Detecting Multiple Outliers in Regression Data Using Genetic Algorithms

Kelly D. Crawford
Amoco Production Company
kcrawford@amoco.com

Roger L. Wainwright
The University of Tulsa
rogerw@penguin.utulsa.edu

Daniel J. Vasicek
Amoco Production Research
dvasicek@amoco.com

Abstract

In this paper a new technique is presented for detecting multiple outliers in regression datasets using genetic algorithms. Each dataset contained known outliers and the genetic algorithm implementation was exceptionally accurate in detecting these outliers in all of the datasets tested.

The genetic algorithm is an optimization technique based on various biological principles. It is capable of searching for global optima among a vast number of choices. By intelligent but somewhat random generation of subsets of data, potential sets of outliers are identified by minimizing the residual sum of squares produced by the least squares method. Relative improvements across the outlier sets are then analyzed to determine which sets are indeed outlying.

1 Introduction

Finding a solution to most combinatorial optimization problems requires an organized search through the problem space. An unguided search is extremely inefficient since many of these problems have huge search spaces. The genetic algorithm, however, provides an alternative approach to such global optimization problems.

In this paper we investigate genetic algorithms for detecting multiple outliers in regression datasets. The rest of the paper is presented as follows. In Section 2 we review the outlier problem. In Section 3 the fundamentals of genetic algorithms are reviewed. In Section 4 a genetic algorithm for the detection of outliers is described. Results are given in Section 5, and conclusions and future research issues are given in Section 6.

2 Outliers

Researchers are accustomed to inexactness in physical measurements and have developed statistical methods to help deal with error. Outlier detection and rejection is a statistical method for dealing with relatively large errors that overwhelm more delicate analysis.

Outliers often arise because some process outside the scope of the original study becomes important. Data entry error is an example of an error source that is usually considered by the outlier model. But human mistakes are not the only source of outliers. Physical processes such as earthquakes, cosmic ray showers, lightning, rain, and wind may cause occasional fluctuations in data.

Least sum of squares, or more commonly least squares, is a regression technique used to estimate a line (or hyperplane) fit to a set of data. As an illustration, consider a dataset consisting of (x,y) pairs. X is called the explanatory variable and y is called the response variable. Since the data we will be looking at has only one response variable, we will use the convenient geometric analogy of dimension to refer to our datasets. In general, an n-dimensional dataset is one that contains $n-1$

explanatory variables and 1 response variable. We will likewise refer to a single observation in the data as a point.

Least squares regression constructs a line through regression data in such a way that the sum of the squared residuals (distances from each point to the line in the y direction) is minimized. For the 2-dimensional case, the data can be plotted and the outliers can often be located by sight. Multivariate data involves multiple explanatory variables, making the analysis much more visually complex. We will refer to multivariate data as multidimensional.

As an example, consider the Hertzsprung-Russell diagram of the CYG OB1 star cluster, a dataset taken from the field of astronomy [12]. For reference, we will call this the CYG dataset. The data is taken from 47 stars in the direction of Cygnus. X is the log of the effective temperature at the surface of the star, and y is the log of its light intensity. Figure 1 shows a plot of this dataset (note that the x axis is plotted in reverse).

Notice the 4 points in the upper right corner of the plot. These points correspond to giant stars whose measurements do not correlate linearly with those of the other stars. These 4 points are considered to be outliers. Notice also the least squares line constructed with the 4 points included (LS), as opposed to the least squares line constructed without the 4 points ($\widehat{LS}$). Clearly, least squares can be strongly influenced by outliers and could potentially mislead a researcher in interpreting data. However, least squares would be an optimal technique if you knew which data points were outliers and which were not. Note how the line $\widehat{LS}$ in Figure 1 is a good fit for the CYG dataset.

To illustrate, consider a dataset P of size n which consists of two subpopulations, G and E. We refer to subpopulation G as the data points sampled from a "Good" population (non-outlying points), and subpopulation E as the data points sampled from an "Error" population (the outliers). While least squares regression applied to P, or LS_P, does a poor job of helping us locate outliers, LS_G gives us enough information to detect the points in E.

Isolating G, of course, is the goal of detecting outliers, so this may at first sound like we are placing the answer before the solution! But this analysis gives us some important insight into the problem. As is often the case with genetic algorithms, we need to remove many of the often-applied heuristics to get down to the basic problem. We will thus consider the brute force approach to outlier detection.

The major unknowns are: how many outliers exist (i.e. what is $|E|$), and which points are in G? The brute force approach considers all possible subsets of P to be candidates for E. One could run least squares regression on all possible sets E (i.e. $P - E$) where E contains only 1 point, and keep the best fit (the smallest sum of squared residuals). Then repeat this where E contains 2 points and keep the best fit. Then repeat this for E containing 3 points, and so on up to half of the dataset. A more formal algorithm follows.

Given $G \subseteq P$, $E \subset P$, and $G \cup E \Rightarrow P$, we can search for i outliers by constructing S^i (where $S^i \subset P$ and $|S^i| = n - i$) using:

$$\text{for } i = 1 \text{ to } k \text{ (where } 1 \leq k \leq \lfloor \tfrac{n}{2} \rfloor)$$
$$G^i = \{S^i | min_{S^i \subset P}(LS_{S^i})\}$$

That is, G^i is the $S^i \subset P$ that minimizes the summed squared residuals over all possible sets of i outliers.

For each i, there are $\binom{n}{i}$ possible ways to form S^i. In fact, overall there are

$$\sum_{i=1}^{k} \binom{n}{i}$$

possible ways to form $S^1 \dots S^k$. It is virtually impossible to evaluate these sets for large n. For example, with $n = 100$ and $k = 20$, the number of combinations is around 10^{20}, which is intractable on even the fastest supercomputers.

The goal of this analysis is to find a function $f(P, i) = G^i$ that would enable us to compute a list of the sum of squared residuals for each i such that $r_i = LS_{f(P,i)}$ (i.e. $r_i = LS_{G^i}$). As i increases, r_i steadily decreases (unless we have an exact fit scenario where the residual becomes zero, meaning that all of the remaining

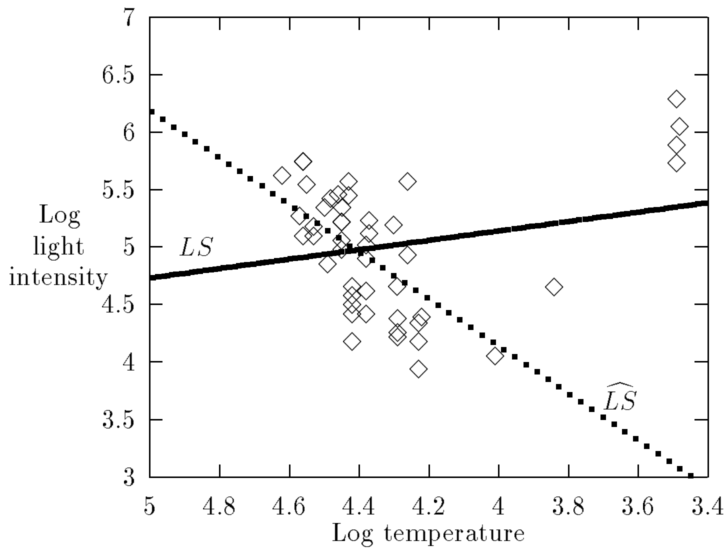

Figure 1: Hertzsprung-Russell diagram for CYG OB1 star cluster

points are exactly on the least squares line). The amount of decrease in the r_i will help us determine if a particular set G^i is outlying.

To accomplish this we can compare the residual for G^i to that of G^{i-1}. If G^i is in fact G, then the difference between r_i and r_{i-1} will be relatively high. Scaling this difference by r_0 produces an estimate for the relative influence of G^i. Computing this measure for all i results in a list of these relative influences. Actual outlier sets can then be detected at points where the influence drops off very sharply or is very high (i.e. relatively close to 1). Our formula for this is

$$(r_{i-1} - r_i)/r_0$$

for $i = 1 \ldots \lfloor \frac{n}{2} \rfloor$.

The above analysis can tend to mask outlier subsets of lesser influence, since changes generally become smaller as i increases. Further information can be gained by using the formula

$$(r_{i-1} - r_i)/r_j$$

for $i = 1 \ldots \lfloor \frac{n}{2} \rfloor$ and $j = 0 \ldots i-1$. As j increases, r_j decreases, thus amplifying the smaller changes in the residuals. This is helpful when determining secondary outlier subsets. We will show some example uses of this technique in the results section.

3 Genetic Algorithms

The genetic algorithm (GA) is a robust search and optimization technique based on the principles of natural genetics and survival of the fittest. Genetic algorithms use the laws of natural selection and genetics to guide a nondeterministic search.

There are several genetic algorithm packages that have been made available to researchers over the past several years. We have implemented our genetic algorithm for outlier detection using LibGA [2]. For the interested reader, Davis, Goldberg and others provide an excellent in-depth study of genetic algorithms [5, 6, 8, 11]. Furthermore, several researchers have investigated the benefits of solving combinatorial optimization problems using sequential and parallel genetic algorithms [1, 7, 10, 13, 14, 15]. It is assumed that the reader is familiar with the fundamental concepts of genetic algorithms.

4 A Genetic Algorithm for Detecting Outliers

The first steps toward constructing a genetic algorithm for outlier detection are defining the encoding of the chromosome (finding a good representation for E), the evaluation function (computing LS_{P-E}), and the recombination operators. Consider a dataset of $n = 20$ points where we suspect the presence of $k = 5$ outliers. A simple encoding of the problem into a chromosome is to use a string of 20 bits, where each bit is associated uniquely with a single data point. A one bit means the data point is used, while a zero bit means the data point is not used. Thus in order to detect the presence of 5 outliers in the dataset, each string of 20 bits must have exactly 5 zero bits, and 15 one bits.

A more compact version (in this case, compactness equates to faster execution) is to use a list of integers corresponding to the outlier set. In the above example, when testing for 5 outliers, we would have a chromosome consisting of 5 integers, each corresponding to a data point to be considered as an outlier.

An initial population of chromosomes is generated randomly such that each chromosome has exactly 5 unique integers (in the range $0 \ldots n-1$). The evaluation function is simply the least squares fit of the 15 remaining data points involved (or $n-k$ where k is the number of outliers being considered).

A recombination operator is applied to the "fittest" chromosomes in such a way that the genetic material of each chromosome is passed on to the children. In addition the recombination operator must make sure that the resulting string after crossover contains no duplicate values. One way to do this is to exchange two randomly chosen values from a chromosome [3]. Occasionally a mutation operator is applied, which in this case is simply another application of the recombination operator.

This technique is called asexual, or one-parent, crossover (which some will argue is no crossover at all, but rather a mutation!). A variant of two-parent uniform order-based crossover (UOX) produces superior results [6]. At each position in the string, a value is randomly chosen from one of the two parents to form the new children. If duplicate values result, they must be removed by selecting a random value outside the current outlier set. This is a major change from our earliest implementation of this algorithm [4].

We found that sorted chromosomes produced the best results. In other words, the list of values in a chromosome is kept in sorted order. The sorted chromosome version found the optimal values more often and with less iterations. Sorted chromosomes were also helpful in changing our insertion algorithm to disallow duplicate chromosomes in the population. Again, this change produced superior results.

The end result of this algorithm will tell us which 5 points to remove in order to produce the best least squares fit of 15 points. In order to detect any outliers, this process must be repeated for $k = 1$ to perhaps as high as $\lfloor \frac{n}{2} \rfloor$. Analysis of each of these results will determine if outliers are indeed present and how many. Note that this algorithm is precisely the f function described in Section 2.

The evaluation function needs to be fast at processing a chromosome. Recomputing a least squares fit from scratch every time is wasteful. Certain information can be kept after initialization that will make this process faster. Using a simple least squares technique, the evaluation function can take advantage of a precomputed matrix of sums.

The data points are represented by the overdetermined system $y = X\beta + \epsilon$, where y is the n vector of dependent variables, X is the $n \times p-1$ matrix of $p-1$ independent variables, β is the $p-1$ vector of coefficients, and ϵ is the n vector of independent errors. Taking $\hat{y} = y - \epsilon$, a least squares solution to the system is $X^T X\beta = X^T\hat{y}$, which results in a matrix of sums [9]. Initially, these sums are computed for the entire dataset and stored in a master matrix. To calculate the residual for a particular chromosome, the sums are copied into a workspace and then reduced by the impact of each point to be thrown out (i.e. each point that is a potential outlier according to the chromosome being evaluated). Since $X^T X$ is a symmetric system, this work is nearly cut in half. After this has been done to all of the points being removed, the system is solved using symmetric Gaussian elimination and back substitution. Based on the resulting vector, a residual is calculated and returned as the fitness for the chromosome under evaluation.

5 Results

There are two parts to our analysis. First, how often did the GA select the best potential outlier sets? Second, how well did our heuristic perform in selecting the correct set of outliers given all sets under consideration? We will begin by looking at the test data used, the GA parameters used for testing, and then we will address the above questions.

5.1 Test Data

We ran tests on 16 different datasets. We selected three of these datasets to illustrate our results. First, the CYG dataset described earlier. Two other datasets we will discuss are called Belgium and Wood. Details of all three datasets can be found in Robust Regression & Outlier Detection by Rousseeuw and Leroy [12].

The Belgium dataset comes from the Belgian Statistical Survey (published by the Ministry of Economy). This dataset plots the number of international phone calls from Belgium in the years 1950-1973. This 24 point, 2-dimensional dataset contains 6 definite outliers, with an additional 2 points that could also possibly be considered as outliers.

The Wood dataset is a 20 point, 6-dimensional dataset containing 4 artificially introduced outliers. Space limitations do not permit us to go into detail on the other 13 datasets, but we can report that our GA was successful in detecting all of the outliers in these datasets as well.

5.2 GA Parameters

The GA was run on each dataset 100 times with different random seeds. Statistics were gathered on the GA's success rate in selecting the outlier sets. The parameters used for the GA tests were: representation = integers, string length = k, for $k = 1 \ldots \lfloor \frac{n}{2} \rfloor$, population size = 10, selection bias = 1.9, mutation rate = 0.05, and maximum number of iterations = 10000.

The Wood dataset, being 6-dimensional and thus more complex, required a population size of 100 and a maximum number of iterations of 20000.

5.3 Selecting Potential Outlier Sets

Of primary importance is the ability of the GA to select the proper sets of potential outliers. Specifically, the GA must be able to find the best set of 2 outliers, the best set of 3 outliers, and so on. The GA was able to do this with a high amount of confidence. Tables 1, 2, and 3 show these best sets for the CYG, Belgium, and Wood datasets, respectively. The average number of iterations required to find the correct set for each k outlier test is shown under heading "average # iters", the minimum residual (from least squares) found is shown under "minimum residual", and the percentage of time (across the 100 trials with different random seeds) the minimum residual was found is under heading "% min found".

From Table 1 we can see that the best set of 4 outliers was found 87% of the time for the CYG dataset. In other words, we ran 100 tests with different random seeds and 87 of those tests resulted in the correct set of 4 outliers. We select 4 here because that happens to be the exact number of outliers in the CYG dataset. Thus we must run the GA with at least 2 different random seeds to be 95% certain of success. On most of the other datasets we registered closer to a 100% success rate, with the rate decreasing gradually as we increased the number of outliers under consideration. This is an expected result related to the exponential increase in the search space. The exception was the multi-dimensional Wood dataset. It is more complex than the other datasets and required more iterations and larger pool sizes to converge properly. Even so, we only found the optimal value 12% of the time. To be 95% certain of finding the right answer (which is also the set of 4

i	*average # iters*	*minimum residual*	*% min found*
0		0.3188090	100
1	47	0.2976430	100
2	423	0.2741670	100
3	1698	0.2490410	46
4	2826	0.1646790	87
5	2202	0.1414550	98
6	1510	0.1161080	63
7	1677	0.1025770	55
8	1716	0.0960415	50
9	2321	0.0895160	40
10	2508	0.0836580	51
11	3363	0.0792069	46
12	3986	0.0734432	47
13	3725	0.0686084	54
14	4204	0.0631716	47
15	2882	0.0592706	50
16	3202	0.0561965	62
17	3581	0.0526319	60
18	4081	0.0500502	60
19	3622	0.0459587	41
20	3686	0.0431333	41

Table 1: GA results for CYG data

i	*average # iters*	*minimum residual*	*% min found*
0		31.6107000	
1	24	25.0392000	100
2	148	19.5838000	100
3	215	14.7047000	100
4	342	9.9381700	100
5	505	5.5513400	99
6	670	0.1931300	96
7	1075	0.0212898	95
8	1388	0.0093784	94
9	1009	0.0070836	88
10	1100	0.0051183	87
11	1379	0.0031194	79
12	1144	0.0021618	85

Table 2: GA results for Belgium data

i	*average # iters*	*minimum residual*	*% min found*
0		0.000582	
1	20	0.000368	100
2	118	0.000282	100
3	229	0.000210	100
4	1172	0.000056	12
5	8414	0.000036	44
6	5389	0.000028	78
7	9352	0.000017	92
8	9793	0.000005	96
9	3354	0.000003	99
10	11545	0.000001	57

Table 3: GA results for Wood data

outliers) we have to run at least 24 tests with different random seeds. Even so, the average number of iterations is quite small compared to the size of the search space.

5.4 Locating the Outliers

Once the outlier sets are determined, we must then apply our heuristics to the residuals for these sets. Table 4 shows this analysis applied to the results of the GA runs for the Belgium dataset. Note that $d = r_{i-1} - r_i$ in this table.

Examine column d/r_0 in Table 4. The numbers decrease dramatically between the 6th and 7th rows. This indicates 6 outliers. In addition, scan the diagonal of this table and notice the spike at row 6, which is another indication of outliers.

For secondary outlier sets, examine column d/r_6 in Table 4. Note the immediate drop in residual differences. Move to column d/r_7. Again notice the sharp drop after the first value. Finally, move to column d/r_8. Notice how the values in this column are relatively flat. These data indicate a possible secondary outlier set of size 2.

Figure 2 shows a composite graph of the Belgium data for columns d/r_0 and the diagonal from Table 4. Note the sharp dropoff on the line for column d/r_0 and the spike on the line for the diagonal. Both occur at 6 outliers. Figure 3 shows a composite graph for columns

i	d/r_0	d/r_1	d/r_2	d/r_3	d/r_4	d/r_5	d/r_6	d/r_7	d/r_8	d/r_9
1	.2078									
2	.1725	.2178								
3	.1543	.1948	.2491							
4	.1507	.1903	.2433	.3241						
5	.1387	.1751	.2240	.2983	.4414					
6	.1695	.2139	.2736	.3643	.5391	.9652				
7	.0054	.0068	.0087	.0116	.0172	.0309	.8897			
8	.0003	.0004	.0006	.0008	.0011	.0021	.0616	.5594		
9	.0000	.0000	.0001	.0001	.0002	.0004	.0118	.1077	.2446	
10	.0002	.0000	.0001	.0001	.0001	.0003	.0101	.0923	.2095	.2774
11	.0003	.0000	.0001	.0001	.0002	.0003	.0103	.0938	.2131	.2821
12	.0000	.0000	.0000	.0000	.0000	.0001	.0049	.0449	.1021	.1351

Table 4: Residual heuristics for the Belgium dataset

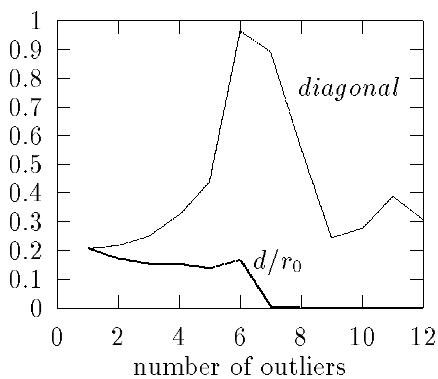

Figure 2: Primary Outliers

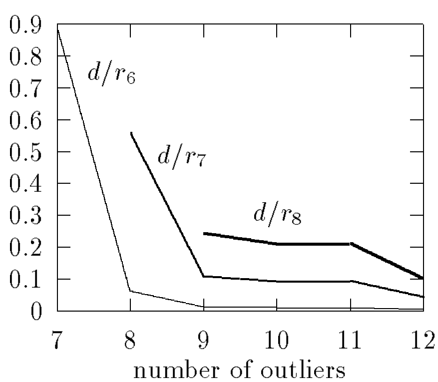

Figure 3: Secondary Outliers

d/r_6, d/r_7, and d/r_8 of Table 4. Compare the sharp dropoffs in the lines corresponding to columns d/r_6 and d/r_7 against the relative flatness of the line for column d/r_8. These visuals make the data easier to interpret.

6 Conclusions and Future Research

In this paper we present a new technique for detecting multiple outliers in regression data using genetic algorithms. Sixteen different datasets with known outliers were used to test our GA implementation. Each dataset contained known outliers, and our GA was successful in finding the outliers in all datasets. The heuristics applied to the residuals of the outliers sets were successful in pinpointing the "true" outlier sets. One limiting factor in our analysis is determining the level of confidence for any particular dataset. More research is required to solve this problem.

Clearly the most significant aspect of this work is that our GA was able to select best sets of outliers from regression data with a significant level of confidence. Because of the enormous size of the solution space, selection of these sets is a problem shared by all outlier diagnostic techniques [12]. To the authors' knowledge, GAs have not been applied to this problem before. The GA proved to be an excellent tool for the selection of these outlier sets.

7 Acknowledgements

This research has been partially supported by OCAST Grant ARO-038. The authors are grateful to Professor Peyton Cook and Eric Ziegel for their comments and suggestions. The authors also wish to acknowledge the support of Sun Microsystems, Inc.

References

[1] A. L. Corcoran and R. L. Wainwright. A genetic algorithm for packing in three dimensions. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pages 1021–1030, 1992.

[2] A. L. Corcoran and R. L. Wainwright. LibGA: A user-friendly workbench for order-based genetic algorithm research. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 111–117, 1993.

[3] K. D. Crawford. Solving the n-queens problem using genetic algorithms. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pages 1039–1047, 1992.

[4] K. D. Crawford, R. L. Wainwright, and D. J. Vasicek. Detecting multiple outliers in multiple dimensions using genetic algorithms. Technical Report UTULSA-MCS-92-4, The University of Tulsa, July 1992.

[5] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann Publishers, 1987.

[6] Lawrence Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, NY, 1991.

[7] Kenneth A. DeJong and William M. Spears. Using genetic algorithms to solve NP-complete problems. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 124–132. Morgan Kaufmann Publishers, Inc, 1989.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[9] W. W. Hager. *Applied Numerical Linear Algebra.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[10] Heinz Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, Inc, 1989.

[11] Gregory J. E. Rawlins, editor. *Foundations of Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[12] P. J. Rousseeuw and A. M. Leroy. *Robust Regression & Outlier Detection.* John Wiley & Sons, New York, NY, 1987.

[13] T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In H. Schwefel and R. Maenner, editors, *Parallel Problem Solving from Nature*, Berlin, Germany, 1991. Springer Verlag.

[14] R. Tanese. Distributed genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, 1989.

[15] Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, 1989.