Evolutionary Algorithms Project

Sajin Mohamed Pallikkathodi Erathali
23037601@studentmail.ul.ie

Problem Statement

The given problem is to design a classifier for determining whether the Adult Earns an income more than 50K based on the input features. The input features are listed below :

age: continuous.
workclass: categorical (Private, Self-emp-not-inc, Local-gov, State-gov).
education: categorical (Bachelors, Some-college, HS-grad, Masters, Doctorate).
marital-status: categorical (Married-civ-spouse, Divorced, Never-married).
relationship: categorical (Wife, Husband, Not-in-family, Other-relative).
race: categorical (White, Asian-Pac-Islander, Black).
sex: categorical (Female, Male).
capital-gain: continuous.
capital-loss: continuous.
hours-per-week: continuous.
native-country: categorical (United-States, Others).

The training(adult_training.csv) and test(adult_test.csv) data were provided in the sample notebook.

In the input provided we had 11 features, however, to fit this data into a more representable format some preprocessing is required.

I decided to go with GE as that would be more fun and challenging over GP and I would be able to tune the grammar, whereby results can be altered not just by the function set and the fitness functions.

Data Preprocessing

Since we are trying to design a binary classifier(ie. Whether the adult earns more than 50K or Not), I had to replace the 'income' column which denoted <=50K and >50K with 0 and 1 respectively.

Additionally, since 4 of the features(age, capital-gain, capital-loss, hours-per-week) were continuous data and the rest we either binary or categorical data, I decided to normalize the continuous data by applying a typical standardization which involved the scaling of these continuous features so that they have a mean of 0 and standard deviation of 1. This process is also called z-score normalization, where we subtract the mean of the column from the actual value and divide it by the standard deviation of the column. This would help at a later stage when using the classifier.

$$v' = \frac{v - \overline{A}}{\sigma_A}$$

Then I looked at the categorical data such as 'workclass', 'education', 'marital-status', 'relationship', 'race'. These data could be modified with one-hot encoding which is done using pandas library(pandas.get_dummies()). This resulted in more granular features ie. we created new columns but each with values indicating whether it belongs to this category or not, for instance the column marital status was split into or replaced by columns Married-civ-spouse, Divorced and Never-married. These data can be now considered as Boolean as well. This helps remove the dependency of the value of the field from the classifier. The classifier now sees this as true or false rather than 1,2,3,4.

Next I looked into the features that can have only two distinct values(Boolean features) which are sex and native-country. I decided to view them as Booleans since they can take only one or the other values, for sex being male or female and native-country being United-States or Others.

Applying the above-mentioned steps on test data as well since we are fitting train data to model, we would need the test data to behave the same way.

Operators used

Initially, I only used the basic operations such as add, sub, mul, protected div, and, or and not.
Then I added a few others as well like psqrt, less_than, greater_than, neg, nand_ but adding these did not alter the fitness or the performance much.

Selection Criteria : The selection methodology used is Tournament Selection as it is more robust towards presence of noisy data since it focuses on a subset of individuals and not the whole population and since the best individuals get promoted and is unlike roulette selection where any individual gets promoted based on their probability. Also we have the advantage of changing the tournament size to alter the result which is one of the test cases I designed for the experiments I conducted.

Experiments

Grammar File

The grammar file is adapted from the heartDisease.bnf and is modified as shown below :

```
<log_op> ::= <conditional_branches> | and_(<log_op>,<log_op>) |
or_(<log_op>,<log_op>) | not_(<log_op>) | <boolean_feature>

<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>

<boolean_feature> ::=
x[1]|x[5]|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]|x[21]|x[22]|x[23]|x[24]
```

<nonboolean_feature> ::= x[0]|x[2]|x[3]|x[4]|<c><c>.<c><c>
<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Here the logical operators are AND, OR and NOT. The conditional branches check for less than or equal and greater than or equal. The number operators are add, sub, mul, pdiv. Then we have the Boolean operators which we identify as the columns in the modified dataset after the above-mentioned preprocessing steps.

Initial experiments were conducted with a smaller Generation size of 100 and population size of 1000. The crossover rate was kept at 0.7 and the mutation rate at 0.01 and the elite size was kept at 1 and tournament size at 7. Max tree depth was kept as 17 as increasing this leads to increased time in execution of the program, which makes sense as it would need to increase the number of computations. These parameters with the above-mentioned grammar yielded me the best fitness score of 0.2234615384615385. On Kaggle this result reported a score of 0.7556. Here the used portion of the genome was 0.29 for the best individual.

Successively, to get better fitness I made a few changes to the parameters like so
Crossover = 0.8
Mutation = 0.004
PopulationSize = 3000
MaxGeneration = 200
EliteSize = HallofFameSize = 1
Tournament Size = 14

And I ran the simulation and observed that the best fitness at generation 200 is 0.19288461538461543. This gave me a Kaggle score of 0.7821. Hence I assumed as the max fitness value decreased the Kaggle score would improve.

Hence I changed the mutation rate to 0.01 and changed the tournament size to 7 in order to run the program faster. I also increased the MaxGeneration to 300. However, the max fitness value at generation 300 increased to a value of 0.19769230769230772.

Interestingly, when submitted to Kaggle it gave me a better score of 0.78989. This leads me to believe that the reduction in fitness value alone does not increase the Kaggle score. This could be because its taking only 33% of the test data to calculate the score. It takes over 2 hours to run this simulation.

The best individual and the training fitness as a result of this run are as shown below :

Best individual: and_(or_(x[20],or_(or_(x[20],greater_than_or_equal(x[2],
sub(mul(x[2],add(x[3],m
ul(x[2],add(x[3],add(x[3],x[3])))))),x[2]))),or_(x[20],greater_than_or_equal(pdiv
(add(x[3],mul(x[2],add(x[0],add(x[3],x[3])))),x[2]), mul(x[2],sub(x[2],mul(x[4],
x[4])))))),and_(or_(x[11],or_(and_(x[10],x[19]),or_(or_(x[15],greater_than_or_e qual(x[3],
mul(mul(x[2],add(x[3],mul(x[2],add(x[3],add(x[3],x[3])))))),x[2]))),or
(x[10],x[13])))),and(or_(x[11],or_(or_(x[15],greater_than_or_equal(x[3], mul(m
ul(x[2],add(x[3],mul(x[2],add(x[3],add(x[3],x[3])))))),x[2]))),or_(x[10],x[13])))
,and_(or_(x[11],or_(x[16],less_than_or_equal(pdiv(x[4],x[4]),x[2]))),or_(not_(x[
20]),and_(not_(x[10]),not_(x[7])))))))

Training Fitness: 0.19769230769230772

Further test include modifying the grammar like so

<log_op> ::= <conditional_branches> | and_(<log_op>,<log_op>) | or_(<log_op>,<log_op>) | not_(<log_op>) | <boolean_feature>
<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>
<boolean_feature> ::= x[1]|x[5]|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]|x[21]|x[22]|x[23]|x[24]|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>
<nonboolean_feature> ::= x[0]|x[2]|x[3]|x[4]|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>
<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<b> ::= 0 | 1

Here I gave a probability of choosing Boolean values to the <Boolean_features> and created a terminal <b> where b can have the values 0 or 1. Here the target is to balance the grammar so that the grammar is not biased.

The experiment was run with tournament size 7 and the population size of 3000 and max generation as 300 and crossover of 0.8 and mutation rate of 0.01.

However, with this modification the Kaggle score did not improve but dropped slightly to .7830

Best Individual from Kaggle Score

Therefore, the best individual for a training fitness of 0. 19769230769230772 and Kaggle score of .7898 is

and_(or_(x[20],or_(or_(x[20],greater_than_or_equal(x[2],sub(mul(x[2],add(x[3],mul(x[2],add(x[3],add(x[3],x[3])))))),x[2]))),or_(x[20],greater_than_or_equal(pdiv(add(x[3],mul(x[2],add(x[0],add(x[3],x[3])))))),x[2]),mul(x[2],sub(x[2],mul(x[4],x[4])))))))),and_(or_(x[11],or_(and_(x[10],x[19]),or_(or_(x[15],greater_than_or_equal(x[3],mul(mul(x[2],add(x[3],mul(x[2],add(x[3],add(x[3],x[3])))))),x[2]))),or_(x[10],x[13])))),and_(or_(x[11],or_(or_(x[15],greater_than_or_equal(x[3],mul(m

ul(x[2],add(x[3],mul(x[2],add(x[3],add(x[3],x[3])))),x[2])),or_(x[10],x[13])))
,and_(or_(x[11],or_(x[16],less_than_or_equal(pdiv(x[4],x[4]),x[2]))),or_(not_(
x[20]),and_(not_(x[10]),not_(x[7])))))))

Depth: 17
Length of the genome: 3915
Used portion of the genome: 0.04

Observations

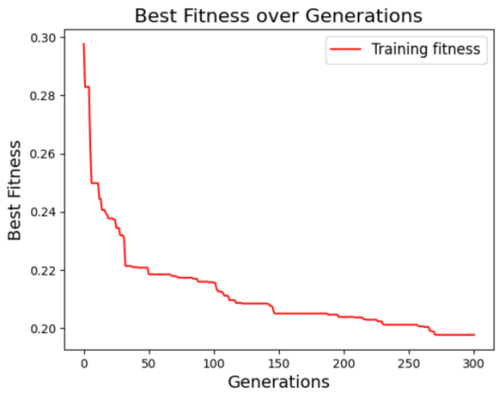

Fig. 1

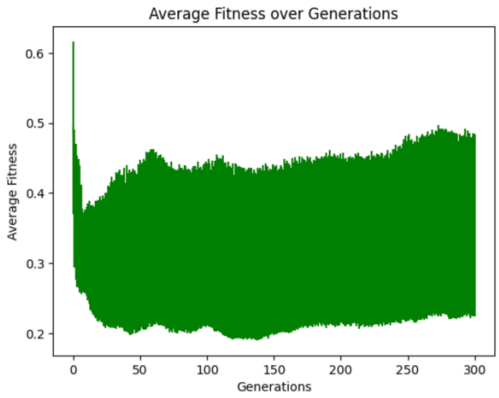

Fig. 2

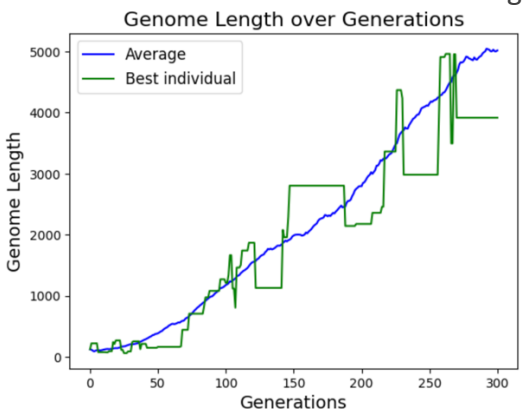

Fig. 3

The figure 1 shows the drop in the fitness value over generations and can be observed that the best fitness was observed at 270th generation. The figure 2 shows the Average fitness over the generations and figure 3 illustrates the length of Genomes increasing over generations in order to obtain a better result, which is why we can see the best individual is not small.

Even though I tried numerous approaches like adding new functions in the functions.py in grape and altering the grammar the best fitness I was able to get for the best result in Kaggle was for a fitness of 0.19769230769230772

Other Experiments Conducted

In an attempt to decrease the fitness value, I tried to add a few other terminals on the grammar by adding other functions like neg, psqrt, max, min, plog as marked in red below, but it only worsened the fitness value compared to the unaltered one.

In one of the trails, I even tried to define the grammar like

<log_op> ::= <conditional_branches> | and_(<log_op>,<log_op>) | or_(<log_op>,<log_op>) | not_(<log_op>) | nor_(<log_op>,<log_op>) | nand_(<log_op>,<log_op>) | <boolean_feature>
<conditional_branches> ::= less_than_or_equal(<num_op>,<num_op>) | greater_than_or_equal(<num_op>, <num_op>) | greater_than(<num_op>, <num_op>) | less_than(<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>) | neg(<num_op>) | psqrt(<num_op>) | plog(<num_op>) | max_(<num_op>,<num_op>) | min_(<num_op>,<num_op>) | <nonboolean_feature>

<boolean_feature> ::= x[1]|x[5]|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]|x[21]|x[22]|x[23]|x[24]|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>|<b>

<nonboolean_feature> ::= x[0]|x[2]|x[3]|x[4]|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>|<c><c>.<c><c>

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

<b> ::= 0 | 1

Here I was trying to give a probability of normalization by providing alternatives for boolean and nonboolean features and added the other function sets to this. Even with this the result did not improve much. It was comparable with the result produced by using the initial grammar illustrated above.

Conclusion

- As with any problem finding a good minima is difficult. It may be established that the best performant version can be obtained by using mathematical formulas in the fitness function, however, here I have only used the same fitness function that was provided in the sample notebook that tries to reduce the percentage of wrongly predicted outputs.

- The larger the population size the better the initial population fitness. If I reduce the population size the initial population fitness value increases which is not what we need.

- Adding additional functions from grape did not carry much weight to the problem, hence its not really efficient solution, although it helps slightly.

- Changing Random seeds did yield better fitness

- Finally, the best runs took hours of computation time, which when done via Google Colab timed out and had to write a script for the session to be active. Better results could be observed by changing the parameters like population size, max generation, mutation rate and crossover.

Future Investigations

Other avenues that is worth investigating would be to enhance the elite size thereby investigating elitism and change the grammatical representation to various styles. Another approach would be to try this problem using strongly typed GP.

Link to Video Report

https://youtu.be/F7xCvNIUJcY