### **Titanic Voyager Classification Using Grammatical Evolution.**

### 

### **Introduction**

The issue depicts a futuristic Titanic Voyage that encounters an accident (much like the actual Titanic) and some of the passengers are taken to a different dimension.

Making a classifier that can determine whether a passenger was taken outside the spaceship when the accident occurred is the goal here. The Passenger ID must be given with this in a TRUE/FALSE format.

I and Rishi employed grammatical evolution to solve this issue because it appeared too complicated to use genetic programming.

We experimented with various features as we would in any machine learning scenario before settling on 10 and 13 features at once. The 13-feature strategy produced better results.

The final result we got was a test accuracy for 0.79106 for a min fitness of 0.2205

### **Data**

Original Data

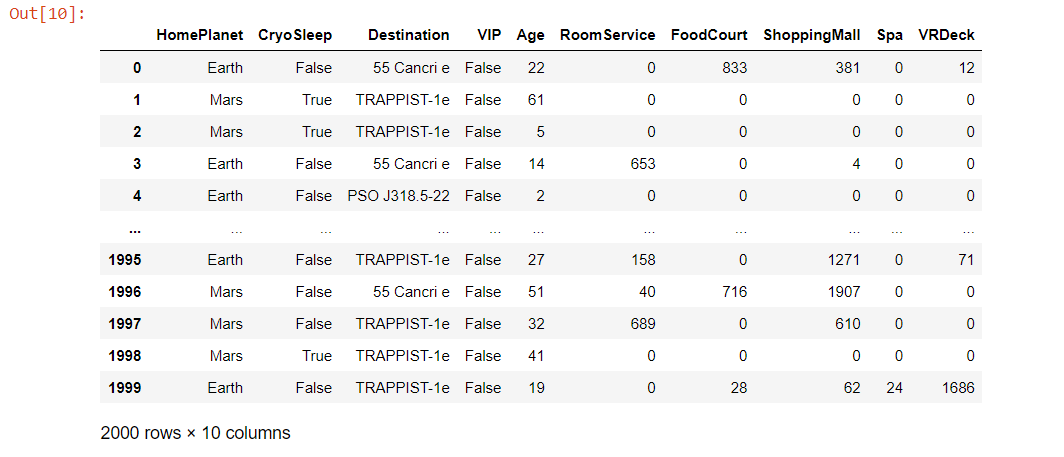
| PassengerID | Number from 0 to 1999 |
| --- | --- |
| HomePlanet | Categorical: Earth, Mars or Europa |
| CryoSleep | Boolean: True/False for if a passenger was sleeping |
| Destination | Categorical: 55 Cancri e, TRAPPIST-1e, PSO J318.5-22 |
| Age | Numerical: 0-79 years. Mean = 28.56 |
| VIP | Boolean: True/False for if a passenger was a VIP or not |
| RoomService | Numerical: 0-6,899. Mean = 213 |
| FoodCourt | Numerical: 0-27,723. Mean = 498 |
| ShoppingMall | Numerical: 0-10,424. Mean = 510 |
| Spa | Numerical: 0-18,572. Mean = 342 |
| VRDeck | Numerical: 0-14,485. Mean = 269 |
| Name | Categorical |
| Transported | Output: Boolean: True/False for if a passenger was transported or not |

The dataset consists of 2,000 rows/passengers and the 13 columns mentioned above. The output column, or the one we are attempting to predict, is the last column, Transported.

For 10 features and 13 features, in which 3 additional features were produced from the original dataset, we used two notebooks and worked on them concurrently.

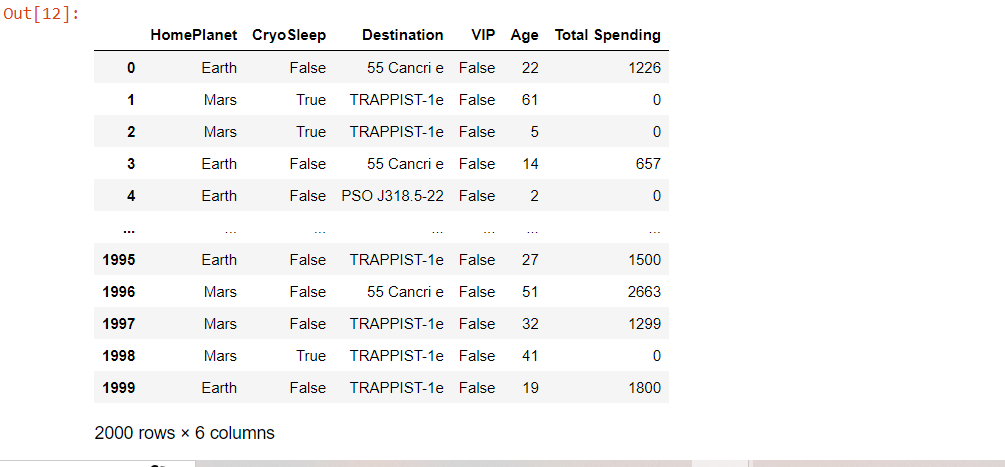
Case 1: **10 features**

For the 10 features notebook, we kept 10 features from the original 13 feature dataset that we got. We removed the Transported feature as it was the data to be predicted. We also removed the Name and PassangerID as the fact that the data was redundant in this case.



| HomePlanet | Categorical: Earth, Mars or Europa |
| --- | --- |
| CryoSleep | Boolean: True/False for if a passenger was sleeping |
| Destination | Categorical: 55 Cancri e, TRAPPIST-1e, PSO J318.5-22 |
| Age | Numerical: 0-79 years. Mean = 28.56 |
| VIP | Boolean: True/False for if a passenger was a VIP or not |
| RoomService | Numerical: 0-6,899. Mean = 213 |
| FoodCourt | Numerical: 0-27,723. Mean = 498 |
| ShoppingMall | Numerical: 0-10,424. Mean = 510 |
| Spa | Numerical: 0-18,572. Mean = 342 |
| VRDeck | Numerical: 0-14,485. Mean = 269 |

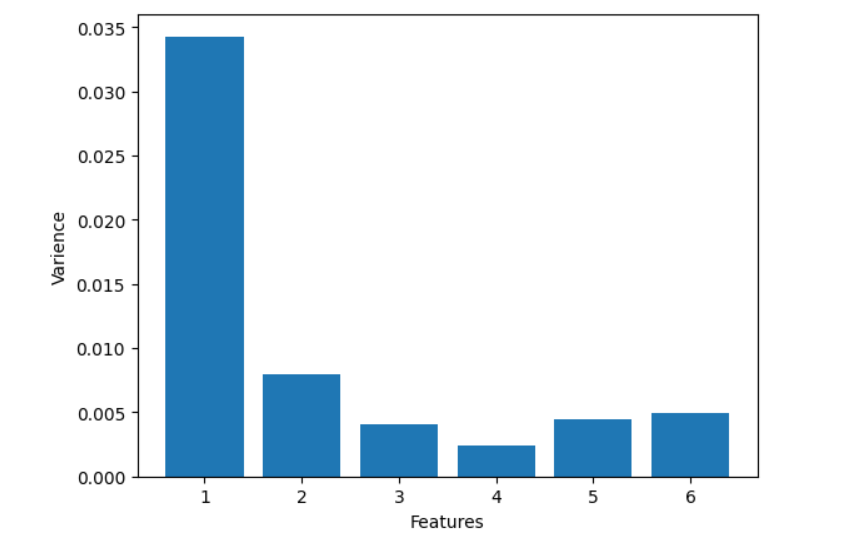
Even the concept of having only six feature was tested. In order to see what happens when we try to reduce the original number of features, we took Total Spending as a Summation of RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck and dropped the RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck. However, it didn't turn out to render good results, so the idea was dropped. The concept of choosing six feature and eliminating the others was inspired by the Titanic's original case scenario, in which the wealthy and powerful were carried first. For this reason, Total Spending was retained as a feature.



To check if all the features had contribution to the classifier, we implemented

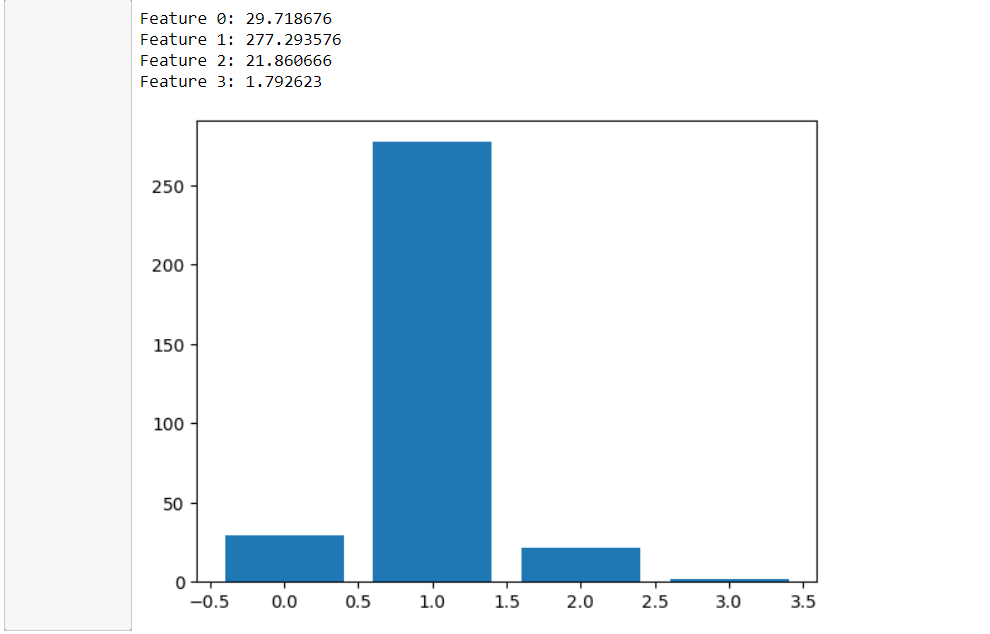
1. Variance test on the Numerical data

We got a decent score for each of the features and hence all could be considered.



1. Chi-Square test on the Categorical data

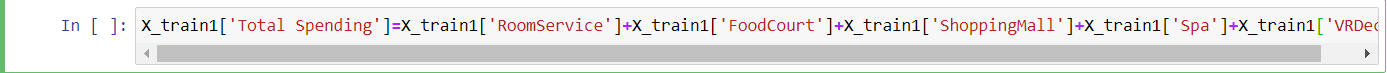
We got a decent score for each of the features besides for ‘VIP’ which is Boolean and can be considered.



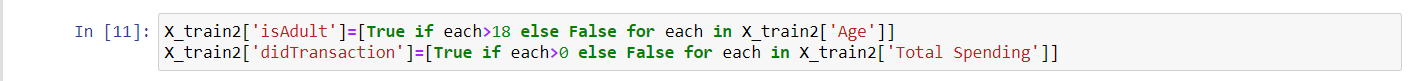
Case 2: **13 features**

To boost the number of features being trained for the 13 feature notebook, we added 3 additional features to the initial 10 feature set. The fresh elements were

1. Total Spending - A measurement of how much money each passenger spent overall on the ship, similar to the case where the wealthy and powerful were transported first.



1. isAdult - This function determines whether the passenger is an adult or not, simulating the case where children would be transferred first.

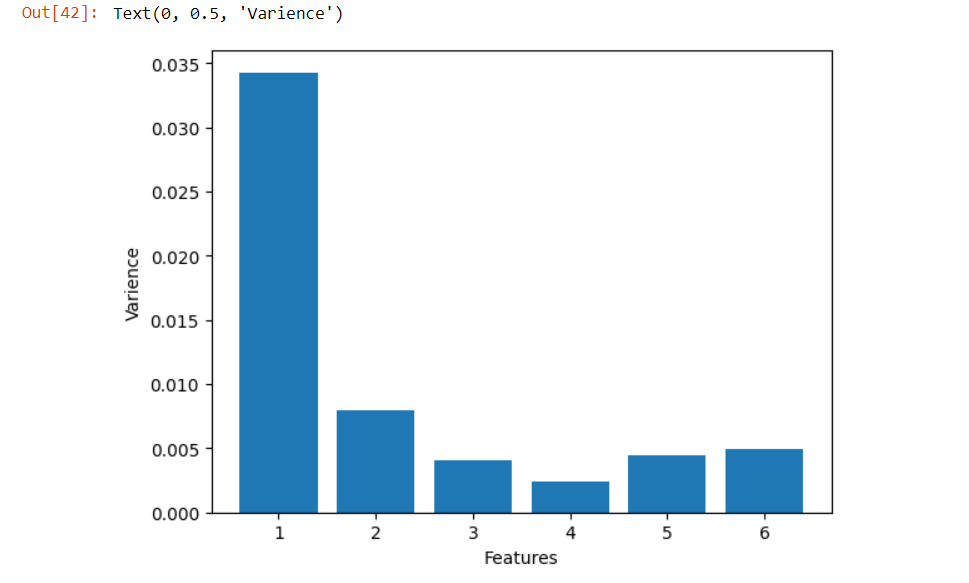


1. didTransaction is a check that determines whether or not a passenger spent money while aboard the ship, simulating the case when the wealthy and powerful were carried first.

| HomePlanet | Categorical: Earth, Mars or Europa |
| --- | --- |
| CryoSleep | Boolean: True/False for if a passenger was sleeping |
| Destination | Categorical: 55 Cancri e, TRAPPIST-1e, PSO J318.5-22 |
| Age | Numerical: 0-79 years. Mean = 28.56 |
| VIP | Boolean: True/False for if a passenger was a VIP or not |
| RoomService | Numerical: 0-6,899. Mean = 213 |
| FoodCourt | Numerical: 0-27,723. Mean = 498 |
| ShoppingMall | Numerical: 0-10,424. Mean = 510 |
| Spa | Numerical: 0-18,572. Mean = 342 |
| VRDeck | Numerical: 0-14,485. Mean = 269 |
| Total Spending | Numerical: 0-31,074. Mean= 1489  Summation of RoomService,FoodCourt,ShoppingMall,Spa and VRDeck |
| isAdult | Boolean: True/False for if a passenger is an adult, i.e age>18 |
| didTransaction | Boolean: True/False for if a passenger did a transaction during the voyage, i.e Total Spending==0 or not |

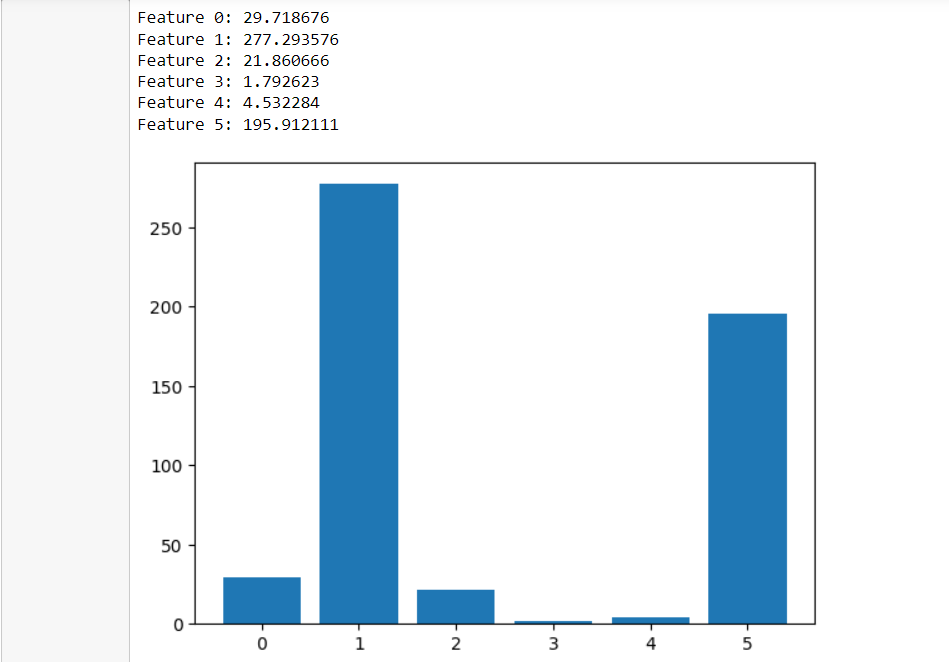
To check if all the features had contribution to the classifier, we implemented

1. **Variance test** on the Numerical data

We got a decent score for each of the features and hence all could be considered

1. **Chi-Square test** on the Categorical data

We got a decent score for each of the features besides for ‘VIP’ which is Boolean and can be considered.



**Preprocessing:**

For Preprocessing the dataset we use sklearn’s library to perform transformation on the dataset by making a pipeline that uses

* Standard Scaler for the Numerical data
* One Hot Encoding for the Categorical data



Advantage of using a pipeline was to streamline the process, the output is already in the form of a numpy array hence saving many transformations that need to be handled before. This was done for both the 10 and 13 feature dataset with a small change in the column fed to the pipeline.

### **Approach**

Since the problem is rather difficult and using genetic programming would not produce good results in this situation, we decided to move on using grammatical evolution.

Since training for lengthy hours was not possible and it was fairly slow, we made the decision to migrate the code from colab to a local Jupyter notebook. The local grape directory needed to be modified, but once done, it was simple to use. For both the 10 and 13 feature versions of the dataset, the objective was to run for 10,000 and 100,000 individuals with greater and lower generational representations, respectively.

**Grammar**

A modified version of the heartdiseas.bnf grammar was employed. Each characteristic was divided into boolean and non-boolean categories, and based on the features, either log, numerical, or conditional branches may use them.

Grammar for 10 features:

<log\_op> ::= <conditional\_branches> | and\_(<log\_op>,<log\_op>) | or\_(<log\_op>,<log\_op>) | not\_(<log\_op>) | <boolean\_feature> | nand\_(<log\_op>,<log\_op>) | nor\_(<log\_op>,<log\_op>) | xor\_(<log\_op>,<log\_op>) | not\_(<boolean\_feature>)

<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<o> ::= +|-|\*

<conditional\_branches> ::= less\_than\_or\_equal(<num\_op>,<num\_op>) | greater\_than\_or\_equal(<num\_op>, <num\_op>)

<num\_op> ::= add(<num\_op>,<num\_op>) | sub(<num\_op>,<num\_op>) | mul(<num\_op>,<num\_op>) | pdiv(<num\_op>,<num\_op>) | <nonboolean\_feature> | <nonboolean\_feature> <o> <nonboolean\_feature> | <nonboolean\_feature> | <nonboolean\_feature> <o> <nonboolean\_feature>

<boolean\_feature> ::= x[6]|x[7]|x[8]|x[9]|x[10]|x[11]|x[12]|x[13]|x[14]|x[15]

<nonboolean\_feature> ::= x[0]|x[1]|x[2]|x[3]|x[4]|x[5]|<c><c>.<c><c>|mean\_(<nonboolean\_feature>)

Grammar for 13 features:

<log\_op> ::= <conditional\_branches> | and\_(<log\_op>,<log\_op>) | or\_(<log\_op>,<log\_op>) | not\_(<log\_op>) | <boolean\_feature> | nand\_(<log\_op>,<log\_op>) | nor\_(<log\_op>,<log\_op>) | xor\_(<log\_op>,<log\_op>) | not\_(<boolean\_feature>)

<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<o> ::= +|-|\*

<conditional\_branches> ::= less\_than\_or\_equal(<num\_op>,<num\_op>) | greater\_than\_or\_equal(<num\_op>, <num\_op>)

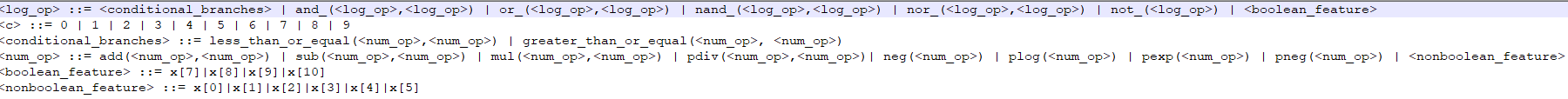
<num\_op> ::= add(<num\_op>,<num\_op>) | sub(<num\_op>,<num\_op>) | mul(<num\_op>,<num\_op>) | pdiv(<num\_op>,<num\_op>) | <nonboolean\_feature> | <nonboolean\_feature> <o> <nonboolean\_feature> | <nonboolean\_feature> | <nonboolean\_feature> <o> <nonboolean\_feature>

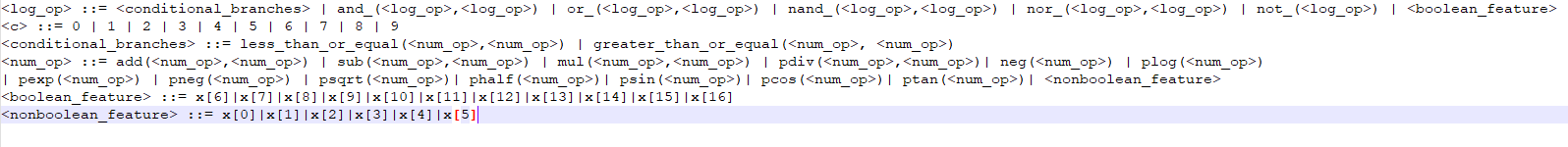
<boolean\_feature> ::= x[6]|x[7]|x[8]|x[9]|x[10]|x[11]|x[12]|x[13]|x[14]|x[15]|x[16]|x[17]|x[18]|x[19]|x[20]

<nonboolean\_feature> ::= x[0]|x[1]|x[2]|x[3]|x[4]|x[5]|x[6]|<c><c>.<c><c>|mean\_(<nonboolean\_feature>)

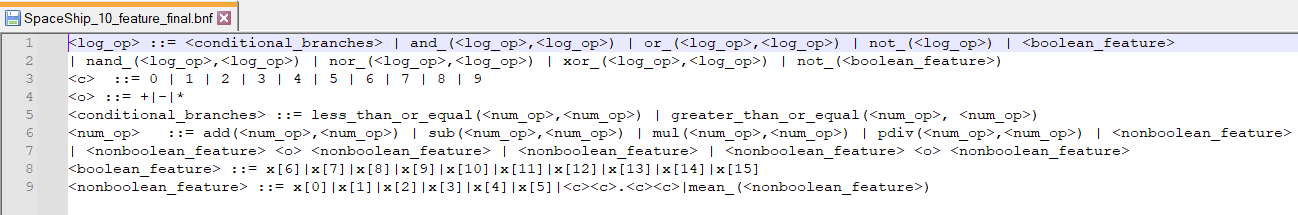
The grammar was updated in terms of new choices that can be used from the function set that grape has inbuilt and some custom function that we built using numpy library and array manipulation. We were first using trigonometric functions in our grammar but we then removed them because we realized that it would not be very useful as they are better fitted for regression problems and not classification. The problem was the number of functions we had, this was increasing the number of choices per codon and the limited number of features was not giving us good results. The fitness was dropping till 0.24 at max.

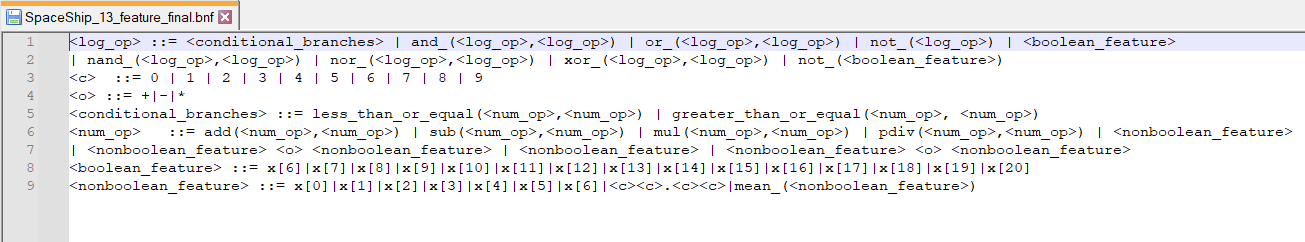
Early versions of the grammar





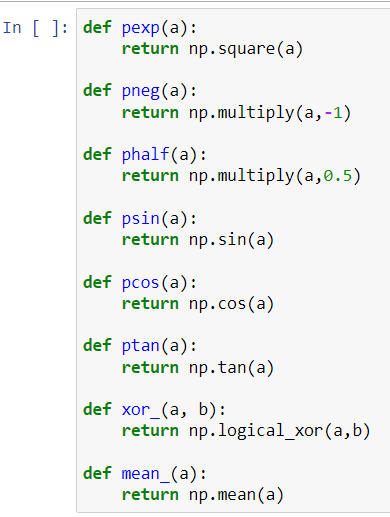
After removing all unnecessary function we were left with the following grammar which gave better results arround min(fitness) 0.21 to 0.225





**Function Set**

We used functions from grape directory and used a some of the functions we created, they are given below



**Parameters**

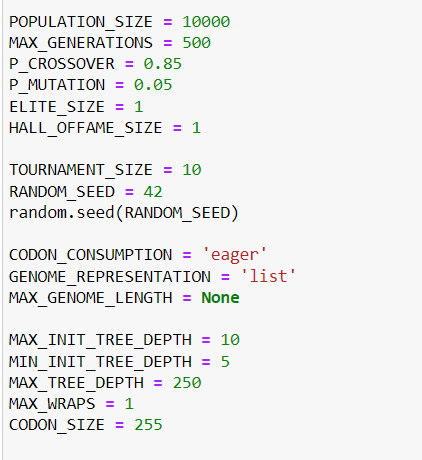
The concept behind the parameters was to run it under two situations simultaneously: one with a bigger population and a lower generation, and the other with the opposite. After asking Allan for assistance, we decided to keep the population size at 10,000 because he advised us that a very big population could lead to overtraining of the few features we have. As a result, we changed the population size from 100,000 to 10,000. Additionally, he noted that since we were taking 100 players, the tournament size needed to be decreased. This was changed to 10.

Due to the same reasoning, the number of generations was also decreased from 10,000 to 500.

Due to the problem's high degree of intricacy, a high crossover and mutation rate was chosen to ensure that genetic material is exchanged.

We kept max wrap set to 1 to allow wrapping in order to somewhat reduce the amount of invalid people, which was between 70 and 80k from a 100k population. This reduced it to roughly 60k.

The tree's depth was kept between 5 and 10, while the maximum depth ranged from 7 to 15.

****

### 

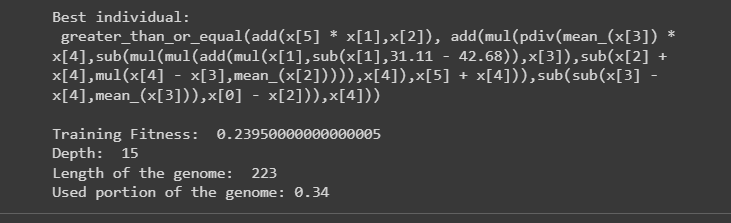
### 

### 

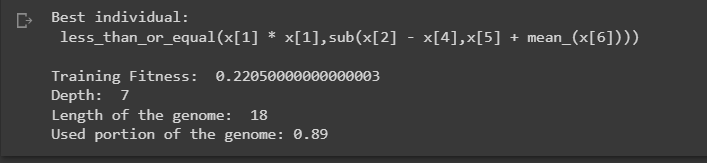
### 

### **Results**

The outcomes we obtained were expressed in terms of min fitness, which was a minimization of the error between training data and predictions. The fitness score range we typically received ranged between 0.21 and 0.225, which resulted in test accuracy of 77 to 79%. Ironically, the test accuracy for the training accuracy of 0.2205 was 79, but the test accuracy for the lowest fitness we obtained, 0.2104, was 77. This might be because the training dataset was overfit.

For 10 features we got 

For 13 features we got



**Observations**

Increasing the number of individuals didn't really change anything; while it might result in a lower fitness score, it also significantly lengthened the run. Each population produced a lot of invalid, thus as the population grew, so did the number of invalid people. This made it simpler to keep the population number as small as possible and determine which other factors were more important. Although increasing the maximum generations decreased the fitness score, it also prolonged the algorithm's runtime.

**Conclusion**

In conclusion, the two notebooks we used took a lot of effort and constant brain storming.

The 13 feature strategy did prove itself useful in the end but keeping the 10 features also gave us important insights about how the grammar is affected. One important thing we noticed is that grammatical evolution, although quite slow, is really useful.

The porting of code from Colab to Jupyter turned out to be a turning point as we were able to train for longer hours without the risk of disconnecting and using out processors to help the algorithm train faster

We submitted 50+ predictions which we think is the highest out of all is enough to show how much effort we put into this project.

**What could’ve been handled differently?**

We could've worked more on the grammar and its simplification because in the end it became too complex.

Maybe if we stuck to the original 10 features and tried to make a simpler and better grammar, we could've ended up with a better score, but by using machine learning techniques in GE helped us to understand how powerful GE can be instead of relying on traditional classifiers.