used the data for all the patients to test all of them, however, it can be seen in the results that some patients work better with some data over others. If we can divide the data into different subsets we could predict for a patient that we don't have data on, if we choose a subset of other patient's data that are similar to the new patient. This generates a general model that uses data that does not belong to a patient but which would yield good results because their data variation is more similar.

5.5 Diffusion

Considering the impact that the project could have, we have decided to publish the JECO library and PancreasModelTools on the GitHub platform, both projects are under the Adaptive and Bioinspired System Group.

The work of developing this project has allowed us to create a paper with the title "Evolving Classification Rules for Predicting Hypoglycemia Events" and accepted for the 2022 IEEE World Congress on Computational Intelligence.

Bibliography

[1] Predicting and Preventing Nocturnal Hypoglycemia in Type 1 Diabetes Using Big Data Analytics and Decision Theoretic Analysis. *Diabetes technology & therapeutics*, 22(11):801–811, nov 2020.

[2] Arthur Bertachi, Clara Viñals, Lyvia Biagi, Ivan Contreras, Josep Vehí, Ignacio Conget, and Marga Giménez. Prediction of nocturnal hypoglycemia in adults with type 1 diabetes under multiple daily injections using continuous glucose monitoring and physical activity monitor. *Sensors*, 20(6), 2020.

[3] Urvesh Bhowan, Mengjie Zhang, and Mark Johnston. Genetic programming for classification with unbalanced data. In Anna Isabel Esparcia-Alcázar, Anikó Ekárt, Sara Silva, Stephen Dignum, and A. Şima Uyar, editors, *Genetic Programming*, pages 1–13, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[4] Mads Bisgaard Bengtsen and Niels Møller. Mini-review: Glucagon responses in type 1 diabetes – a matter of complexity. *Physiological Reports*, 9(16):e15009, 2021.

[5] Carlos Cervigón, J. Manuel Velasco, Clara Burgos-Simón, Rafael J. Villanueva, and J. Ignacio Hidalgo. Probabilistic fitting of glucose models with real-coded genetic algorithms. pages 736–743, 2021.

[6] Philip E. Cryer, Stephen N. Davis, and Harry Shamoon. Hypoglycemia in Diabetes . *Diabetes Care*, 26(6):1902–1912, 06 2003.

[7] Darpit Dave, Daniel J. DeSalvo, Balakrishna Haridas, Siripoom McKay, Akhil Shenoy, Chester J. Koh, Mark Lawley, and Madhav Erraguntla. Feature-based machine learning model for real-time hypoglycemia prediction. *Journal of Diabetes Science and Technology*, 15(4):842–855, 2021. PMID: 32476492.

[8] Ivanoe De Falco, Antonio Della Cioppa, Tomas Koutny, Michal Krcma, Umberto Scafuri, and Ernesto Tarantino. Genetic programming-based induction of a glucose-dynamics model for telemedicine. *Journal of Network and Computer Applications*, 119:1–13, 2018.

[9] Amparo Güemes, Giacomo Cappon, Bernard Hernandez, Monika Reddy, Nick Oliver, Pantelis Georgiou, and Pau Herrero. Predicting quality of overnight glycaemic control in type 1 diabetes using binary classifiers. *IEEE Journal of Biomedical and Health Informatics*, 24(5):1439–1446, 2020.

[10] J. Ignacio Hidalgo, J. Manuel Colmenar, Gabriel Kronberger, Stephan M. Winkler, Oscar Garnica, and Juan Lanchares. Data based prediction of blood glucose concentrations using evolutionary methods. *Journal of Medical Systems*, 41, 2017.

[11] J. Ignacio Hidalgo, J. Manuel Colmenar, José L. Risco-Martin, Alfredo Cuesta-Infante, Esther Maqueda, Marta Botella, and José Antonio Rubio. Modeling glycemia in humans by means of grammatical evolution. *Applied Soft Computing*, (20):40–53, 2014.

[12] J. Ignacio Hidalgo, J. Manuel Colmenar, J. Manuel Velasco, Gabriel Kronberger, Stephan M. Winkler, Oscar Garnica, and Juan Lanchares. *Identification of Models for Glucose Blood Values in Diabetics by Grammatical Evolution*, pages 367–393. Springer International Publishing, Cham, 2018.

[13] J. Ignacio Hidalgo, Esther Maqueda, Jose L. Risco-Martan, Alfredo Cuesta-Infante, J. Manuel Colmenar, and Javier Nobel. glucmodel: A monitoring and modeling system for chronic diseases applied to diabetes. *Journal of Biomedical Informatics*, 48:183 – 192, 2014.

[14] Jose Ignacio Hidalgo, Jose Manuel Velasco, Juan Lanchares, Sergio Contador, and Oscar Garnica. *An analysis of solutions based on Genetic Programming to solve problems of*

symbolic regression of data from continuous glucose monitoring. AEPIA, Granada, Spain, 2018.

[15] Christina-Maria Kastorini, George Papadakis, Haralampos J. Milionis, Kallirroi Kalantzi, Paolo-Emilio Puddu, Vassilios Nikolaou, Konstantinos N. Vemmos, John A. Goudevenos, and Demosthenes B. Panagiotakos. Comparative analysis of a-priori and a-posteriori dietary patterns using state-of-the-art classification algorithms: A case/case-control study. *Artificial Intelligence in Medicine*, 59(3):175–183, 2013.

[16] Nuno Lourenço, J. Manuel Colmenar, J. Ignacio Hidalgo, and Óscar Garnica. Structured grammatical evolution for glucose prediction in diabetic patients. page 1250–1257, 2019.

[17] Nuno Lourenço, Joaquim Ferrer, Francisco Pereira, and Ernesto Costa. A comparative study of different grammar-based genetic programming approaches. pages 311–325, 03 2017.

[18] Mirko Messori, Chiara Toffanin, Simone Del Favero, Giuseppe De Nicolao, Claudio Cobelli, and Lalo Magni. Model individualization for artificial pancreas. *Computer Methods and Programs in Biomedicine*, 171:133–140, 2016.

[19] Clara Mosquera-Lopez, Robert Dodier, Nichole Tyler, Navid Resalat, and Peter Jacobs. Leveraging a big dataset to develop a recurrent neural network to predict adverse glycemic events in type 1 diabetes. *IEEE Journal of Biomedical and Health Informatics*, pages 1–1, 2019.

[20] Omer Mohammed Mujahid, Iván Contreras, and Josep Vehí. Machine learning techniques for hypoglycemia prediction: Trends and challenges. *Sensors (Basel, Switzerland)*, 21, 2021.

[21] Silvia Oviedo, Ivan Contreras, Carmen Quirós, Marga Giménez, Ignacio Conget, and

Josep Vehi. Risk-based postprandial hypoglycemia forecasting using supervised learning. *International Journal of Medical Informatics*, 126:1–8, 2019.

[22] Jean-Eudes Ranvier, Fabien Dubosson, Jean-Paul Calbimonte, and Karl Aberer. Detection of hypoglycemic events through wearable sensors. CEUR-WS, 2016.

[23] Ravi Reddy, Navid Resalat, Leah M. Wilson, Jessica R. Castle, Joseph El Youssef, and Peter G. Jacobs. Prediction of hypoglycemia during aerobic exercise in adults with type 1 diabetes. *Journal of Diabetes Science and Technology*, 13(5):919–927, 2019. PMID: 30650997.

[24] Phyo Phyo San, Sai Ho Ling, and Hung T. Nguyen. Deep learning framework for detection of hypoglycemic episodes in children with type 1 diabetes. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3503–3506, 2016.

[25] Shariq I. Sherwani, Haseeb A. Khan, Aishah Ekhzaimy, Afshan Masood, and Meena K. Sakharkar. Significance of hba1c test in diagnosis and prognosis of diabetic patients. *Biomarker Insights*, 11:BMI.S38440, 2016. PMID: 27398023.

[26] Josep Vehí, Iván Contreras, Silvia Oviedo, Lyvia Biagi, and Arthur Bertachi. Prediction and prevention of hypoglycaemic events in type-1 diabetic patients using machine learning. *Health Informatics Journal*, 26(1):703–718, 2020. PMID: 31195880.

[27] Jose Velasco, Oscar Garnica, Juan Lanchares, Marta Botella, and Ignacio Hidalgo. Combining data augmentation, edas and grammatical evolution for blood glucose forecasting. *Memetic Computing*, 10, 06 2018.

Appendix A

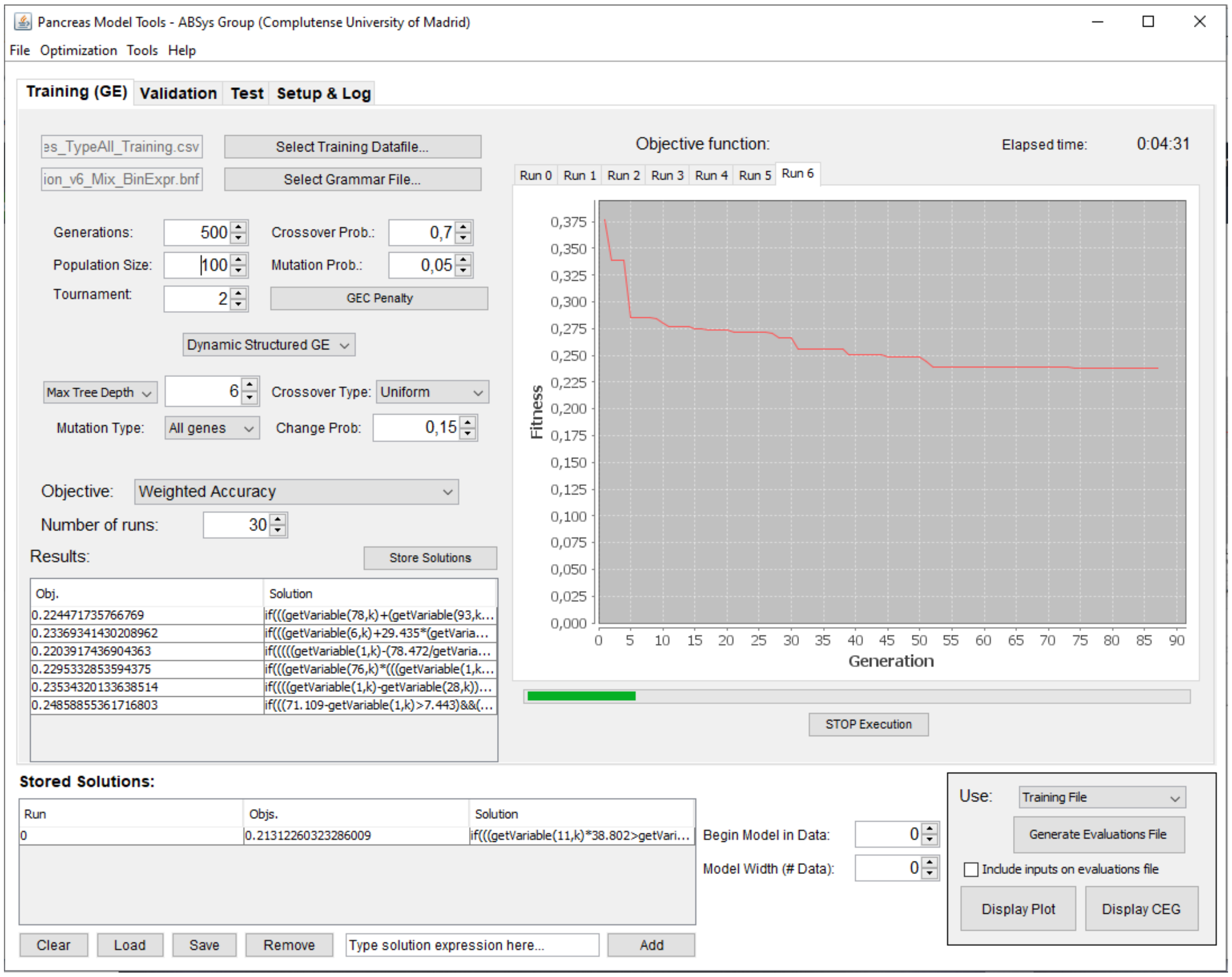
JECO and Pancreas Model Tools

JECO is the library where the two algorithms have been programmed in, as well as the mutation, crossover operators and genetic algorithm used. In JECO we have defined two abstract classes whose "Evaluate" method that computes the fitness and depends on the problem remains abstract. These classes are called "AbstractProblemDSGE" and "AbstractProblemSSGE". Other utility classes have been implemented to perform some tests and to implement the main classes, they can all be found in the jeco.core.algorithm.sge package. The grammar reader has been extended to add the functionality of the SGE, and it can be found in the jeco.core.util.bnf package.

The "Evaluate" method is defined in the Pancreas Model Tools project, which is the tool used to obtain the results shown. It is responsible for loading and dealing with the patient's data and specifying the "Evaluate" function. This project has a user interface that is used to set the input parameters (crossover, mutation, number of generations, population, etc) for training and testing. Pancreas Model Tools also has extra logging capabilities that allow the intermediate data to be saved into a file, allowing us to save the fitness of each individual, their phenotype, and genotype on each generation. Additionally, the final models can be directly tested by using an independent testing file and the confusion matrix obtained with the tools.

Pancreas Model Tools uses a graphical interface to set the parameters and also has an option to save and load a properties file. The graphical interface can be seen in Fig A.1.

A version without a user interface that works over a directory also exists to generate results on a bigger scale using a set of training/test files, in this case, the parameters are set by loading a properties file that specifies all of the information needed by the algorithms, as well as the directory over which to perform the training/test.


Pancreas Model Tools - ABSys Group (Complutense University of Madrid)
File Optimization Tools Help
Training (GE) Validation Test Setup & Log
Select Training Datafile...
Select Grammar File...
GEC Penalty
Store Solutions
Objective function:
Run 0 Run 1 Run 2 Run 3 Run 4 Run 5 Run 6
Fitness
Generation
STOP Execution
Generate Evaluations File
Display Plot
Display CEG
Clear Load Save Remove Add

es_TypeAll_Training.csv

ion_v6_Mix_BinExpr.bnf

Generations: 500

Crossover Prob.: 0,7

Population Size: 100

Mutation Prob.: 0,05

Tournament: 2

Dynamic Structured GE

Max Tree Depth: 6

Crossover Type: Uniform

Mutation Type: All genes

Change Prob: 0,15

Objective: Weighted Accuracy

Number of runs: 30

Elapsed time: 0:04:31

Results:

Obj.	Solution
0.224471735766769	if(((getVariable(78,k)+(getVariable(93,k...
0.23369341430208962	if(((getVariable(6,k)+29.435*(getVaria...
0.2203917436904363	if(((((getVariable(1,k)-(78.472/getVaria...
0.2295332853594375	if(((getVariable(76,k)*(((getVariable(1,k...
0.23534320133638514	if((((getVariable(1,k)-getVariable(28,k))...
0.24858855361716803	if(((71.109-getVariable(1,k)>7.443)&&(...

Stored Solutions:

Run	Objs.	Solution
0	0.21312260323286009	if(((getVariable(11,k)*38.802>getVari...

Begin Model in Data: 0

Model Width (# Data): 0

Use: Training File

Include inputs on evaluations file [ ]

Type solution expression here...

Figure A.1: *Pancreas Model Tools GUI*