Cost Estimation

Mihail Mihaylov

Reg. No: 201888925

Introduction and Background

This report includes a description of the cost estimation assignment, as well as some introduction to genetic programming. This assignment involves implementing a genetic programming algorithm to generate a function based on a set of data with different number of inputs and one output. The idea is to create a function based on a big portion of the data and then test it against data lines which have not been involved in creating the function to determine its effectiveness. Symbolic regression is a part of genetic programming which is used for this purpose. Symbolic regression searches through mathematical expressions to find a model which fits a given dataset best. This project has been implemented using a framework called JGap, which is a Genetic Algorithm and Genetic Programming package written in the programming language Java. JGap contains all the necessary methods to implement a symbolic regression model which can be used on different data sets. The data sets used in this case are .arff file (Attribute-Relation File Format). From the different data sets supplied, the 2 chosen to be run are called Albrecht and Kemerer. They are both data sets which only use numerical values. The first data set Albrecht contains 8 attributes: Input, Output, Inquiry, File, FPAdj, RawFPcounts, AdjFP and Effort. The first 7 are used as input attributes and the output is the Effort. Using those input attributes the aim was to develop a function which can determine the effort. The second data also contains 8 attributes: ID, Language, Hardware, Duration, KSLOC, AdjFP, RAWFP and EffortMM. From those 8 attributes the first one (ID) was ignored, because it only shows the number of the data line and is not relevant to determining the output (EffortMM). The other 6 attributes were used as inputs to generate the function to calculate the output.

Implementation Details

For the implementation there were 2 Java classes created for each data set. One of the classes contains the actual algorithm and configuration and the other one the fitness function. First of all the data was read and stored into different ArrayList depending on its attributes. Once that is done a configuration is created and Variable objects (which are part of the JGap library) were created to match for each input attribute. After the variables were created a GPFitnessEvaluator was selected as DeltaGPFitnessEvaluator. This was selected as such because of the need to compute a defect rate. After data some basic configurations were set which involved the maximum initial depth of the population which was set to 4, the size of the population (set to 1500), the maximum depth of nodes to crossover (set to 8), then the program creation strictness was set to false which ensures that in case of an error with a certain solution it would continue running and mark this solution as a bad solution. Also, in order to ensure only the best individuals from a population proceed to the next generation, elitism was activated. This is the main configuration of the program. Other values such as the crossover and mutation are already pre-set and after running the algorithm a few times it was determined that the original values gave better solutions. The crossover is set to 0.9 which is in 90% of the cases a crossover occurs and the mutation is set to 0.1 which means in 10% of the cases a mutation occurs.

After the initial configuration was set the creation method is implemented. Which contains the type of variables used, the inputs and the types of operands. It also creates the initial population and sets the maximum number of nodes for a solution. The type of variables used was a Double. The operands used are: add, subtract, multiply, divide, add3 (which is to add 3 variables), add4 (adds 4 variables), multiply3 (multiplies 3 variables), Logarithm, Exponential, Modulus, Sine, Cosine, Tangent, Arcsine, Arccosine and Arctangent. The maximum number of nodes a solution can have is set to 500 (which would provide a very long model but increases the accuracy of the prediction).

The fitness function class is used to implement a fitness function to determine how good a model is. The type of fitness used in this project is the mean absolute error. This determines how good a model is by running it with all the available data lines and generating an estimated effort. After that the deviation between the estimated effort and the actual effort recorded in the data set is added together for all the lines and the returned value is the fitness. This means that the closer that value is to 0, the closer the estimated value is to the actual value and also, of course, the better the solution. In the case where 0 fitness is achieved, it means that for the full training data set provided all the estimated effort values match the actual effort values perfectly.

Another part which is worth describing is the testing. Once the algorithm is ran it needs to be tested on data lines. For this purpose, for the Albrecht data set, the last 2 data lines, out of 24 in total, were omitted when a model was generated and after the algorithm had finished running those 2 lines were evaluated using the all time best solution and comparing against the actual recorded effort provides a mean absolute error value for the testing suite. For the Kemerer data set, which contains only 15 data lines, the last one was omitted to be used as a test suite at the end.

The number of times the initial population was evolved is set to 2000. In most cases it would reach the best possible value for the run somewhere between 1300th and 1700th generation. Those values only work like that if the population is set to over 1500 and the maximum number of nodes is set to 500.

Presentation of results

Both data sets were run numerous times with different population, evolution and maximum nodes values. Multiple results were obtained with different fitness values.

Albrecht

The best fitness value obtained for the Albrecht data set was 4.05. The results from this run can be seen below:

Best solution fitness: 4.05
Best solution: (((((File * (File * (cosine (((log File) + (sine ((sine (4.0 % (Exp(FPAdj)))) + ((sine (Input - (Input - (sine (sine (Input - (Input - (Input - Output)))))))) - (cosine RawFPcounts)))) + (log File) + FPAdj) % (tangent Inquiry))))) / ((((sine (Input - ((Input - ((Input - Output) - (sine ((5.0 % Inquiry) + ((sine ((sine (Input - (Input - ((sine (Input - (Exp((abs (Input - Output)))))) - (abs (((sine RawFPcounts) % (log FPAdj)) - Output)))))) - (cosine RawFPcounts))) - (cosine RawFPcounts)))))) * FPAdj))) - (cosine ((File * (File * (abs ((log FPAdj) + ((sine (File * (Inquiry % (log FPAdj)))) + (sine RawFPcounts)))))) / (sine (4.0 % (Exp(FPAdj))))))) * FPAdj) + File)) + (((sine (File * ((File * (File * (abs ((log FPAdj) + (((abs (sine ((Exp((abs (abs (abs (abs (Input - Output)))))) * ((Input - (Input - Output)) - (FPAdj + File))))) - ((Inquiry - RawFPcounts) * FPAdj)) % (Exp(FPAdj))))))) * ((sine (Input - Output)) % (log FPAdj))))) + (((Input - Output) % (tangent ((Input - (sine (Exp((sine (sine ((((abs (Input - (Input - (Input - Output)))) - ((abs (Input - (Input - (sine (abs (Input - Output)))))) % Inquiry)) % (Exp(FPAdj))) + (abs (Input - Output)))))))) % Inquiry))) + (abs (Input - Output)))) * (cosine (Exp(FPAdj))))) / (Exp(FPAdj))) / (arc_tangent ((5.0 - (Exp((sine ((abs (8.0 - ((sine (abs ((5.0 % Inquiry) + ((Inquiry - RawFPcounts) + Inquiry + File)))) % (Input - Output)))) + File))))) / (sine (sine ((sine ((sine RawFPcounts) + (abs (sine ((sine ((sine RawFPcounts) + (abs (abs (abs (abs (abs (abs (Input - Output)))))))))) + (abs ((sine (RawFPcounts * AdjFP)) + (((sine (sine (abs (abs (abs (abs (Input - Output))))))) % (log FPAdj)) * ((sine (File * Input)) + ((log FPAdj) + ((7.0 % (Exp(FPAdj))) + (sine ((sine ((sine (sine ((sine (sine ((sine FPAdj) + (sine (File * Input))))) + (sine ((Inquiry - RawFPcounts) * FPAdj))))) + ((sine (sine (6.0 % Inquiry))) - (cosine RawFPcounts)))) + (((sine RawFPcounts) + File) % FPAdj))))))))))))) + (abs ((sine ((sine (4.0 % Inquiry)) + (abs (abs (Input - Output))))) + (((Inquiry - RawFPcounts) % (log (sine (7.0 % Inquiry)))) *

FPAdj))))))))) + (((sine (((File * (File * (abs ((sine ((sine (5.0 % Inquiry)) + (abs (Input - (Input - (sine (abs (abs (Input - Output)))))))))) + (((Inquiry - RawFPcounts) % (log FPAdj)) * FPAdj))))) / (Exp(FPAdj))) + (abs (FPAdj + File)))) + (abs (((sine ((sine ((Input - ((Inquiry - RawFPcounts) * FPAdj)) % (Exp(FPAdj)))) + ((sine (Input - (Input - (Input - Output)))) - (cosine RawFPcounts)))) + ((Inquiry - RawFPcounts) % FPAdj)) + ((FPAdj + (abs ((sine (sine (sine (abs ((5.0 % Inquiry) + (6.0 + Inquiry + File)))))) + (((sine (sine ((sine ((FPAdj + (abs (Input - Output))) * (cosine ((Inquiry % Inquiry) - (sine (abs (sine ((Exp((abs (Exp((abs (Input - Output))))))) * (sine (sine (4.0 % Inquiry))))))))))) - Input))) + (abs (Input - Output))) * (cosine (Exp(FPAdj))))))) * FPAdj)))) * FPAdj)

Depth of chromosome: 30

The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.

The estimated effort is: 7.198617318191118

The actual effort according to the data is: 6.1

The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.

The estimated effort is: 3.4933460913354755

The actual effort according to the data is: 0.5

The best solution was obtained by running the algorithm using the first 22 out of 24 data lines. After the solution with 4.05 fitness value was obtained the 23rd and 24th data lines were run using that solution to determine how well it does with unexplored data. From the 24th line the estimated effort using this solution was ~7.2 where the actual effort according to the data is: 6.1. From the 23rd line the estimated effort was ~3.49 where the actual effort recorded was 0.5. This means that the mean average error on the test suite is 4.09 which is really close to the mean average error from the training data set (1st to 22nd data line). When the algorithm was run a graphical node tree was produced which can be seen in figure 1 below. In the node tree below, each node represents an operand and each edge is a value.

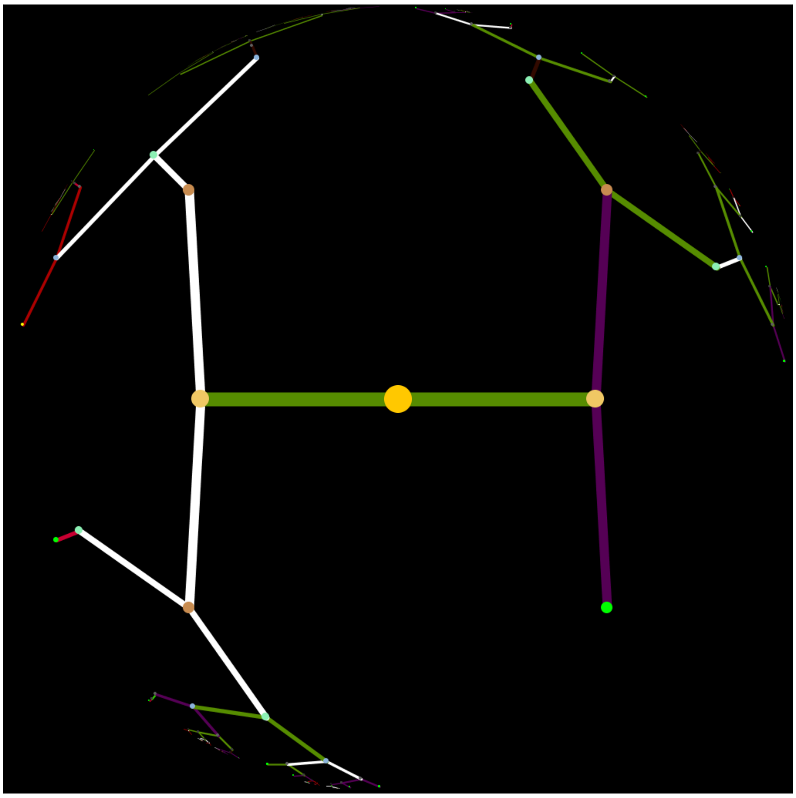

Figure 1: Albrecht best sample run node tree with fitness: 4.05

Kemerer

The best fitness value obtained for the Kemerer data set was 4.41. The results from this run can be seen below.

Best solution fitness: 4.41

Best solution: ((sine (Hardware + ((log Hardware) + ((7.0 + (KSLOC / Language) + ((abs ((KSLOC + (tangent RawFP)) + ((log Hardware) * (tangent (abs (abs (tangent (sine (tangent (sine (tangent (cosine (tangent (tangent RawFP)))))))))) * (tangent Hardware)))) / RawFP) + (abs (cosine (Exp(Language))))) / Language) + (cosine (KSLOC + (KSLOC + Duration + 5.0) + (((log RawFP) * (tangent (sine (sine Hardware))) * 6.0) + (KSLOC / Language) + (abs ((tangent AdjFP) + (tangent AdjFP)))) + (AdjFP % (tangent RawFP))))))) * (abs (KSLOC + 4.0)) * (tangent (tangent AdjFP))) + (KSLOC + (abs (log (Hardware * Hardware * ((((abs (tangent (cosine (tangent (tangent RawFP))))) * (abs (tangent ((tangent AdjFP) + (AdjFP % (sine (cosine (log Language))))))) * (((KSLOC + (((tangent (((cosine (KSLOC + Duration + 2.0)) + (tangent AdjFP) + (((sine (tangent RawFP)) * Duration * 6.0) + Duration + KSLOC + (tangent RawFP)) + Hardware) / Language)) + ((tangent AdjFP) + (tangent AdjFP)) + (log RawFP)) + (((tangent (log Hardware)) + (sine (cosine (tangent (tangent RawFP)))) + Language + (tangent AdjFP)) + (tangent AdjFP) + (((((tangent AdjFP) + (tangent AdjFP)) + (tangent ((((tangent AdjFP) + (tangent AdjFP) + (log RawFP)) + (tangent (sine (tangent Hardware)))) + (tangent (sine (tangent RawFP))) + (tangent AdjFP))) + (log RawFP)) + (tangent (sine (sine Hardware)))) + (tangent AdjFP) + (tangent AdjFP)))) + 5.0) * ((tangent RawFP) + (tangent (KSLOC + (KSLOC + Duration + 5.0) + (((log RawFP) * (tangent (sine (tangent (log Hardware)))) * 6.0) + (KSLOC / Language) + (abs ((tangent AdjFP) + (tangent AdjFP)))) + (((tangent RawFP) + (tangent (((cosine ((tangent AdjFP) + Duration + 6.0)) + (tangent AdjFP) + (((sine (tangent RawFP)) * Duration * 6.0) + Duration + KSLOC + (tangent RawFP)) + Hardware) / Language)) + (log RawFP)) % (tangent RawFP)))) + ((tangent RawFP) + (tangent AdjFP) + (log RawFP))) * ((tangent RawFP) + (tangent AdjFP) + (abs ((tangent RawFP) + (tangent (((cosine (KSLOC + Duration + 2.0)) + (tangent AdjFP) + (((sine (tangent RawFP)) * Duration * 6.0) + Duration + KSLOC + (tangent RawFP)) + Hardware) / Language)) + (log RawFP))))) + 6.0 + ((sine ((AdjFP + (Language * Duration * 2.0) + Language + (cosine ((log RawFP) + (log Hardware) + (tangent ((((cosine (KSLOC + Duration + 2.0)) + (tangent AdjFP) + (((log RawFP) * (tangent (sine (tangent (sine (tangent (sine (Duration - 7.0))))))) * 6.0) + Duration + KSLOC + (tangent Hardware)) + Hardware) + ((arc_tangent 4.0) * Duration * 6.0) + Language + (tangent (log RawFP))) / Language)) + (abs ((tangent AdjFP) + (tangent AdjFP)))))) / (tangent (sine RawFP)))) * Duration * (log RawFP)))) * ((abs (((tangent (cosine (log RawFP))) + (RawFP / KSLOC)) / Language)) + ((abs (tangent AdjFP)) + (tangent (sine (sine Hardware)))) + (tangent (sine (sine Hardware)))) * KSLOC) + (tangent (log Hardware)) + (sine Hardware))))) + (tangent (((cosine (sine (tangent RawFP))) + ((sine (tangent RawFP)) * Duration * 6.0) + Language + (tangent AdjFP)) / Language)) + (((tangent ((sine (log RawFP)) * ((tangent RawFP) + (tangent AdjFP) + (log RawFP)) * ((tangent (((tangent AdjFP) + (AdjFP % (abs (abs (cosine (Exp(Language))))))) / Language)) % (RawFP / KSLOC)))) + (AdjFP % (abs ((tangent AdjFP) + (tangent AdjFP) + (((tangent AdjFP) + (AdjFP % (abs ((tangent AdjFP) + (tangent AdjFP) + (log RawFP))))) / Language))))) + (tangent (tangent (((cosine (sine (tangent RawFP))) + (sine (tangent (cosine (tangent RawFP)))) + Language + (tangent AdjFP)) / Language))))) + ((sine (tangent RawFP)) * Duration * 6.0)

Depth of chrom: 28

The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value:

The estimated effort is: -15.480335376045943

The actual effort according to the data is: 69.9

The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value:

The estimated effort is: 59.93953739422466

The actual effort according to the data is: 246.9

The best fitness value solution was obtained using 13 out of the 15 data lines in the data set for training purposes and the other 2 for testing purpose. Even though the fitness value achieved on the training set was 4.41, when the solution was ran using the testing suite the estimated effort values were not very close to the actual effort ones. In the case of the 15th data line the estimated result is -15.48 compared to the actual one which is 69.9. For the 14th data line the estimated result is 59.94 compared to the actual effort which is 246.9. This gives a mean average error for the testing suite of 272.64, which is very different from the training suite fitness function, thus proving this solution ineffective against new entries even though it is effective for the training set ones. When the algorithm was run a graphical node tree was produced which can be seen in Figure 2 below. In the node tree below, each node represents an operand and each edge is a value.

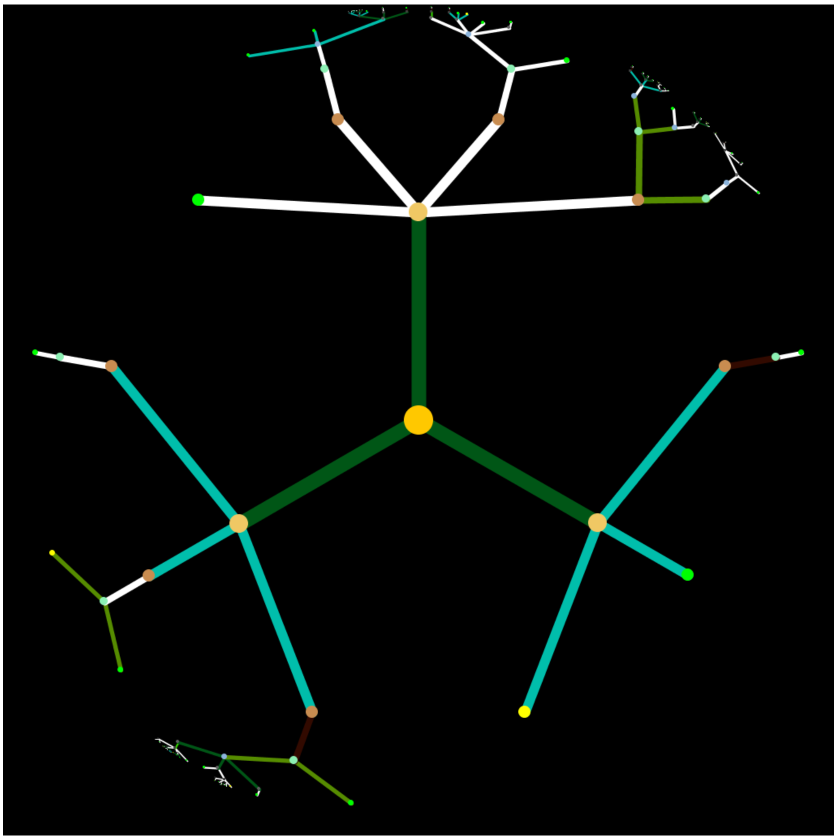

Figure 2: Kemerer best sample run node tree with fitness: 4.41

Even though the solution with the best fitness on the training set proved ineffective against the testing suite there was another solution obtained using 14 out of 15 data lines for the training set and only 1 line for the testing suite which proved a lot more effective even though it had a worse fitness value. The said solution had a mean average error value of 17.38 for the training set but it provided an estimated effort for the training set line of 76.84 compared to the actual 69.9, which makes the mean average error from this solution for the testing suite 6.94. The rest of the sample runs can be seen in Appendix 1.

Comparison of results

Other non-GP approaches were examined using the WEKA software. This part will show 2 non-GP approaches results from the same datasets as the symbolic regression and compare them. The first one was linear regression. This was run with both the Albrecht and Kemerer data set and it was examined in 2 different scenarios. One of them was using the full data set as a training set and the other one was to split the data at 90% for the training and 10% for the testing suite, which for

Albrecht it means 2 out of the 24 data lines were used for testing and for the Kemerer it's 1 out of the 15 data lines (same as the Symbolic Regression approach described before). The results from those can be seen in Figure 3 and Figure 4 below respectively.

```
=== Run information ===
Relation:     albrecht
Instances:    24
Attributes:   8
              Input
              Output
              Inquiry
              File
              FPAdj
              RawFPcounts
              AdjFP
              Effort
Test mode:    evaluate on training data

=== Classifier model (full training set) ===


Linear Regression Model

Effort =

      0.1589 * Input +
      0.3321 * Output +
      0.5048 * Inquiry +
      0.3532 * File +
    -14.8689

Time taken to build model: 0.05 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.962
Mean absolute error                      6.1364
Root mean squared error                  7.5934
Relative absolute error                 32.5358 %
Root relative squared error             27.295  %
Total Number of Instances               24
```

Figure 3: Linear regression full data set Albrecht

```
=== Run information ===
Relation:     albrecht
Instances:    24
Attributes:   8
              Input
              Output
              Inquiry
              File
              FPAdj
              RawFPcounts
              AdjFP
              Effort
Test mode:    split 90.0% train, remainder test

=== Classifier model (full training set) ===


Linear Regression Model

Effort =

      0.1589 * Input +
      0.3321 * Output +
      0.5048 * Inquiry +
      0.3532 * File +
    -14.8689

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correlation coefficient                  1
Mean absolute error                     23.499
Root mean squared error                 28.8023
Relative absolute error                 53.5507 %
Root relative squared error             46.827  %
Total Number of Instances                2
```

Figure 4: Linear regression with 90% training set

As can be seen from the figures above the Mean Absolute Error for both linear regressions is worse than the one achieved using the Genetic Programming detailed in the previous sections. However as can also be seen from these figures the model was obtained after less than a second and for the Symbolic Regression Genetic Programming it took a couple of hours to achieve those results along with numerous runs. Another huge difference is that the model created using linear regression is fairly simple and easy to understand and calculate if necessary, while the symbolic regression has a lot more nodes and edges (a bit short of 500). Please see below Figure 5 and 6 showing the Gaussian processes ran with the same data sets in the same two scenarios (1 with full training set and 1 with test split 90/10).

```
Relation:     albrecht
Instances:    24
Attributes:   8
              Input
              Output
              Inquiry
              File
              FPAdj
              RawFPcounts
              AdjFP
              Effort
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Gaussian Processes

Kernel used:
  Linear Kernel: K(x,y) = <x,y>

All values shown based on: Normalize training data

Average Target Value : 0.20415472779369626
Inverted Covariance Matrix:
    Lowest Value = -0.17440338778430498
    Highest Value = 0.9866920601468916
Inverted Covariance Matrix * Target-value Vector:
    Lowest Value = -0.26145225204439054
    Highest Value = 0.24478839798311944



Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.9242
Mean absolute error                     11.6491
Root mean squared error                 14.1956
Relative absolute error                 61.7646 %
Root relative squared error             51.0273 %
```

Figure 5: Gaussian Processes Full data set Albrecht

```
Attributes:   8
              Input
              Output
              Inquiry
              File
              FPAdj
              RawFPcounts
              AdjFP
              Effort
Test mode:    split 90.0% train, remainder test

=== Classifier model (full training set) ===

Gaussian Processes

Kernel used:
  Linear Kernel: K(x,y) = <x,y>

All values shown based on: Normalize training data

Average Target Value : 0.20415472779369626
Inverted Covariance Matrix:
    Lowest Value = -0.17440338778430498
    Highest Value = 0.9866920601468916
Inverted Covariance Matrix * Target-value Vector:
    Lowest Value = -0.26145225204439054
    Highest Value = 0.24478839798311944



Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correlation coefficient                  1
Mean absolute error                     39.2591
Root mean squared error                 53.2138
Relative absolute error                 89.4655 %
Root relative squared error             86.5154 %
Total Number of Instances                2
```

Figure 6: Gaussian Processes with 90% training set Albrecht

As can be seen again from the Figures above the Gaussian Processes are a lot faster than Symbolic Regression but the results achieved from them are worse, considering that the run of the Guassian Processes gave a 39.26 Mean absolute error value when run with a test split of 90% while the Symbolic regression gave a best result of 4.09 for the Mean absolute error for the testing suite. However the symbolic regression took about 50 runs before it produced a result like that.

For the Kemerer data set the same algorithms were used as for the Albrecht but with only 1 data line out of 15 in the testing set and 14 in the training set. The results from both the Linear regression and Gaussian processes with both the full data set and the 90% training set can be seen in Figures 7 through to 10 below.

```
=== Run information ===
Instances:    15
Attributes:   7
              Language
              Hardware
              Duration
              KSLOC
              AdjFP
              RAWFP
              EffortMM
Test mode:    evaluate on training data

=== Classifier model (full training set) ===


Linear Regression Model

EffortMM =

     53.4674 * Hardware +
      0.389  * AdjFP +
   -294.1583

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.8301
Mean absolute error                    103.8514
Root mean squared error                141.7055
Relative absolute error                 67.6453 %
Root relative squared error             55.7598 %
Total Number of Instances               15
```

Figure 7: Linear Regression Full Data Set Kemerer

```
=== Run information ===
Instances:    15
Attributes:   7
              Language
              Hardware
              Duration
              KSLOC
              AdjFP
              RAWFP
              EffortMM
Test mode:    split 90.0% train, remainder test

=== Classifier model (full training set) ===


Linear Regression Model

EffortMM =

     53.4674 * Hardware +
      0.389  * AdjFP +
   -294.1583

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correlation coefficient                  0
Mean absolute error                    374.8564
Root mean squared error                374.8564
Relative absolute error                516.3871 %
Root relative squared error            516.3871 %
Total Number of Instances                1
```

Figure 8: Linear Regression with 90% training set Kemerer

The results from the Linear regression show a Mean absolute error (MAE) value of 103.85 for the full data set and a Mean absolute error value of 374.86 when the data is split into 90% training set and 10% training set which for the Kemerer data file that involves 14 out of 15 for the training and 1 for the testing set. Unlike the Albrecht data set the Kemerer data set gave a worse MAE value for the test set when the MAE value for the training set was at it's lowest. The symbolic regression using the same test split as described above gave an MAE value of 17.38 for the training set but it provided an MAE for the testing suite 6.94. Those results from the Symbolic regression prove better than the results obtained from the Linear regression in Figure 7 and 8 above as well as the results from the Gaussian Processes from Figure 9 and 10 below. For the 90/10 split the Kemerer data set gave a MAE of 374.86 for its testing set, while the Gaussian process gave an MAE of 34.02 for its testing set. Even though those results are a bit worse for fitness the solution for linear regression is a lot simpler than the one for symbolic regression it is also a lot faster.

```
Instances:    15
Attributes:   7
              Language
              Hardware
              Duration
              KSLOC
              AdjFP
              RAWFP
              EffortMM
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Gaussian Processes

Kernel used:
  Linear Kernel: K(x,y) = <x,y>

All values shown based on: Normalize training data

Average Target Value : 0.18083712292418053
Inverted Covariance Matrix:
    Lowest Value = -0.2664479619898656
    Highest Value = 0.9965572676280858
Inverted Covariance Matrix * Target-value Vector:
    Lowest Value = -0.23918117413719217
    Highest Value = 0.588027672093561



Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.6876
Mean absolute error                    152.5983
Root mean squared error                210.0956
Relative absolute error                 99.3974 %
Root relative squared error             82.6706 %
```

Figure 9: Gaussian Processes Full data set Kemerer

```
Instances:    15
Attributes:   7
              Language
              Hardware
              Duration
              KSLOC
              AdjFP
              RAWFP
              EffortMM
Test mode:    split 90.0% train, remainder test

=== Classifier model (full training set) ===

Gaussian Processes

Kernel used:
  Linear Kernel: K(x,y) = <x,y>

All values shown based on: Normalize training data

Average Target Value : 0.18083712292418053
Inverted Covariance Matrix:
    Lowest Value = -0.2664479619898656
    Highest Value = 0.9965572676280858
Inverted Covariance Matrix * Target-value Vector:
    Lowest Value = -0.23918117413719217
    Highest Value = 0.588027672093561



Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correlation coefficient                  0
Mean absolute error                     34.0228
Root mean squared error                 34.0228
Relative absolute error                 46.8685 %
Root relative squared error             46.8685 %
Total Number of Instances                1
```

Figure 10: Gaussian Processes with 90% training set Kemerer

The results above show that the Symbolic Regression implemented in this project provides more accurate results but is a lot slower and more complicated than other non-GP approaches.

Appendix 1: Albrecht and Kemerer Sample test runs

Albrecht data set sample runs

The actual solutions were omitted due to their large size.
Run: 1 Max nodes: 500 Population: 2000 Iterations: 2000
[JGAP][15:13:56] INFO GPGenotype - Best solution fitness: 17.15
[JGAP][15:13:56] INFO GPGenotype - Depth of chrom: 31
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 5.26870613140335
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value

The estimated effort is: 1.5085098786104074
The actual effort according to the data is: 0.5

Run: 2 Max nodes: 500 Population: 2000 Iterations: 2000
[JGAP][15:22:43] INFO GPGenotype - Best solution fitness: 47.25
 [JGAP][15:22:43] INFO GPGenotype - Depth of chrom: 41
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 6.78406240321787
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: 6.057316624783443
The actual effort according to the data is: 0.5

Run: 3 Max nodes: 450 Population: 2000 Iterations: 2000
[JGAP][15:35:59] INFO GPGenotype - Best solution fitness: 35.23
 [JGAP][15:35:59] INFO GPGenotype - Depth of chrom: 30
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 10.600000000000005
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: 4.447500000000008
The actual effort according to the data is: 0.5

Run: 4 Max nodes: 400 Population: 2000 Iterations: 2000
[JGAP][15:48:04] INFO GPGenotype - Best solution fitness: 19.26
 [JGAP][15:48:04] INFO GPGenotype - Depth of chrom: 35
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 1.7105777428146087
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: -1.0905777087540816
The actual effort according to the data is: 0.5

Run: 5 Max nodes: 350 Population: 2000 Iterations: 2000
[JGAP][15:57:00] INFO GPGenotype - Best solution fitness: 96.76
 [JGAP][15:57:00] INFO GPGenotype - Depth of chrom: 32
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 2.547784377720557
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: 3.96274055811757
The actual effort according to the data is: 0.5

Run: 6 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][02:12:46] INFO GPGenotype - Best solution fitness: 4.05
[JGAP][02:12:46] INFO GPGenotype - Depth of chrom: 30
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 7.198617318191118
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: 3.4933460913354755
The actual effort according to the data is: 0.5

Run: 7 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][02:25:46] INFO GPGenotype - Best solution fitness: 39.34
[JGAP][02:25:46] INFO GPGenotype - Depth of chrom: 37
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 0.9676507465691606
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: 2.821986347046825
The actual effort according to the data is: 0.5

Run: 8 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][14:56:05] INFO GPGenotype - Best solution fitness: 52.53
[JGAP][14:56:05] INFO GPGenotype - Depth of chrom: 53
The data entry: 12.0, 15.0, 0.0, 15.0, 0.95, 273.68, 260.0 was used to test the function with the best fitness value.
The estimated effort is: 4.165906007355417
The actual effort according to the data is: 6.1
The data entry: 15.0, 15.0, 6.0, 3.0, 1.05, 189.52, 199.0 was used to test the function with the best fitness value.
The estimated effort is: 1.8899647623767073
The actual effort according to the data is: 0.5
The Mean Absolute Error from the test suite is: 3.32405875502129

Kemerer data set sample runs

Run: 1 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][02:21:22] INFO GPGenotype - Best solution fitness: 14.51
[JGAP][02:21:22] INFO GPGenotype - Depth of chrom: 28
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 21.035801925392477
The actual effort according to the data is: 69.9
The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value.
The estimated effort is: 411.8435572588876
The actual effort according to the data is: 246.9

Run: 2 Max nodes: 500 Population: 1500 Iterations: 2000

[JGAP][02:44:43] INFO GPGenotype - Best solution fitness: 4.41
[JGAP][02:44:43] INFO GPGenotype - Depth of chrom: 28
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: -15.480335376045943
The actual effort according to the data is: 69.9
The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value.
The estimated effort is: 59.93953739422466
The actual effort according to the data is: 246.9

Run: 3 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][02:38:15] INFO GPGenotype - Best solution fitness: 41.49
[JGAP][02:38:15] INFO GPGenotype - Depth of chrom: 28
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 93.1090001905442
The actual effort according to the data is: 69.9
The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value.
The estimated effort is: 172.81549945190716
The actual effort according to the data is: 246.9

Run: 4 Max nodes: 500 Population 1500 Iterations: 2000
[JGAP][03:02:42] INFO GPGenotype - Best solution fitness: 53.34
[JGAP][03:02:42] INFO GPGenotype - Depth of chrom: 29
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 146.35243268614124
The actual effort according to the data is: 69.9
The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value.
The estimated effort is: 198.99464300568863
The actual effort according to the data is: 246.9

Run: 5 Max nodes: 500 Population 1000 Iterations: 2000
[JGAP][03:04:55] INFO GPGenotype - Best solution fitness: 10.77
[JGAP][03:04:55] INFO GPGenotype - Depth of chrom: 29
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0
was used to test the function with the best fitness value
The estimated effort is: 61.652627484841545
The actual effort according to the data is: 69.9
The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value.
The estimated effort is: 19.25962512256283
The actual effort according to the data is: 246.9

Run: 6 Max nodes: 500 Population 1500 Iterations: 2000
[JGAP][03:11:25] INFO GPGenotype - Best solution fitness: 16.47
[JGAP][03:11:25] INFO GPGenotype - Depth of chrom: 39
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0

was used to test the function with the best fitness value
The estimated effort is: 1.6674177535727306
The actual effort according to the data is: 69.9
The data entry: 1.0, 1.0, 26.0, 164.8, 1347.5, 1375.0 was used to test the function with the best fitness value.
The estimated effort is: 182.6652102694292
The actual effort according to the data is: 246.9

Run: 7 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][03:25:54] INFO GPGenotype - Best solution fitness: 17.38
[JGAP][03:25:54] INFO GPGenotype - Depth of chrom: 19
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 76.84466016619807
The actual effort according to the data is: 69.9
Run: 8 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][04:44:12] INFO GPGenotype - Best solution fitness: 12.34
[JGAP][04:44:12] INFO GPGenotype - Depth of chrom: 36
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 231.12511806071785
The actual effort according to the data is: 69.9

Run: 9 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][15:14:34] INFO GPGenotype - Best solution fitness: 30.18
[JGAP][15:14:34] INFO GPGenotype - Depth of chrom: 33
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 32.5597872863751
The actual effort according to the data is: 69.9
The Mean Average Error for the test suite is: 37.34021271362491

Run: 10 Max nodes: 500 Population: 1500 Iterations: 2000
[JGAP][15:19:24] INFO GPGenotype - Best solution fitness: 32.43
[JGAP][15:19:24] INFO GPGenotype - Depth of chrom: 28
The data entry: 3.0, 1.0, 14.0, 60.2, 1044.3, 976.0 was used to test the function with the best fitness value.
The estimated effort is: 39.96650957153037
The actual effort according to the data is: 69.9
The Mean Average Error for the test suite is: 29.933490428469632