Master of Science in Artificial Intelligence and Machine Learning: -

Evolutionary Computation Final Project CS6271(2023/4) :-

Team:-

Vilohit Keshava Murthy Achar

(Student id:- 23077751)

Ayush Ratnaparkhi

(Student Id:- 23119012)

Prof. Conor Ryan

Dept. of Computer Science and Information Systems

Problem statement:

Predict whether income exceeds $50K/yr. based on census data. This is a shorter version of the also known as the "Census Income" dataset (donated on 4/30/1996).

Dataset:

In this project, we're working with the Census Income dataset, a collection of individual parameters and corresponding income levels categorized as either above or below $50,000 annually. The dataset comprises continuous attributes like age, capital gain, capital loss, and hours per week, alongside categorical attributes such as workclass, education, marital status, relationship, race, sex, and native country.

Preprocessing & Methodology:

To process the data, we're utilizing the Pandas library to read and convert the files into data frames. Instead of Genetic Programming, we've opted for Grammatical Evolution in our implementation. The Genetic Programming approach typically requires GRAPE and a grammar file, both of which are included in the later part of the code after the data preprocessing step.

During preprocessing, we employ one-hot encoding on the categorical values to transform them into numerical representations. Following the prediction phase, the output is saved to a CSV file, which will be subsequently uploaded to Kaggle.

In the realm of evolutionary computation, the proposed solution is realized through the utilization of GRAPE. This implementation involves leveraging libraries such as deap, algorithms, csv, and grape. These tools collectively contribute to the effective execution of the evolutionary computation approach, providing the necessary frameworks for the solution's development and integration.

Implementation:

Grammatical evolution begins with a starting population made up of randomly generated programs encoded according to a preset grammar. The population is iteratively refined over several generations by using genetic operators like crossover and mutation one after the other. Finding competent programs capable of solving the given problem statement is the aim.

Upon importing the dataset and performing one-hot encoding on categorical columns for both the training and test sets, the subsequent task is to convert the income column in the training data. The existing format of the income column involves string representations such as <=50k and >50k. To facilitate further analysis, it is essential to transform these string values into numerical counterparts, specifically 0 for <=50k and 1 for >50k.

```
[ ]  1 X_train = df_train.copy()
     2 # warning: cannot drop it more than once
     3 X_train.drop(['income'], axis=1, inplace=True)
```

You should represent the outputs with 0 where the income is smaller or equal to 50K and with 1 if it is greater than 50K.

Follow exactly this approach, because the test targets are represented like this in the competition.

```
     1 # class labels
     2 l, _ = X_train.shape
     3
     4 y_train = np.zeros([l,], dtype=int)
     5
     6 for i in range(l):
     7   if df_train['income'].iloc[i] == '>50K':
     8     y_train[i] = 1
     9   elif df_train['income'].iloc[i] == '<=50K':
    10     y_train[i] = 0
```

```
[ ]  1 print(y_train[0:5]) #print head
```

```
[0 1 0 0 1]
```

To effectively employ the fitness function with GRAPE, it is imperative that the features are arranged in rows, while the samples are organized in columns.

If your data does not conform to this structure, it becomes necessary to transpose the matrix accordingly.

This ensures proper alignment for the fitness function to appropriately evaluate the genetic programs within the GRAPE framework.

```
[ ] print('Before Translose Training (X,Y):\t', X_train.shape, y_train.shape)
    print('Before Translose Test (X):\t', X_test.shape)

    X_train = np.transpose(X_train)
    X_test = np.transpose(X_test)

    print('Training (X,Y):\t', X_train.shape, y_train.shape)
    print('Test (X):\t', X_test.shape)
```

```
Before Translose Training (X,Y):        (5200, 25) (5200,)
Before Translose Test (X):      (10402, 25)
Training (X,Y):  (25, 5200) (5200,)
Test (X):       (25, 10402)
```

Setting GE Parameters:

```
[ ]  POPULATION_SIZE = 1000
     MAX_GENERATIONS = 200
     P_CROSSOVER = 0.7
     P_MUTATION = 0.01
     ELITE_SIZE = 2
     HALL_OF_FAME_SIZE =  10

     TOURNAMENT_SIZE = 15
     RANDOM_SEED = 42
     random.seed(RANDOM_SEED)

     CODON_CONSUMPTION = 'lazy'
     GENOME_REPRESENTATION = 'list'
     MAX_GENOME_LENGTH = None

     MAX_INIT_TREE_DEPTH = 15
     MIN_INIT_TREE_DEPTH = 5
     MAX_TREE_DEPTH = 90
     MAX_WRAPS = 0
     CODON_SIZE = 255

     REPORT_ITEMS = ['gen', 'invalid', 'avg', 'std', 'min', 'max',
                     'best_ind_length', 'avg_length',
                     'best_ind_nodes', 'avg_nodes',
                     'best_ind_depth', 'avg_depth',
                     'avg_used_codons', 'best_ind_used_codons',
                     'structural_diversity', 'fitness_diversity',
                     'selection_time', 'generation_time']
```

The subsequent phase involves the preparation of the toolbox, as already outlined in the notebook. The toolbox encompasses specific details about the selection method, notably Tournament Selection.

The advantage of employing this method lies in affording each individual an opportunity to showcase its fitness score before the selection process unfolds.

Following the generation of the initial population for generation 0, the workflow transitions to the execution of `ge_eaSimpleWithElitism()` from the algorithms library.

This main execution process is pivotal, as it systematically computes the optimal fitness value while safeguarding against loss through continuous population improvement in each successive generation.

```
gen = 0 , Best fitness = (0.3011538461538461,)
gen = 1 , Best fitness = (0.2973076923076923,) , Number of invalids = 164
gen = 2 , Best fitness = (0.28711538461538466,) , Number of invalids = 175
gen = 3 , Best fitness = (0.2828846153846154,) , Number of invalids = 104
gen = 4 , Best fitness = (0.2828846153846154,) , Number of invalids = 122
gen = 5 , Best fitness = (0.2828846153846154,) , Number of invalids = 156
gen = 6 , Best fitness = (0.2792307692307693,) , Number of invalids = 148
gen = 7 , Best fitness = (0.2759615384615385,) , Number of invalids = 43
gen = 8 , Best fitness = (0.27365384615384614,) , Number of invalids = 37
gen = 9 , Best fitness = (0.2619230769230769,) , Number of invalids = 26
gen = 10 , Best fitness = (0.2619230769230769,) , Number of invalids = 4
gen = 11 , Best fitness = (0.2619230769230769,) , Number of invalids = 0
gen = 12 , Best fitness = (0.2619230769230769,) , Number of invalids = 0
gen = 13 , Best fitness = (0.2619230769230769,) , Number of invalids = 0
gen = 14 , Best fitness = (0.2503846153846154,) , Number of invalids = 0
gen = 15 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 16 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 17 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 18 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 19 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 20 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 21 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 22 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 23 , Best fitness = (0.23884615384615382,) , Number of invalids = 0
gen = 24 , Best fitness = (0.2334615384615385,) , Number of invalids = 0
gen = 25 , Best fitness = (0.2334615384615385,) , Number of invalids = 0
gen = 26 , Best fitness = (0.2334615384615385,) , Number of invalids = 0
gen = 27 , Best fitness = (0.23288461538461536,) , Number of invalids = 0
gen = 28 , Best fitness = (0.23173076923076918,) , Number of invalids = 0
gen = 29 , Best fitness = (0.23173076923076918,) , Number of invalids = 0
gen = 30 , Best fitness = (0.22615384615384615,) , Number of invalids = 0
```

```
gen = 175 , Best fitness = (0.20288461538461533,) , Number of invalids = 0
gen = 176 , Best fitness = (0.20288461538461533,) , Number of invalids = 0
gen = 177 , Best fitness = (0.20288461538461533,) , Number of invalids = 0
gen = 178 , Best fitness = (0.20288461538461533,) , Number of invalids = 0
gen = 179 , Best fitness = (0.20288461538461533,) , Number of invalids = 0
gen = 180 , Best fitness = (0.20134615384615384,) , Number of invalids = 0
gen = 181 , Best fitness = (0.2007692307692308,) , Number of invalids = 0
gen = 182 , Best fitness = (0.2007692307692308,) , Number of invalids = 0
gen = 183 , Best fitness = (0.20038461538461538,) , Number of invalids = 0
gen = 184 , Best fitness = (0.20038461538461538,) , Number of invalids = 0
gen = 185 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 186 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 187 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 188 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 189 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 190 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 191 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 192 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 193 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 194 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 195 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 196 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 197 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 198 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 199 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
gen = 200 , Best fitness = (0.20019230769230767,) , Number of invalids = 0
```

If Best fitness value is low, then accuracy on Kaggle will be improved. Because that represent loss of accuracy if provided with dataset.

Details of Best Individual:

```
# Best individual
import textwrap
best = hof.items[0].phenotype
print("Best individual: \n","\n".join(textwrap.wrap(best,80)))
print("\nTraining Fitness: ", hof.items[0].fitness.values[0])
print("Depth: ", hof.items[0].depth)
print("Length of the genome: ", len(hof.items[0].genome))
print(f'Used portion of the genome: {hof.items[0].used_codons/len(hof.items[0].genc
```

```
Best individual:
 or_(and_(or_(x[13],or_(less_than_or_equal(91.86,pdiv(x[2],54.75)),or_(less_than_
or_equal(74.99,pdiv(x[0],x[0])),and_(greater_than_or_equal(x[3], x[2]),or_(less_
than_or_equal(74.59,pdiv(x[3],64.74)),or_(x[11],or_(less_than_or_equal(74.55,pdi
v(x[3],24.75)),not_(not_(x[10])))))))),x[16]),greater_than_or_equal(pdiv(sub(65
.08,x[3]),add(add(sub(44.01,98.69),pdiv(x[2],79.55)),pdiv(19.54,pdiv(79.43,sub(6
7.80,pdiv(x[3],add(add(sub(40.34,98.92),pdiv(x[2],79.53)),pdiv(19.51,pdiv(79.55,
sub(67.53,pdiv(60.41,pdiv(x[3],76.01))))))))))), sub(98.55,pdiv(x[2],79.97))))

Training Fitness:  0.20019230769230767
Depth:  19
Length of the genome:  808
Used portion of the genome: 0.28
```

In further execution we are working on generating csv but before that we need to predict output using X_test data which we have read from testing file.

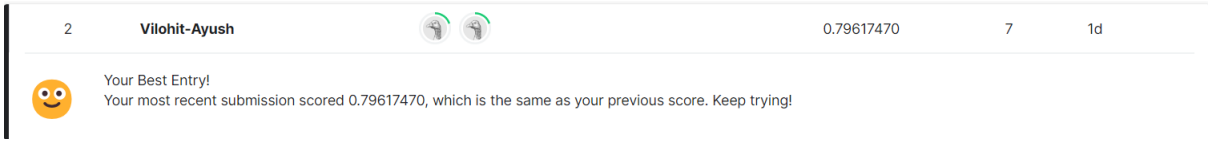
Using predict () method will predict y_pred but we don't know the expected output.

Now y_pred values are in terms of True and False which we need to convert to give to model to evaluate the result which we have achieved using following method:

```
y_pred.astype(int)
```

Result:

After predicting output, we are writing output to csv which will then be uploaded to Kaggle competition and as you can see below we are able to achieve following accuracy securing 2nd rank in competition:



We have carried different iterations which are following along with result:

POPULATION_SIZE	MAX_GENERATIONS	P_CROSSOVER	P_MUTATION	ELITE_SIZE	HALLOF FAME_SIZE	TOURNAMENT_SIZE	RANDOM_SEED	MAX_INIT_TREE_DEPTH	MIN_INIT_TREE_DEPTH	MAX_TREE_DEPTH	MAX_WRAPS	CODON_SIZE	Fitness	Depth	Length of Gnome	Used portion of genome	
100	50	0.7	0.01	2	9	5	42	13	3	90	0	255	0.2498	6	2322	0.01	
1000	100	0.6	0.01	2	9	15	42	15	5	90	0	255	0.2171	17	2136	0.07	
1000	210	0.7	0.01	2	10	15	42	14	4	90	0	255	0.21634	43	725	0.46	
1000	200	0.8	0.01	2	10	15	42	15	5	90	0	255	0.21807	36	8341	0.06	
1000	200	0.7	0.01	2	10	15	42	15	5	90	0	255	0.20019	19	808	0.28	BEST
1000	260	0.7	0.02	2	10	15	42	15	5	90	0	255	0.25288	22	456	0.8	
1000	220	0.7	0.01	2	10	15	42	15	5	90	0	255	0.2001	20	806	0.28	