CS492 Search-Based Software Engineering

- Symbolic Regression with Genetic Algorithm -

20163287 박지혁

1. Manual

It is needed to install the `Scala` because it is the main implementation language for this project. After that, please compile with the following command:

```
make all
```

And then, please execute the solver with following comand:

```
scala Training {csv} {mode}?
```

The `{csv}` denotes the csv file name having a training input set and the `{mode}` denotes the training mode. The solver provides following three training modes:

1. `linear_local`: a local search for the linear regression.
2. `linear_gen`: a genetic algorithm for the linear regression.
3. `tree_gen`: a genetic programming for the symbolic regression.

If you did not give the mode then, the default mode will be selected. It is the third one, `tree_gen`.

And the tool give another command for calculate the Mean Square Error(MSE) for a test set:

```
scala Test {expr} {csv}
```

The `{expr}` denotes the expression using Reverse Polish Notation(RPN) and the `{csv}` denotes the csv file name having a test set.

2. Main Algorithms

I tried to implement three approaches for this project. Most of codes are in the directory **sym_reg** except for the object having main entry function for each of training and test commands.

1) Linear Regression with Local Search (linear_local):

The first approach is local search for the linear regression, because I believe that it is one of the simple but effective approaches. I implemented the `LocalLinear` class for this approach. I used the data structure `LinearExpr` to express the constant and coefficients for each of variables with the following shape:

$$c_0 + c_1x_1 + c_2x_2 + \ldots + c_{57}x_{57}$$

In the first step, I started with the zero expression. And then, I tried to locally search for randomly selected c_i. So, it means that it uniformly selects the constant or some coefficient of a variable and does a hill climbing. I compared the MSE between expressions having c_i as c_i, c_i-r, or c_i+r where c_i denotes the current constant and r denotes the current range. And it repeats the hill climbing 10 times with the smaller range r/5 for the same c_i. This is a small step for the selected c_i, and it repeats many times.

2) Linear Regression with Genetic Algorithm (linear_gen):

Another algorithm is to apply a genetic algorithm for the linear regression. I think that the constant and coefficients look like genes. So, I applied the genetic algorithm for it and its implementation is in the `GeneticLinear` class. I used the same data structure `LinearExpr` for it.

For the initial population, I used the three constants 0, 100.0, -100.0 and other 97 random linear expressions. And I defined the `crossover` function for two linear expressions, and it randomly selects coefficients between them for each of variables. Moreover, the `mutate` function change the coefficients into random numbers bounded in [-100.0, 100.0] in the probability 0.1.

Based on such functions, the population will be changed until the best one will be not changed. For the next generation, it selects two expressions by using the roulette mechanism with the probability 0.5. Finally, I repeat this algorithm many times for better solutions.

3) Symbolic Regression with Genetic Programming (`tree_gen`):

Final approach is applying the genetic programming into the symbolic regression. I unused the some unary operations: `asin`, `acos`, `atan`, `sqrt`, and `log`, because the domains of such operations are not total set of real numbers.

Data Structure

I implemented the `Node` class for represent the tree structure of expressions. The `TreeExpr` class has a `Node` object and a variable cache to optimize the calculation of its MSE. Also, a Node object has a member value `size` initialized with the number of nodes in sub-tree having the root as it. This `size` value is used to provide uniform selection for nodes.

Initial Population

I used 100 randomly generated expressions as the initial population. They are created based on the grow initialization with the maximum depth 10. In the probability 0.1, I selected the terminal nodes: variables or constants. In other cases, I selected the unary and binary operators in the same probabilities. For the children of nodes, I recursively applied such manner into them.

Selection

I used the roulette mechanism with the probability 0.5. If no expression was selected, I repeated the same process from the first expression of the given population.

Generation Creation

I selected two different expressions in the parent population and generate the next generation. I kept alive 5 best solutions in parent populations and create new 95 children from two selected expressions. I used the `crossover` and `mutate` function to generate child expression.

Crossover

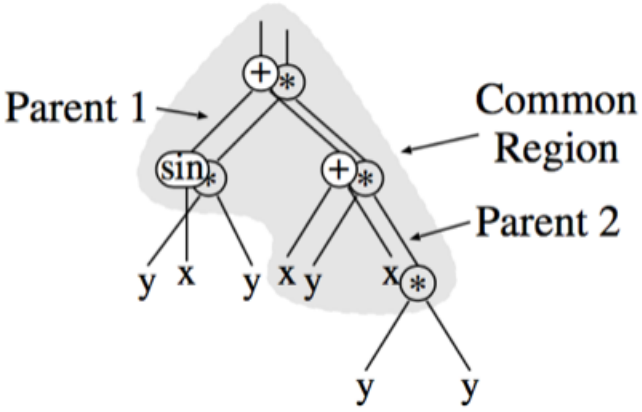

Figure 1 the crossover algorithm for two expressions.

I used the *uniform crossover* approach described in the **Figure 1**. For the common regions, I randomly selected the nodes from parents with same probabilities.

Mutation

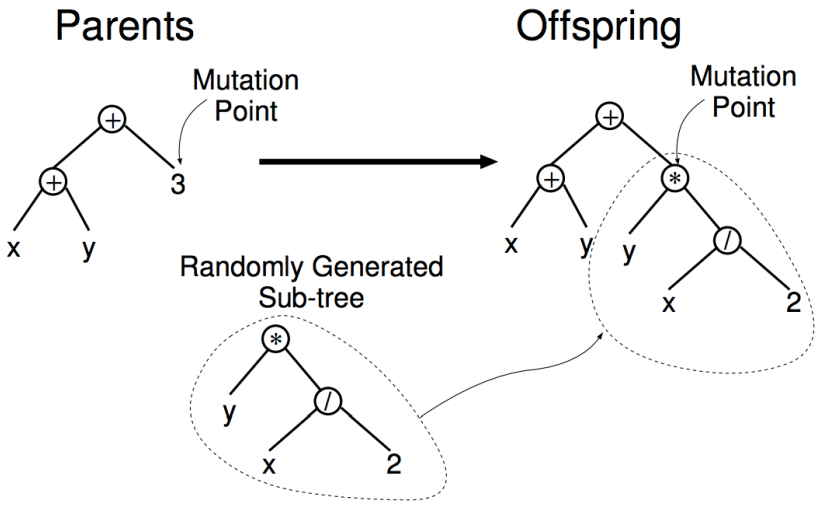

Figure 2 the sub-tree mutation algorithm for an expression.

I used two different methods for mutations: *sub-tree mutation* and *point mutation*. First, the sub-tree mutation is replacing a node into a randomly generated sub-tree. The **Figure 2** describes it. The node is randomly selected with uniform distributions. The point mutation is replacing the operations, constants or variable index of nodes into another values. I applied the point mutation into all the nodes in the given expression with the probability 0.1. I selected a method in the ratio 3:7 between the point and sub-tree mutation, respectively.

3. Evaluation

In this section, I will explain how to evaluate the experiments for three algorithms.

1) Linear Regression with Local Search (`linear_local`):

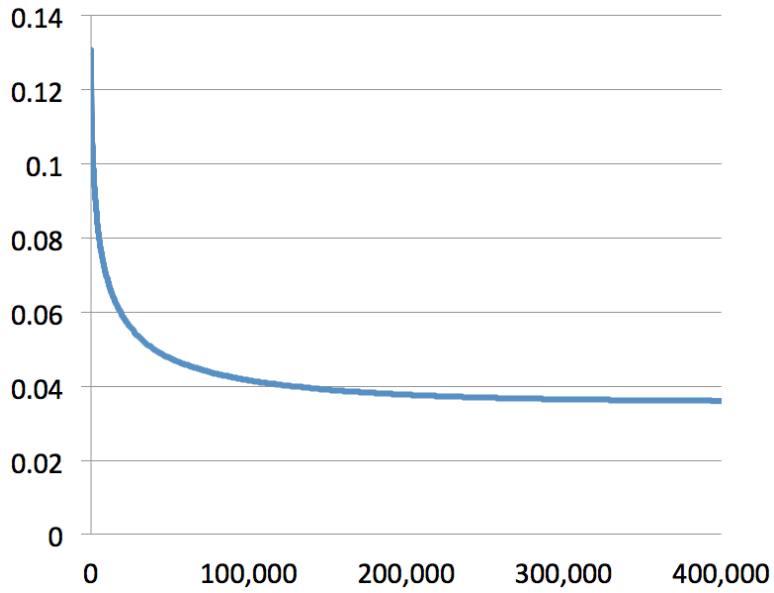

Figure 3 the line graph for MSEs of local search for linear expressions.

First, I repeated the small steps of the local search in 400,000 times. It took about 5 hours and the result is described in the **Figure 3**. At the first time, the MSE decreased quickly. After that, it is exponentially but smoothly slow down. At the end, the best one is the following expression with the MSE **`0.035763484842246486`**:

```
-0.6242713600000394 x1 -0.0025702400000000335 * x2 0.0 * x3 -
0.03136512000000148 * x4 0.04240895999999674 * x5 -
0.004710399999999991 * x6 -0.00236543999999982 * x7 -
0.0011519999999999942 * x8 0.003968000000000016 * x9
0.01801216000000066 * x10 -5.529599999999998E-4 * x11
0.005360639999999836 * x12 0.006840319999999802 * x13 -
3.0719999999999855E-5 * x14 -0.001909759999999987 * x15 -
0.002068479999999985 * x16 -0.021647360000002055 * x17 -
0.004049919999999982 * x18 -0.0047411199999999815 * x19 -
0.004838399999999974 * x20 -0.0036607999999999706 * x21 -
0.006952959999999961 * x22 -0.003194879999999961 * x23
0.013050879999999512 * x24 -0.005867519999999982 * x25
0.008965120000000142 * x26 -0.004812799999999996 * x27 -
```

```
0.0057599999999999865 * x28 -0.005734399999999989 * x29
0.004372479999999967 * x30 6.707199999999988E-4 * x31
0.005631999999999954 * x32 0.004208639999999965 * x33
8.243199999999972E-4 * x34 -6.45119999999999E-4 * x35 -
0.005708799999999989 * x36 -4.3008000000000033E-4 * x37 -
0.029230080000002295 * x38 -0.06825984000000432 * x39 -
3.174400000000004E-4 * x40 0.004459519999999966 * x41
1.8431999999999997E-4 * x42 0.003502079999999976 * x43
1.0239999999999983E-4 * x44 5.068800000000027E-4 * x45
0.0016435199999999916 * x46 0.0032358399999999755 * x47
0.005125119999999957 * x48 -2.0480000000000037E-4 * x49
0.004567039999999962 * x50 -0.0014643199999999955 * x51 -
0.0014284799999999915 * x52 0.0019046399999999876 * x53 -
0.0013055999999999925 * x54 0.07233023999999402 * x55
0.018099199999999944 * x56 0.015836160000000262 * x57
0.011422720000000604 * + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + +
```

2) Linear Regression with Genetic Algorithm (`linear_gen`):

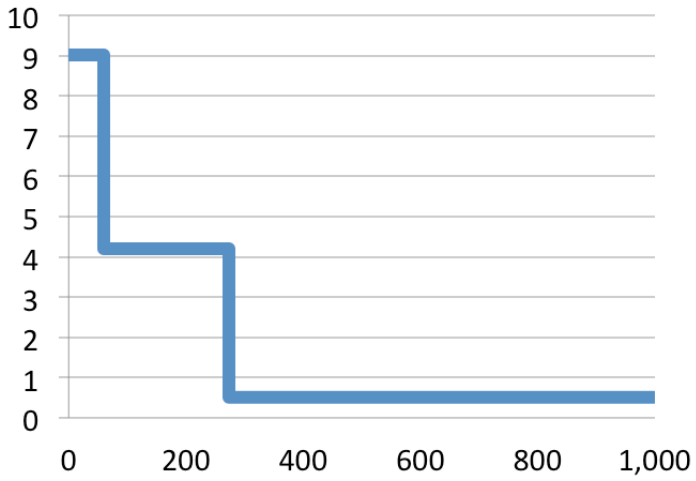

Figure 4 the line graph for MSEs of a genetic algorithm for linear expressions.

I iterated the sub-routine 1,000 times for the genetic algorithm to generate better linear expressions. The results are described in the Figure 4. It is actually converged after the 275 iteration. The final result is the following expression with the MSE **`0.5106285315509027`**:

```
-9.344606206150797 x1 0.0 * x2 0.0 * x3 0.0 * + + x4 0.0 * x5
0.0 * + x6 0.3319408000840851 * x7 0.0 * + + + x8 0.0 * x9 0.0
```

```
* x10 0.0 * + + x11 0.0 * x12 0.0 * + x13 0.0 * x14 0.0 * + +
+ + x15 0.0 * x16 0.0 * x17 0.0 * + + x18 0.0 * x19 0.0 * +
x20 0.0 * x21 0.0 * + + + x22 0.0 * x23 0.0 * x24 0.0 * + +
x25 0.0 * x26 0.0 * + x27 0.0 * x28 0.0 * + + + + + x29 0.0 *
x30 0.0 * x31 0.0 * + + x32 0.0 * x33 0.0 * + x34 0.0 * x35
0.0 * + + + x36 0.0 * x37 0.0 * x38 0.0 * + + x39 0.0 * x40
0.0 * + x41 0.0 * x42 0.0 * + + + + x43 0.0 * x44 0.0 * x45
0.0 * + + x46 0.0 * x47 0.0 * + x48 0.0 * x49 0.0 * + + + x50
0.0 * x51 0.0 * + x52 0.0 * x53 0.0 * + + x54 0.0 * x55 0.0 *
+ x56 0.0 * x57 0.0 * + + + + + + +
```

3) Symbolic Regression with Genetic Programming (tree_gen):

trial	# iter	MSE
1	449	0.09594558693216323
2	2426	**0.037812805567220026**
3	1013	0.06499097208555954
4	459	0.0780488070716398
5	1074	0.08448263026652983
6	364	0.1028408894920673
7	130	0.10876526471911088
8	1272	0.0754834525620049
9	111	0.19405066329658477
10	1308	0.07290902308595913

Table 1 the results of genetic programming for the symbolic regressions

In order to generate better symbolic regressions, I stopped the routine when it has same best MSE in populations. And I restarted the algorithm from a randomly initialized population. The **Table 1** shows the results of 10 trials. The best one is the following expression with the MSE **0.037812805567220026**:

```
x33 -92.32563009189113 + abs x13 x12 / 5.113580785878247 x40 /
/ + -0.26417801255433915 sin x16 x44 / / tanh ^ x16 x18 / x56
~ ^ abs x13 x38 + x9 x1 - - ~ / ^ sinh 11.281832921768508 x33
x31 - / abs x19 x13 * -37.999189569333225 - -17.82429985243311
sinh -49.36498179186097 -36.61056027080532 + / / ^ -
17.05761697430297 cos tanh x49 0.584817577656982 ^ cos + ~ /
tanh + cosh exp
```

4. Conclusion

I designed the three search-based algorithms for symbolic regressions. First, I tried to locally search for linear regression for each of coefficients. It is very simple approach but generate a quite good solution. Second one is the genetic algorithm for linear regressions by using coefficients of expressions as their genes. It is much more complex than the first approach, but it is not efficient and converge very soon. Finally, I tried to generate symbolic regressions based on tree structures with genetic programming concept. Although it also converges fast, I stopped the routine if it has same results in some couple of iterations. And then, I iterate the routine in many times. It is much better than just letting it go. Finally, the best one is the linear regression based on the local search with MSE **`0.035763484842246486`**. However, the efficiency of genetic programming for symbolic regression may be better than this approach.