Genetic Programming Rice Categorization Problem

Thanushan Pirapakaran
Adrian Binu
COSC 4P82
Prof: Brian Ross
Brock University
February 10th, 2024

The Rice Problem

Given a set of scrambled datasets of rice, train a genetic program to find a function that can accurately categorize the rice. In particular, the dataset is of seven descriptive variables: area, perimeter, major axis length, minor axis length, eccentricity, convex area, extent, and class. Thus, the program should utilize these first seven variables to develop a function that effectively classifies each rice sample as either Cammeo or Osmancik.

Furthermore, in every GP algorithm fashion, we must find the most optimal settings that provide the best performance. This is in terms of population, generations, crossover rate, mutation rate, and training vs. testing split.

Experiment

The parameters are chosen in mind to run on Google Colab, so naturally we decided to limit our pop size to 800 and generation to 50 as we thought otherwise it would take too long with little progress in fitness because of how quickly the trees grow in complexity. We did not change the limits on trees either from part a) and also set elitism to 1 but saw very little difference without it. Everything is listed also in Table 1 The fitness for an individual is based on how many hits that individual can get, but to standardize the fitness we divided it by the total number of cases and had it subtracted by 1, the subtraction tells us that lower fitness is better and this allows gp to evolve the better-fit individuals. This is good because we want lower fitness values as Deap's example GP system uses a minimization technique, where individuals with lower fitness values are more favorable. We also squared the whole thing to be more biased towards more fit individuals. But before we could return this value of fitness we had to first get the predicted value from gp after it calculated it with the 7 float variables listed in the float array and then just like shown in Figure 9 we used the predicted value and categorized it, if it was greater than 0 then gp predicted Cammeo otherwise it was Osamancik. Then we compared the gp answer and the actual answer and if it was right then gave it a plus one hit.

```
If (GP_ans >= 0.0  and ans[i] = "Cammeo")
   then hits = hits + 1;
Else if (GP_ans  < 0.0 and ans[i] = "Osmancik")
   then hits = hits + 1;
```

Figure 9: Fitness Function Pseudocode

The language for the rice problem is listed in Table 2, the only difference between this and part a)'s language table is that we decided to get rid of sin and cos because we realized that without these two, we got faster and better results. So really it was just slowing us down for nothing.

Parameters	Settings 1	Settings 2
Pop. Size	800	800
Crossover Rate	70%	70%
Mutation Rate	30%	30%
Generations	50	50
Number of Elites	1	1
Min Tree Size (init)	0	0
Max Tree Size (init)	17	17
Min Tree Size (Mut.)	0	0
Max Tree Size (Max)	17	17
Tournament Size	3	3
Training Split	30%	50%
Testing Split	70%	50%

Table I: Parameters of two separate experiments

The dataset is of about 3800 rice samples, with eight separate rice data points. The first seven data points are numeric descriptive variables, while the last one is the class of the rice (Cammeo or Osmancik). In other words, the first seven data points are the GP's input variables, and the last data point is the GP's target variable. As the data linked in the bibliography is ordered in terms of rice classification, we must scramble the data to ensure that the training set consists of a variation of data samples. This ensures that the training set includes a diverse range of data samples to achieve better performance with less data. Thus, minimizing memorization and promoting generalization.

To identify the best training vs. testing split, we used a comparative experiment to compare a 30%/50% vs. 50%/50% training/testing split. As illustrated in Table I, all but the training and testing splits are identical.

The Results

Looking at the performance plots below there are many trends that can be seen. For one the size of the trees go up even if the fitness converges as shown in Figures 3 and 6 which are measured in nodes. The best fitness plots like in Figures 2 and 5 are much lower than the average, this is to be expected. There is very little difference in plots between the 30% and 50% splits but our results show that there is a slight difference and we determined that when we gave the program too much training data it would simply overtrain itself and memorize everything.

Also if you look at the confusion matrices in Figures 7 and 8 the 30% split has a slightly higher percentage in true positive and true negative values. You can even see the difference in the matrices, if you look at the 30% confusion matrix GP gets more Cammeo's right and Osmancik right but also gets them wrong fewer times too, but for both matrices, you can see that GP got more Osmancik right than Cammeo. This can be seen also in the overall accuracy rate as shown in Figures 10 and 11 where the 30% split had a higher accuracy rate out of all the 10 runs for each experiment.

Conclusion

After all comparisons, the best training and testing accuracy came from the

30% training and 70% testing split. This is due to the fact that the lowered training split minimized memorization and promoted generalization. In fact, our best-performing individual resulted in an 85% test accuracy and 82% training accuracy.

Furthermore, we assessed that when the fitness plots converged, the trees exponentially grew in size. Otherwise suggesting an increase in bloat during later generations.

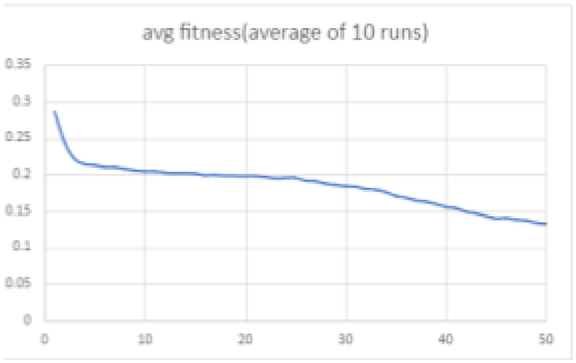


Figure 1: Avg fitness 30% training split

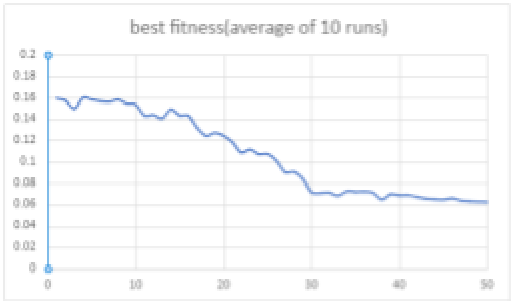


Figure 2: Best fitness 30% training split

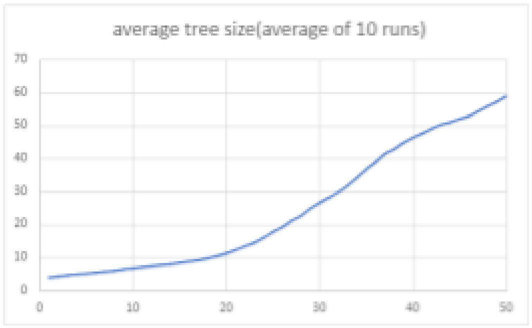


Figure 3: Avg tree size 30% training split

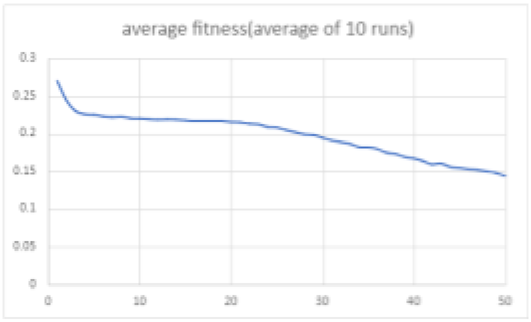


Figure 4: Avg fitness 50% training split

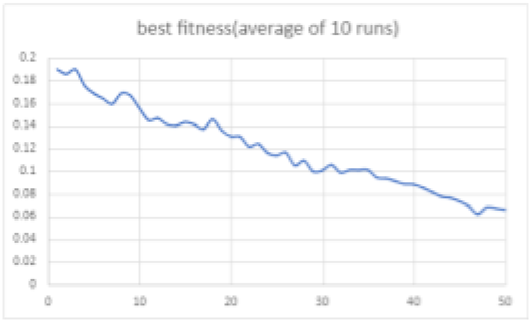


Figure 5: Best fitness 50% training split

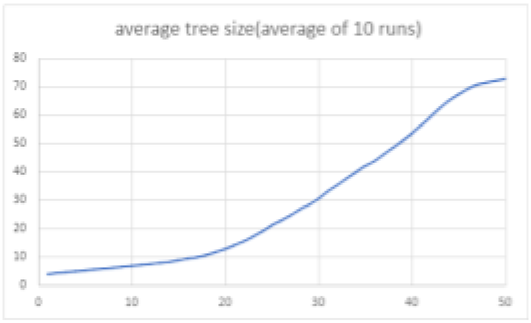
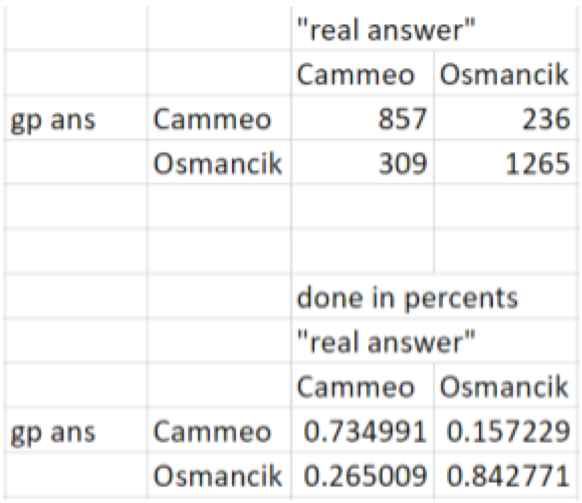
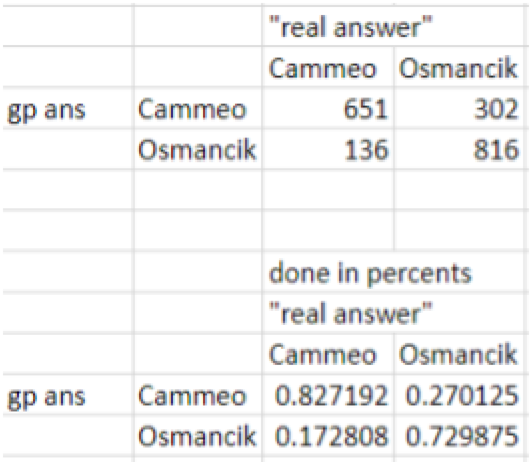


Figure 6: Avg tree size 50% training split



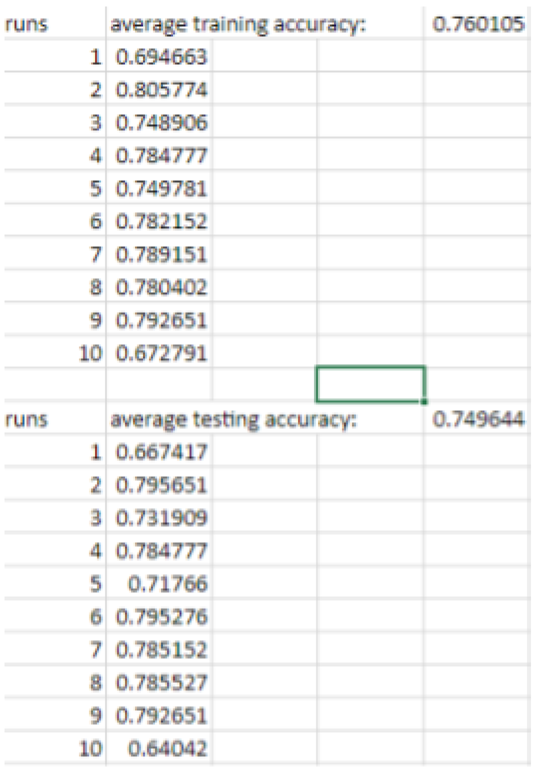
		"real answer"	
		Cammeo	Osmancik
gp ans	Cammeo	857	236
	Osmancik	309	1265
		done in percents	
		"real answer"	
		Cammeo	Osmancik
gp ans	Cammeo	0.734991	0.157229
	Osmancik	0.265009	0.842771

Figure 7: 30% training split confusion matrix



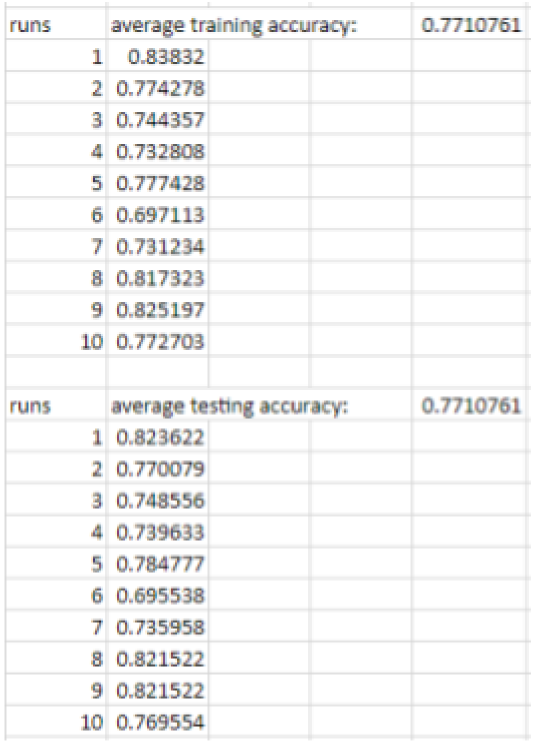
		"real answer"	
		Cammeo	Osmancik
gp ans	Cammeo	651	302
	Osmancik	136	816
		done in percents	
		"real answer"	
		Cammeo	Osmancik
gp ans	Cammeo	0.827192	0.270125
	Osmancik	0.172808	0.729875

Figure 8: 50% training split confusion matrix



runs	average training accuracy:		0.760105
1	0.694663		
2	0.805774		
3	0.748906		
4	0.784777		
5	0.749781		
6	0.782152		
7	0.789151		
8	0.780402		
9	0.792651		
10	0.672791		
runs	average testing accuracy:		0.749644
1	0.667417		
2	0.795651		
3	0.731909		
4	0.784777		
5	0.71766		
6	0.795276		
7	0.785152		
8	0.785527		
9	0.792651		
10	0.64042		

Figure 10: for 30% training split



runs	average training accuracy:		0.7710761
1	0.83832		
2	0.774278		
3	0.744357		
4	0.732808		
5	0.777428		
6	0.697113		
7	0.731234		
8	0.817323		
9	0.825197		
10	0.772703		
runs	average testing accuracy:		0.7710761
1	0.823622		
2	0.770079		
3	0.748556		
4	0.739633		
5	0.784777		
6	0.695538		
7	0.735958		
8	0.821522		
9	0.821522		
10	0.769554		

Figure 11: for 50% training split

Acronyms

GP Genetic Program

Appendix

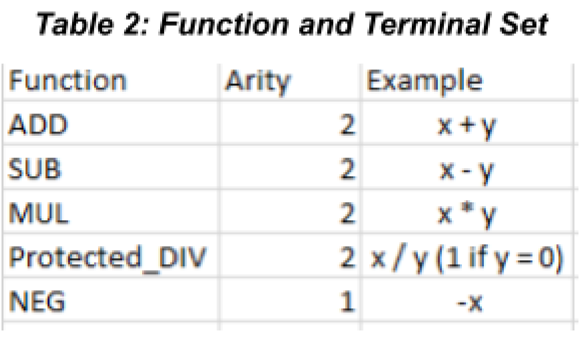
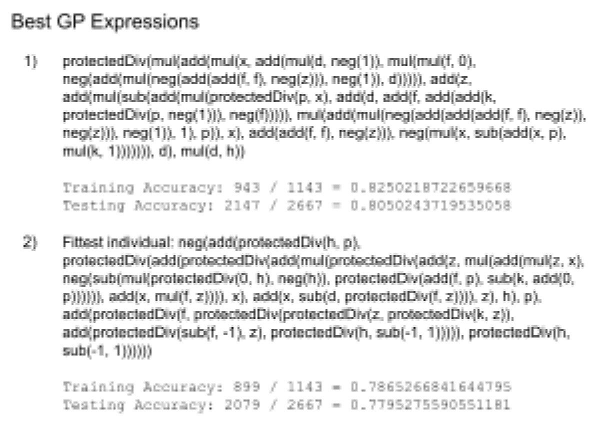


Table 2: Function and Terminal Set

Function	Arity	Example
ADD	2	x + y
SUB	2	x - y
MUL	2	x * y
Protected_DIV	2	x / y (1 if y = 0)
NEG	1	-x



Best GP Expressions

1) protectedDiv(mul(add(mul(x, add(mul(d, neg(1)), mul(mul(f, 0), neg(add(mul(neg(add(add(f, f), neg(z))), neg(1)), d))))), add(z, add(mul(sub(add(mul(protectedDiv(p, x), add(d, add(f, add(add(k, protectedDiv(p, neg(1))), neg(f))))), mul(add(mul(neg(add(add(add(f, f), neg(z)), neg(z))), neg(1)), 1), p)), x), add(add(f, f), neg(z))), neg(mul(x, sub(add(x, p), mul(k, 1))))))), d), mul(d, h))

```
Training Accuracy: 943 / 1143 = 0.8250218722659668
Testing Accuracy: 2147 / 2667 = 0.8050243719535058
```

2) Fittest individual: neg(add(protectedDiv(h, p), protectedDiv(add(protectedDiv(add(mul(protectedDiv(add(z, mul(add(mul(z, x), neg(sub(mul(protectedDiv(0, h), neg(h)), protectedDiv(add(f, p), sub(k, add(0, p)))))), add(x, mul(f, z)))), x), add(x, sub(d, protectedDiv(f, z)))), z), h), p), add(protectedDiv(f, protectedDiv(protectedDiv(z, protectedDiv(k, z)), add(protectedDiv(sub(f, -1), z), protectedDiv(h, sub(-1, 1))))), protectedDiv(h, sub(-1, 1))))))

```
Training Accuracy: 899 / 1143 = 0.7865266841644795
Testing Accuracy: 2079 / 2667 = 0.7795275590551181
```

Bibliography

[1] Rice (Cammeo and Osmancik). (2019). UCI Machine Learning Repository. https://doi.org/10.24432/C5MW4Z.
[2] Poli, Riccardo, et al. *A Field Guide to Genetic Programming*. Lulu Press, 2008.

Note
-the average of 10 runs, for memory consumption graph

-make labels for individuals