COST ESTIMATION

Assignment 4: Advanced Topics in Software Engineering

Abstract

Symbolic Regression with Kemerer & Miyazaki Datasets.

Garry James McBride

University of Strathclyde

MSc Advanced Computer Science

Contents

Background/Introduction

The task at hand is by using historical data to investigate the use of genetic programming for cost estimation and compare the data selected with existing prediction techniques. Cost estimation is a challenging problem with various parameters and working out how much it is going to cost.

Unfortunately, because of the number of variables involved in the calculation for a cost estimation, it will never be an accurate or exact science, especially with the nature of humans being unpredictable, a factor that makes us far too complex. Furthermore, the complexity of software systems will also be complicated due to the number of tasks, also having complexities that prove to be extremely difficult to judge. There are many different approaches to cost estimation and making predictions, such an algorithmic approach known as 'COCOMO' (which stands for Constructive Cost Model) used to calculate amounts in terms of effort or the time schedule for projects or tasks.

Measured usually in terms of effort, a common metric or value is usually months, person or years, effort is equal to time spent for an individual working for a period. Different fields from specific areas important to an organization are important factors in cost estimation, ranging from management, training or development etc, etc. To summarize cost estimation, it is an important tool, affecting aspects like planning, or budgeting of a certain task or project, whether it be done at the start of a project, or end to both judge what resources are included within the budget regarding time, money, manpower or materials to name a few, and in the end these predictions are compared to the final result.

Datasets

Using the data sets provided:

- **Kemerer**
  The dataset contains 15 lines of data, with 8 attributes which are under the titles of: ID, Language, Hardware, Duration, KSLOC, AdjFP, RAWFP and the output stated as EffortMM. All the attributes are of numeric value.
- **Miyazaki94**
  A similar size to kemerer with the data set containing 48 lines of data, with 9 attributes which are under the titles of: ID, KLOC, SCRN, FORM, FILE, ESCRN, EFORM, EFILE, and the output stated as 'MM'. All the attributes are of numeric value except from the 'ID' attribute that contains 1 non-numerical value followed by 1 numeric value, e.g. 'A1, A2, A3, B1, B2, B3".

Proceeding with these two datasets, they were chosen because of the similarity in size regarding attributes. The difference between the two, is the fact Miyazaki has 33 more lines of data than kemerer, experimenting with population, mutation and crossover rates should prove to be interesting since the one set has more than double the amount of data, but only 1 more attribute. Taking into consideration the length of time one data set takes to run, the smaller ones were chosen as the data 'China' would prove to run for hours, which if in a real-life situation could potentially cost a lot of money, time and effort for a company to do the task at hand.

The 'ID' attributes will be ignored since they are used as identification for the data lines and are not to be calculated because the results would not be correct from the multiplicated attributes to reach the intended goal of the output provided. The output values are also not calculated as they are our 'target' but are recognized and included in the work as the fitness value of how close a calculation is to the target and is then rated by a fitness value, shown in the example below:

a x b / c * d - e + f * g = 134.67 (TARGET)

If the target was 145.67, the example above would have a fitness value of '11' as it is this amount off the intended target. The lower the fitness value the better, as it states it is closer to the target required.

Implementation Details

Upon reflection of the frameworks available, DEAP was selected due to the fact GP (Genetic programming) is a conceptually a simple problem. Due to the straightforwardness of the task at hand, DEAP makes a good introductory example for framework when doing genetic programming. With tasks and structures done differently with DEAP, instead of being limited with predefined types, different ways of generating appropriate types are available now to be used. With the ability to customize initializers (instead of dealing with closed initializers), the ability to choose operators wisely, and the ease of being able to write algorithms that fit all the programmers needs (instead of sealed ones which prove to make the task more difficult) the DEAP framework is most effective for the current project of cost estimation and the chosen datasets being applied/studied.

Using PyCharm (Developed by JetBrains) DEAP is already available to be installed in the interpreter section inside settings. The framework does not present itself right away when installed, the necessary packages must be installed/imported:

- **Operator** - Exports a set of functions implemented in C corresponding to the intrinsic operators of Python
- **Math** – For functions which support complex numbers or calculations
- **Random** – For random number generation
- **NumPy** – For number array objects, sophisticated functions
- **DEAP** – Distributed Evolutionary Algorithms in Python with imported aspects: Creator, GP, base, tools, algorithms
- **ARFF** - for reading and writing Weka ARFF files

After installation of these packages, the frame work must be built. Being used to programming fitness values, creating populations, crossover functions or mutation from scratch, using frameworks was tricky to understand at first but proved to be simple to use once understood. The framework structure was created firstly by adding primitive sets, a starting building block for the individuals, thus the evolution can positively function. A zero-division error was added in case of a situation where a value can be protected against a negative result, which would then in fact crash the program.

Creator was next to develop as symbolic regression needs two objects, easily made with the creator function using ‘Individual’ and ‘Fitness’. Toolbox follows creator as parameters had to be registered specific to the evolution process. Toolbox is DEAP’s way of doing so.

Results

The structure for the framework was set up so it would first start off with adding content to separate lists. Three lists were made for the output, inputs and for 2 data lines at the end of each set to be placed inside for testing later away from the rest of the data population. The data was read, and the rows were split, ignoring 'ID' as stated before and the output attribute, and the last 2 lines were removed and placed into their designated list.

Methods were then created to avoid errors when a 0 is found, replaced by 1, so that the code and progress does not fail when doing mathematic calculations such as Cosine, Sin, Division and other operators. The difference in the size of the data sets were huge at first because the second set chose was 'China' which proved to be much longer to run, due to the hundreds more lines of code and around 10 more attributes. Another method was added to help China run past a certain amount of generations, and when the dataset was switched for another but still bigger than Kemerer (Miyazaki) the same method had to be implanted since it would crash after 500 generations due to the size of the program.

The Algorithm function was writing to state which calculation methods the framework was to use when finding a solution, and then the creator tool for individual and the fitness function were implemented, and then further developed to secure an accurate and good fitness value could be retrieved from the data set. A method for running the algorithm was created at the end and 'print' statements were added to help the ease of knowing which results are for what purpose or where they originated from.

Kemerer Dataset

Having set up functions inside the code to change rates for mutation, crossover, population and generations, the data was run 6 times with different rates ranging from 0.05, 0.2 and 0.1 for mutation since anything higher or lower will not give good results for these aspects. Crossover was kept higher so it would take as many lines of the data to swap around looking for that perfect solution. Generations were kept around 1500 as they tended to end up the same number for a large amount of time, they may or may not have improved for a higher generation, but time is of the essence and the data set already was taking around an hour and 30 minutes to run.

The rates for each run are shown below in the table:

	Population	**Mutation Rate**	**Crossover**	**Generations**
RUN 1	*500*	*0.1*	*0.9*	*1500*
RUN 2	*400*	*0.2*	*0.8*	*1500*
RUN 3	*400*	*0.2*	*0.8*	*1500*
RUN 4	*400*	*0.05*	*0.7*	*1500*
RUN 5	*450*	*0.1*	*0.7*	*1500*
RUN 6	*450*	*0.2*	*0.9*	*1500*

From the results above a mixture of different rates were used for each run, and when a certain formula proved to be more effective than others, it was ran but slightly changed to see if the results would improve, e.g. run 4 and 5 are close in relation but after the 4th run proved to be quick and successful,

the population as bumped up by 50 and the mutation was increased to see the results. The generations were kept at 1500, as some runs would jump at the last minute when using 1000, so 1500 was chosen so that a number would stay the same for a certain amount of iterations to see how accurate it was and to make sure It wasn't going to get better before the program is stopped.

	Fitness value
RUN 1	*0.22235215487841486*
RUN 2	*32.47677795990228*
RUN 3	*11.808645570518603*
RUN 4	*72.87186864027467*
RUN 5	*19.895931441414408*
RUN 6	*179.67904867031834*

The end results from the runs proved to be of really good quality, especially the first run. Where the population was changed, and mutation was increased with runs 4&5 it has proved to be successful since it has gotten better with a decreased of around 50, bearing in mind the smaller the number the higher the fitness as we must get as close to the output as possible.

The two test suites chosen at the end were:

	Test Data Line 1	**Test Dataline 2**
Test Suites	*3.0, 1.0, 14.0, 60.2, 1044.3, 976.0*	*1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0*

There was no random choice when picking the testing data as each run would be different, and it would be more interesting to see the results applied to the same testing data as every other run.

Figure 1

The two tests which had the calculation applied to them at the end were give an estimated effort and then an actual effort rate, followed by a mean average error for both the test suites combined. The numbers for actual in both tests stayed the same throughout all runs, and a dramatic rise and fall were present in the estimated results for test 1 (see figure 1). The same can be said for the second data test, but only at the end where it dropped from high numbers to minus, a strange occurrence which could only be explained because of the high mutation and crossover rate.

	The mean average error for the test suite
RUN 1	*6.061890972198768*
RUN 2	*41.53874675810468*
RUN 3	*66.42133248600359*
RUN 4	*13.847241782059285*
RUN 5	*123.5721538109008*
RUN 6	*515.7619449106343*

Shown above are the results from the combined mean average for both tests. The average is supposed to be the difference between the actual an the estimated from both data lines added together, but only some of the results add up with the majority missing by a long shot. An interesting factor as the fitness value is good, but the results aren't there to back it up. Run 1 and 4 have the best results, with run 6 having the largest by some means.

Miyazaki94 Dataset

Moving onto the Miyazaki dataset, the same rates mutation, crossover, population and generations, are present but are changed in a diverse way again not in the same order as the previous data set. The data was run 6 times again with generations kept around 1500 for the same reasons stated before. The dataset is larger than Kemerer, so the function that was made for China was implemented here to ensure the program makes it past 500 generations, the code was running for around 1 hour and then would crash, an issue that could cause an organization time and money they do not have, this function proved to be successful and made a difference straight away.

The rates for each run are shown below in the table:

	Population	**Mutation Rate**	**Crossover**	**Generations**
RUN 1	*400*	*0.1*	*0.9*	*1500*
RUN 2	*800*	*0.2*	*0.9*	*500*
RUN 3	*800*	*0.05*	*0.9*	*1500*
RUN 4	*700*	*0.2*	*0.8*	*1500*
RUN 5	*700*	*0.2*	*0.8*	*1500*
RUN 6	*750*	*0.2*	*0.9*	*1500*

A large variety of population was chosen in search of interesting results, with the mutation roughly the same and crossover, this dataset concentrated more on population and generations than before since the findings from Kemerer were intriguing. An interesting aspect was to see how well the

program would run with large data at a much smaller population size, 500 was chosen and the results found were extremely interesting.

	Fitness value
RUN 1	*885.9500634362215*
RUN 2	*293.3400093808403*
RUN 3	*305.00350809181487*
RUN 4	*159.24827338624124*
RUN 5	*568.1117872693131*
RUN 6	*123.06126161207459*

From the fitness results above, interesting Run 2, which has 1000 less generations than the rest has much better results than RUN 1 & 3. Considering the drop-in generations, the population was kept at the highest for run 2, this is interesting as Run 1 has a small population, highlighting that the generations aren't necessary as important as population when running these datasets. With a high population and high generation rate, run 6 proves to be the best followed by 4.

	Test Data Line 1	Test Dataline 2
Test Suites	*35.1, 51.0, 0.0, 6.0, 347.0, 0.0, 122.0*	*55.0, 14.0, 24.0, 47.0, 202.0, 504.0, 932.0*

Sticking with only 2 data lines chosen from the end, even thought the data set is much larger than Kemerer, it seems more interesting to compare results that are equal in terms of quantity. The two selected tests are shown above.

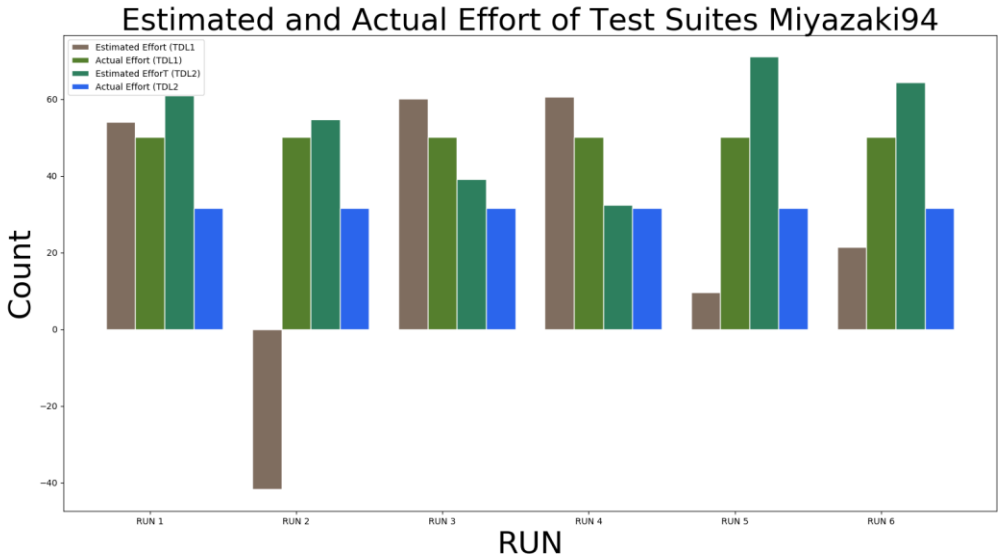

Figure 2

From Figure 2 the numbers for actual effort have stayed the same, which estimate around the same rates, but much smaller than Kemerer, a much larger data set may be the reason behind this since there would be more to crossover to find an adequate solution. Oddly the estimation for test 1

dropped by a large amount in its second run, around the same rating as the other runs but in minus instead. A possible reason for this could be the afore mentioned small population size, strangely though the population size did not affect the full set fitness from the other data lines but has had its impact on the testing suite.

	The mean average error for the test suite
RUN 1	*33.40558633823525*
RUN 2	*68.57453001589347*
RUN 3	*17.492688261146228*
RUN 4	*11.37214192868639*
RUN 5	*0.9850568296358446*
RUN 6	*4.27247291217239*

Complete opposite from Kemerer, Miyazaki has better fitness functions even with the rise in Run 2 because of the strange issue with the smaller population size that was used. The bigger data set seems to be more accurate and reliable than the smaller Kemerer, with better fitness functions, interesting results from drastic changes in population and considering the full set fitness function is 46 data lines together (minus 2 for the tests) the fitness value is extremely close to the target.

What separates both the data sets from their results is how accurate Miyazaki stays throughout, in contrast Kemerer has a strong fitness value from the full set, but then when applying the solution to the tests, it doesn't seem to be keeping the accuracy is set itself to begin with, but Miyazaki keeps this trend and proves to be more interesting, from an observation it would seem the bigger the data set the more accurate/interesting results are found.

Comparison of results

Using Weka, the data sets were applied to some non-GP approaches to see the difference in accuracy or observations that could be found. Weka's Linear Regression was used with the methods of Percentage Split, Full Set and then Cross Validation applied to different results from the GP approach.

Kemerer Percentage Split

	Test Data Line 1		Test Dataline 2	
	Estimated Effort	**Actual Effort**	**Estimated Effort**	**Actual Effort**
RUN 1	*22.67770848311177*	*69.9*	*288.06040054468946*	*246.9*
RUN 2	*159.11177220600118*	*69.9*	*199.22697455210348*	*246.9*
RUN 3	*176.7810996985465*	*69.9*	*206.4402327874571*	*246.9*
RUN 4	*111.68492874213663*	*69.9*	*218.96231303992266*	*246.9*
RUN 5	*32.46499533446852*	*69.9*	*160.7628508546307*	*246.9*
RUN 6	*-49.50453663458771*	*69.9*	*-149.45740827604655*	*246.9*

Previously shown in a grouped bar chart, the table above is the results from the GP using the DEAP framework is shown as a table instead to see more accurately and detailed the results from the running of each test. The Linear Regression will be compared to the Mean Absolute error shown before on page 5.

Summary	**Fitness value**
Correlation coefficient	*0.8022*
Mean absolute error	*112.2523*
Root mean squared error	*158.6368*
Relative absolute error	*130.9954 %*
Root relative squared error	*161.2125 %*
Total Number of Instances	*5*

The table above, highlights in red the result from Linear Regression. The results compared with the Kemerer data set prove to be much higher, which shows the GP has more luck finding a better fitness value than Weka.

Miyazaki Percentage Split

	Test Data Line 1		Test Dataline 2	
	Estimated Effort	**Actual Effort**	**Estimated Effort**	**Actual Effort**
RUN 1	*54.05368468692217*	*50.1*	*60.95190165131309*	*31.5*
RUN 2	*-41.720303610168344*	*50.1*	*54.745773594274866*	*31.5*
RUN 3	*60.018382928662135*	*50.1*	*39.074305332484094*	*31.5*
RUN 4	*60.63599985918429*	*50.1*	*32.3361420695021*	*31.5*
RUN 5	*9.614943170344988*	*50.1*	*71.00000000001917*	*31.5*
RUN 6	*21.483754021504392*	*50.1*	*64.388718890668*	*31.5*

Summary	**Fitness value**
Correlation coefficient	*0.5078*
Mean absolute error	*37.8645*
Root mean squared error	*46.6041*
Relative absolute error	*59.5464 %*
Root relative squared error	*67.3941 %*
Total Number of Instances	*16*

As for Miyazaki, the Linear Regression proves to be around the same quality as the GP approach. Interestly again the bigger data set seems to have less issues than the smaller, shown before on page 7, possibly a factor that comes into play with Linear regression as well.

Kemmer full set

	Fitness value
RUN 1	*0.22235215487841486*
RUN 2	*32.47677795990228*
RUN 3	*11.808645570518603*
RUN 4	*72.87186864027467*
RUN 5	*19.895931441414408*
RUN 6	*179.67904867031834*
Linear	*103.8514*

The table above shows the fitness value from full set after running the code as previously shown before, although this time highlighted in red is our Linear Regression result, which is also full set. It shows a similar pattern where the GP is better, a lot better as it would happen. Interestingly the smaller Data set chooses where it wants to be accurate and where it doesn't, it seems to work better on Weka than it does with a GP approach, but with a GP approach it doesn't show very accurate or reliable results.

Miyakai Full Set

	Fitness value
RUN 1	*885.9500634362215*
RUN 2	*293.3400093808403*
RUN 3	*305.00350809181487*
RUN 4	*159.24827338624124*
RUN 5	*568.1117872693131*
RUN 6	*123.06126161207459*
Linear	*22.8875*

Shown above, as previously shown on page 6 the Miyazaki dataset has improved massively with the use of Linear Regression. Linear Regression is quicker as well and seems to have a better result than the GP. There isn't a single run that comes close to the Linear Regression result, the closet is run 6, but it off by around 100.