16:09 Code Golf and Obfuscation with Genetic Algorithm Based Symbolic Regression

by JBS

Any reasonably complex piece of code is bound to have at least one lookup table (LUT) containing integer or string constants. In fact, the entire data section of an executable can be thought of as a giant lookup table indexed by address. If we had some way of obfuscating the lookup table addressing, it would be sure to frustrate reverse engineers who rely on juicy strings and static analysis.

For example, consider this C function.

```
char magic(int i) {
    return (89 ^ (((859 - (i | -53)) | ((334 + i) | (i /
           (i & -677)))) & (i - ((i * -50) | i | -47))))
           + ((-3837 << ((i | -2) ^ i)) >> 28) / ((-6925 ^
           ((35 << i) >> i)) >> (30 * (-7478 ^ ((i << i) >>
           19))));
}
```

Pretty opaque, right? But look what happens when we iterate over the function.

```
int main(int argc, char** argv) {
    for(int i=10; i<=90; i+=10) {
        printf("%c", magic(i));
    }
}
```

Lo and behold, it prints "PoC||GTFO"! Now, imagine if we could automatically generate a similarly opaque, magical function to replicate any string, lookup table, or integer mapping we wanted. Neighbors, read on to find out how.

Regression is a fundamental tool for establishing functional relationships between variables in data and makes whole fields of empirically-driven science possible. Traditionally, a target model is selected *a priori* (*e.g.*, linear, power-law, polynomial, Gaussian, or rational), the fit is performed by an appropriate linear or nonlinear method, and then its overall performance is evaluated by a measure of how well it represents the underlying data (*e.g.*, Pearson correlation coefficient).

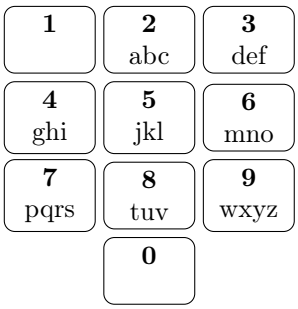
Symbolic regression[30] is an alternative to this in which—instead of the search space simply being coefficients to a preselected function—a search is done on the space of possible functions. In this regime, instead of the user selecting model to fit, the user specifies the set of functions to search over. For example, someone who is interested in an inherently cyclical phenomenon might select C, $A+B$, $A-B$, $A \div B$, $A \times B$, $\sin(A)$, $\cos(A)$, $\exp(A)$, $\sqrt{A}$, and A^B, where C is an arbitrary constant function, A and B can either be terminal or non-terminal nodes in the expression, and all functions are real valued.

Briefly, the search for a best fit regression model becomes a genetic algorithm optimization problem: (1) the correlation of an initial model is evaluated, (2) the parse tree of the model is formed, (3) the model is then mutated with random functions in accordance with an entropy parameter, (4) these models are then evaluated, (5) crossover rules are used among the top performing models to form the next generation of models.

What happens when we use such a regression scheme to learn a function that maps one integer to another, $\mathbb{Z} \to \mathbb{Z}$? An expression, possibly more compact than a LUT, can be arrived at that bears no resemblance to the underlying data. Since no attempt is made to perform regularization, given a deep enough search, we can arrive at an expression which *exactly* fits a LUT!

– — — – — — — — – – –

Please rise and open your hymnals to 13:06, in which Evan Sultanik created a closet drama about phone keypad mappings.



He used genetic algorithms to generate a *new* mapping that utilizes the 0 and 1 buttons to minimize the potential for collisions in encoded six-digit English words. Please be seated.

[30] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

What if we want to encode a keypad mapping in an obfuscated way? Let's represent each digit according to its ASCII value and encode its keypad mapping as the value of its button times ten plus its position on the button.

Character	Decimal ASCII	Keypad Encoding
'a'	97	21
'b'	98	22
'c'	99	23
'd'	100	31
'e'	101	32
'f'	102	33
'g'	103	41
'h'	104	42
'i'	105	43
'j'	106	51
'k'	107	52
'l'	108	53
'm'	109	61
'n'	110	62
'o'	111	63
'p'	112	71
'q'	113	72
'r'	114	73
's'	115	74
't'	116	81
'u'	117	82
'v'	118	83
'w'	119	91
'x'	120	92
'y'	121	93
'z'	122	94

So, all we need to do is find a function `encode` such that for each decimal ASCII value i and its associated keypad encoding k : `encode`$(i) \mapsto k$. Using a commercial-off-the-shelf solver called Eureqa Desktop, we can find a floating point function that exactly matches the mapping with a correlation coefficient of $R = 1.0$.

```
int encode(int i) {
    return 0.020866*i*i+9*fmod(fmod(121.113,i),0.7617)-
           162.5-1.965e-9*i*i*i*i*i;
}
```

So, for any lower-case character c, `encode`$(c) \div 10$ is the button number containing c, and `encode`(c) % 10 is its position on the button.

In the remainder of this article, we propose selecting the following *integer* operations for fitting discrete integer functions C, $A + B$, $A - B$, $-A$, $A \div B$, $A \times B$, $A \hat{} B$, $A \& B$, $A|B$, $A << B$, $A >> B$, $A\%B$, and $(A > B)?A : B$, where the standard C99 definitions of those operators are used. With the ability to create functions that fit integers to other integers using integer operations, expressions can be found that replace LUTs. This can either serve to make code shorter or needlessly complicated, depending on how the optimization is done and which final algebraic simplifications are applied.

While there are readily available codes to do symbolic regression, including commercial codes like Eureqa, they only perform floating point evaluation with floating point values. To remedy this tragic deficiency, we modified an open source symbolic regression package written by Yurii Lahodiuk.[31] The evaluation of the existing functions were converted to integer arithmetic; additional functions were added; print statements were reformatted to make them valid C; the probability of generating a non-terminal state was increased to perform deeper searches; and search resets were added once the algorithm performed 100 iterations with no improvement of the convergence. This modified code is available in the feelies.[32]

The result is that we can encode the phone keypad mapping in the following relatively succinct—albeit deeply unintuitive—integer function.

```
int64_t encode(int64_t i) {
    return ((((-7|2*i)^(i-61))/-48)^(((345/i)<<321)+
           (-265%i)))+((3+i/-516)^(i+(-448/(i-62))));
}
```

This function encodes the LUT using only integer constants and the integer functions $*$, $/$, $<<$, $+$, $-$, $|$, $\oplus$, and %. It should also be noted that this code uses the left bit-shift operator well past the bit size of the datatype. Since this is an undefined behavior and system dependent on the integer ALU's implementation, the code works with no optimization, but produces incorrect results when compiled with `gcc` and `-O3`; the large constant becomes 31 when one inspects the resulting assembly code. Therefore, the solution is not only customized for a given data set; it is customized for the CPU and compiler optimization level.

While this method presents a novel way of obfuscating codes, it is a cautionary tale on how susceptible this method is to over-fitting in the absence of regularization and model validation. Penalizing overly complicated models, as the Eureqa solver did, is no substitute. Don't rely exclusively on symbolic regression for finding general models of physical phenomenon, especially from a limited number of observations!

[31] `git clone https://github.com/lagodiuk/genetic-programming`
[32] `unzip pocorgtfo16.pdf SymbolicRegression/*`