School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Unsupervised Outlier Detection using Evolutionary Algorithm Techniques

Alex Monckton

Supervisor: Andrew Lensen

AIML501/589

Abstract

Datapoints that deviate strongly from the common trends in the dataset, also known as outliers or anomalies, can appear in many real-world applications through measurement errors and unforeseen circumstances. Coming up with methods to detect them autonomously is a challenging yet promising research area. In this paper we investigate a range of both supervised and unsupervised outlier detection techniques using machine learning. Focusing on unsupervised and evolutionary algorithms, we propose our own GA based algorithm inspired by methods used in previous papers, with a focus on optimising the slow runtime. We are successful in shrinking the runtime to less than half on a larger dataset, but also leave plenty of discussion and room for future improvement.

Table of Contents

1. Introduction

1.1. Problem Statement

A common problem faced in machine learning and data interpretation is the presence of outliers: data points that deviate largely from the common trends in the dataset. Also referred to as anomalies, outliers can occur in datasets through measurement errors or unforeseen circumstances and can cause drastic changes in statistical analysis and machine learning results. Therefore, being able to detect these outliers automatically through outlier or anomaly detection is a valuable research area. There are also many applications that extend from cleaning up a dataset, for example detecting malicious network connections or unexpected behaviour in network systems [1]. Other suspicious activities such as impersonation, unusual surveillance behaviour, and even patients with abnormal symptoms in a medical setting could all be detected as outliers [2].

1.2. Motivations

Particularly for large and high-dimensional datasets, machine learning for outlier detection is a promising technique that can be used to find datapoints that don't belong to any one class, or a class that has too few datapoints to be viable. These definitions split outliers into two types, global outliers which deviate from all other points in the dataset, and local outliers which are far from their closest (either just inliers or both) neighbours. There are two main approaches to outlier detection in machine learning, through supervised learning or unsupervised learning.

In supervised learning, a model is trained on previously experienced labelled data, and therefore information and domain knowledge about the outliers needs to be known beforehand. This makes it unfeasible for detecting new types of anomalies in fraud detection and other related applications.

Therefore, unsupervised outlier detection is an interesting research area as it can be effective at detecting outliers without needing pre-labelled data, something that can be hard to find in real world practices [2]. Instead, outliers are identified by looking at similarities between points in the data, either through clustering-based detection or proximity-based. In both cases, these algorithms require specifying variables such as cluster size or neighbourhood distance. There is no universal best fit algorithm for these unsupervised methods, and they are therefore extremely sensitive to parameter tuning. Research has shown that a poor parameter setting can lead to inferior outlier detection [2], and therefore additional attention on these parameters becomes its own separate optimisation problem.

Some recent works have taken a different approach and looked at using evolutionary computation to evolve an outlier detection model. These machine learning algorithms can learn some representations and parameters themselves without needing to specify beforehand. Evolutionary algorithms such as Genetic Programming can even have the potential to return an interpretable symbolic-based structure to give insight as to why the computer is making certain decisions around the outlier detection. One of the more commonly used evolutionary techniques called Genetic Algorithm (GA) shows potential for outlier detection for its simplicity and ability to be adaptable to many situations. There have been previous attempts at using GA for outlier detection [5, 8, 9], however these investigations were performed across reasonably small datasets, the largest being at 7000 instances with just 4 dimensions [5]. In real world problems, datasets are usually significantly larger in size and dimensionality and therefore it is important that an outlier detection method is scalable and optimised.

1.3. Research Goals

Outliers are usually defined as data that deviates strongly from all other points (including other outliers), however we decided to investigate an outlier detection method that only looked at deviation from inliers. In some cases of anomaly detection, it might be possible that the anomalies themselves are not too different from each other, made of a smaller sized class potentially even clustered together. The key difference with our proposed algorithm when compared to previous methods discussed in the paper, is that our distance function for each outlier only looks at its closest non-outliers, ignoring proximity with other outliers.

In this project we first investigate a range of outlier detection techniques and propose our own unsupervised GA based algorithm inspired by methods used in previous papers, focusing on optimising the training runtime for use on larger sized datasets. The goal is to investigate this different proximity evaluation, and attempt to maintain high precision and recall as much as possible whilst optimising the computation required. We first experiment and perform tests on an initial algorithm, mostly inspired by a previous method but with modified fitness evaluation, and then attempt to optimise the GA further for a second version.

2. Background Concepts

2.1. Evolutionary Computation

Evolutionary Computation (EC) is a subfield of Artificial Intelligence inspired by real world biological processes. Evolutionary algorithms such as Genetic Algorithm (GA) or Genetic Programming (GP) evolve a group of individuals over a series of generations through minor mutations and genetic cross-over, selecting the best through a fitness measure, emulating natural selection (**Figure 1**). These mutations and the genes which are crossed over are usually chosen by random chance, resulting in a diverse range of possible individuals able to be generated and explored, whilst maintaining elitism by saving the best performers over each generation. Over time the population of individuals will evolve to optimise the desired result, which is evaluated through the fitness function, as the individuals are ranked and selected for the next generation based on this score. This can either be done deterministically or stochastically: either only the best performing individuals are selected, or any individual can be picked at random with better fitness meaning a higher chance to be chosen.

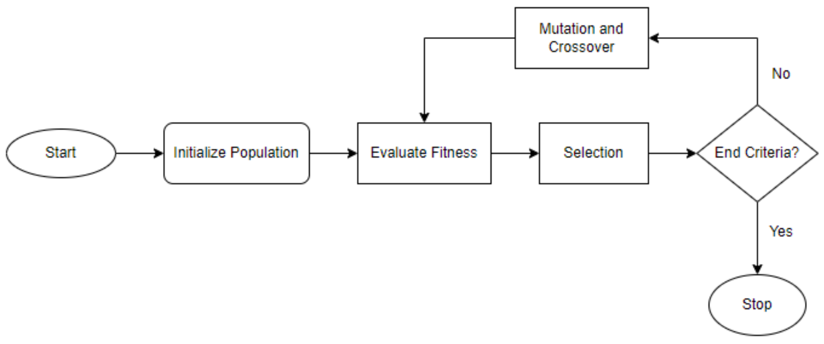

Figure 1: Evolutionary Algorithm Flowchart

2.2. Genetic Algorithm

In the GA, individuals are represented by a series of 1s and 0s called a chromosome, with each 1 or 0 being a gene. This format makes this algorithm useful for problems where the goal is to find the optimal selection from a group of features or variables, with a 1 representing a feature being included and a 0 being excluded. The initial population consists of a group of randomly generated individuals, which are then ranked by taking the genes from each chromosome and applying the fitness function on that selection of the input data. After ranking, the best individuals are selected, and these are assigned as the parents for the next generation. A gene cross-over algorithm is applied which involves taking two parents, and randomly selecting half of the genes from one and the other half from the other to create a new individual, called the offspring. This maintains the positions and values of the genes in the already proven well performing parents, only generating different combinations of already existing chromosomes. A mutation step is also applied to either these new offspring or to other existing parents in the population. This would involve flipping one or more chosen genes in the individual, either changing a 0 to a 1 or a 1 to a 0. This could be done randomly or with a local search to optimise the mutation, such as in a memetic algorithm. The individuals generated through the gene cross-over and mutation stages are often combined with the selected parents for the next generation to maintain elitism. Elitism means the current best individuals are never lost in each generation and kept through to the end of the training. To maintain the same number of individuals in every generation, the number of individuals generated through cross-over and mutation should equal the population size minus the number of parents generated.

2.3. Genetic Programming

GP follows a similar process to GA; however, the individual representation is very different. In GP every individual is a program complete with functions and terminals, usually represented by a tree diagram. Individuals are randomly generated by picking from a set of pre-defined functions (i.e. addition, subtraction, multiplication, division), and terminals (i.e. set numbers or variables), to create an arbitrary function that can then be modified to evolve over time. Fitness evaluation can be done when a set of training data is passed through each individual's program and the output is compared with its known desired values through a similarity metric such as mean square error. The programs which have the closest overall predicted outputs to the desired outputs will have the lowest mean square error, and therefore have the lowest fitness value, which makes this type of problem a fitness minimisation problem. Crossover in GP usually involves taking subtrees of one individual's program tree structure and either switching it or overwriting a subtree in another individual's program. Mutation can have many forms, either simple functions or terminal nodes (such as numbers or variables) can be randomly switched with another compatible node of a different type, or nodes could even be added or removed from the program entirely. The rest of the process is much the same as GA, however the more complex and customisable individual representation in GP makes it suitable for a larger range of different applications.

2.4. Mean Square Error

A common metric used in machine learning for evaluating the difference between two sets of continuous data is Mean Square Error (MSE). It can be useful as a fitness function in supervised learning when the results passed through a learned algorithm can then be compared with its corresponding test data. In MSE the difference between each pair of points in the data is calculated and squared, these are then summed up and divided by the total number of points (**Equation 1**).

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Equation 2: Mean Square Error Formula

where Y is the true observed value, $\hat{Y}$ is the corresponding predicted value, and n is the number of samples in the dataset.

2.5. Precision and Recall

Precision and recall are two evaluation metrics useful for classification problems especially when there is an imbalance in size between the classes, as overall accuracy would be biased towards the larger class.

Precision is a measure of how accurate the predictions of one class are, which in this case would be the accuracy of the returned outliers. It is calculated as the number of relevant individuals in the returned instances, divided by the total number of *returned* instances. It can also be seen as a sort of measure of the *quality* of the predicted outlier data.

On the other hand, Recall is a measure of how many of the correct class instances were returned as predictions. It is calculated as the number of relevant individuals in the returned instances, divided by the total number of *relevant* instances. In this case it can be seen as more of a measure of the *quantity* of the predicted outlier data.

2.6. Multi-objective Optimisation

Traditionally when using evolutionary algorithms there is a single objective or fitness function that needs to be optimised. However, sometimes a problem might involve needing to optimise two or more conflicting objectives, for example minimising house price whilst maximising property size. This cannot be evaluated with a single fitness value and therefore requires a special fitness ranking system. Because there is no single best solution that exists, a common technique is to find the best set of individuals that fall closest to the optimal Pareto front. The Pareto front is a curve in multi-dimensional space which contains all individuals which cannot be dominated by any other individual in the population, i.e. can't be beaten on any one objective without being worse on another objective. The goal is to find a non-dominated curve with a variety of individuals evenly spaced across the curve to return a diverse range of quality solutions across the different objectives. This also gives the user freedom to choose the solution they'd prefer depending on which objective(s) they deem most important.

2.7. NSGA-II

One commonly used fitness ranking system for multi-objective optimisation is the Nondominated Sorting Genetic Algorithm II [3] (NSGA-II). The previous naïve approaches for MOO involved comparing every individual with every other individual to find which were nondominated for every rank, and this had an overall worst-case complexity of $O(MN^3)$ where N is the population size and M is the number of objectives. The NSGA-II is a faster algorithm with a complexity of only $O(MN^2)$, as for each individual a domination count *np* (number of solutions that dominate it), and a dominate set *Sp* (solutions it dominates) are calculated. The entire population can then be ranked by first going through each set *Sp* of the nondominated individuals with *np* = 0 and reducing its domination count by 1, any individuals that then have a *np* = 0 after this step can be ranked under the second curve, with this process repeating for the remaining ranks. Despite being faster, NSGA-II does have a higher storage requirement over the naïve approach as there are now $O(N^2)$ values instead of O(N) before to store, but the saving in speed is worth the trade-off. Individuals can then be sorted within each rank by their crowding distance with their neighbours, to maintain diversity and encourage a uniform distribution across the curve. The crowding distance is calculated by taking the absolute distance between the two neighbours either side, and giving the two individuals at each end of the curve an infinite distance. Therefore, all individuals in a population can be ranked for selection in an evolutionary algorithm by first going through each Pareto curve rank from 1 and up, until a rank is reached that contains more individuals than remaining slots for the selection. The individuals in this rank are then sorted by crowding distance and added to the selection from highest to lowest.

3. Literature Review

3.1. Using GP For an Ensemble of Outlier Detection Algorithms

There are many different types of outlier detection algorithms, some are designed for detecting local outliers (i.e. Local Outlier Factor (LOF), Local Outlier Probability (LoOP)), while others are useful for obtaining global outliers (Support Vector Machine (SVM), X-means). In some outlier detection problems, it is not always clear whether a global or local outlier detection system should be used, and combining the two isn't always straightforward.

In 2019, Folino [4] proposed an ensemble outlier detection method that utilised GP to combine various algorithms with different parameters to detect new network attacks. A GP function set consisted of the standard Avg, Max, Product and Square functions, as well as a selection of both local and global outlier detection algorithms, with and without Principal Component Analysis (PCA) dimensionality reduction and a variety of parameter settings. The investigation was done using a supervised approach with predetermined validation data, so the fitness function was simply the classification accuracy. The dataset that was performed on was the well-known NSL-KDD dataset, which contained 6 different attacks (ipsweep, nmap, portsweep, satan, r2l, and u2r) as output anomaly classes.

Through the investigation it was found that global outlier techniques performed well on some attacks (notably the ip-sweep, nmap and satan), and local outlier techniques performed better on others (satan, r2l, u2r). An ensemble was then built to combine both the local outlier and global outlier algorithms to find a function that could perform well on both, and the results showed that for most attacks there was a considerable increase in accuracy when only 5% of the data was used as validation, with the remaining 2 attacks improving at 20% validation data. The results showed good accuracy, performing around 70-80%, and made use of Genetic Programming to find an optimal aggregation of outlier detection techniques. However, the main challenge with their method is it relies strongly on quality validation data that is necessary to train the GP model. In real world applications this validation data may not be easily accessible, which is one of the advantages of an unsupervised approach.

3.2. One Class Classification with KDE and GP

In the field of network security, the access to quality data poses a great challenge for supervised learning. As the widespread use of computer networks continues to increase, new types of anomalies are evolving making existing data potentially redundant for capturing new attacks. Additionally, information about existing anomalies might be difficult to obtain due to security and privacy reasons, and there would be a very small sample size of these intrusive events. Therefore, as only the normal network behaviour data is usually available, an unsupervised technique called "one class classification" has been used by Cao, Nicolau, and McDermott [1] in 2016 where they proposed a new method for network anomaly detection.

Kernel Density Estimation (KDE) is a way of modelling the density of a given dataset and can be used for outlier detection by assigning a density threshold, classifying data between outlier and inlier depending on whether its density meets the value. However, the computational cost at query time can be high, as it scales with the number of data points [1]. KDE stores all the training data before computing the density of a new datapoint, increasing the cost as the training data gets larger. Therefore, the proposed method is to model a density estimation through GP, as this query computational cost would only be as large as the number of tree nodes. This comes with a second advantage, returning a potentially interpretable expression for classifying data.

First the KDE is used to estimate the density of the training data, however an issue with this is that regions outside of the training data won't be able to have their density predicted. Therefore, some artificial data is generated by sampling the low-density regions from a gaussian distribution with same mean and standard deviation as the training data. This is combined with the training data for evaluating the fitness function of the GP, which is the root mean square error between the KDE density and the GP predicted density (figure below). RMSE is the same as MSE except for an additional square root applied on the result (**Equation 2**).

$$Fitness = \sqrt{\frac{(\sum_{i=1}^{m+p}(f(x_i) - d(x_i))^2)}{(m+p)}}$$

Equation 2: RMSE Fitness Function

where m is the number of samples in the training set and p is the number of samples in the artificially generated set. f is the GP algorithm and d is the KDE density.

Five datasets were used in their experiments, and each was randomly sampled at a 50/50 split between training and test data. As this was a one-class classification investigation, datasets with only one dominant target class with a small number of non-target classes were used, and all non-target instances were used in the testing set only. Experiments were conducted on the proposed One-Class Genetic Programming (OCGP), as well as on One-Class Kernel Density Estimation (OCKDE) and One-Class Support Vector Machine (OCSVM) for comparison. Results were measured using the Area Under Curve (AUC) metric, which is a summary of how well the model can distinguish between the positive and negative class across multiple density threshold values. Some computational cost measurements were also taken. For the OCGP, the number of nodes in the tree, for the OCSVM, the number of support vectors, and for the OCKDE, the number of training points.

It was found that the AUC performance across the five datasets and all three models were very similar, with the OCGP often even performing higher accuracy than the OCSVM. The number of training points and support vectors, however, scaled largely with the size of the dataset, whereas for the OCGP the number of nodes in the tree would stay consistent at around 200. This suggests that the OCGP is the most computationally efficient at query time, particularly for large datasets, whilst maintaining a similar accuracy to other methods. The training time, however, for the OCGP model is much slower than the other two methods, taking around 10 min compared to only a few seconds for OCKDE and OCSVM, but it has solved the main disadvantage of KDE, which is its slow query time, making it the most suitable for big datasets in real world applications.

3.3. Outlier Detection with Genetic K-Means

The previous article was an example of a statistical method for unsupervised outlier detection, in which a prior knowledge of the distribution of the dataset is used, however there are two other methods [5]. In deviation-based or spatial outlier detection, outliers are identified as the objects which cause the most change when excluded from a dissimilarity function (for example the variance across the dataset). The third and most widely used method is distance-based outlier detection, where outliers are identified through calculating the distance to their nearest neighbours. This method is so popular because it can be used on any feature space where a distance measure can be defined [5] and doesn't require any prior statistical distribution. However, its main disadvantage is the large computational cost involved when calculating nearest distances, which scales enormously with large datasets.

Many researchers have proposed techniques to deal with this issue, one unsupervised outlier detection technique is to make use of clustering, a type of algorithm that is designed to group clumps of similar data together called clusters. These clustering algorithms can also be used to find datapoints that don't fit in to any one group, labelling them as outliers. To avoid the computationally expensive distance calculation, a GA can be combined with K-Means clustering in an algorithm called Improved Genetic K-Means (IGK), which was used by Marghny and Taloba [6] in their proposed outlier detection. In IGK [7], the individual representation is a string of integers with length number of points in the dataset, with each integer representing the cluster each point is assigned to. The initial population is generated randomly using integers up to a specified K, and the fitness function is calculated as the sum of the distances from each point to its cluster's centroid (**Equation 3**), with the goal being to minimise it.

$$Fitness = \sum_{j=1}^{\acute{k}} \sum_{X_i \in C_j} \left| X_i - Z_j \right|^2$$

Equation 3: IGK Fitness Function

where k represents the number of clusters, C_j represents the *j*th cluster and Z_j represents the centroid.

Mutations is done probabilistically where the closer a point is to a new centroid the more likely for it to change to that cluster. An additional step is used to enhance the IGK where the two cluster centroids that are closest to each other merge into the same cluster, and the process is repeated until only k clusters remain. The outlier detection step is then applied to the resulting clusters, datapoints are ranked based on an Outlyingness Factor [6], which is dependent on the distance to its cluster centroid. These outlyingness factors are normalised between 0 and 1, and then a threshold can be assigned to classify outliers that have an outlyingness higher than the threshold. This threshold value used in the paper was set at 0.9, and the algorithm was run on three different synthetic datasets. Because of the way the datasets were generated the original centroids are known, and so to evaluate performance the mean square error was taken between the predicted centroids and the actual. These were compared to a standard K-means clustering algorithm and an Outlier Removal Clustering (which is like the proposed method except uses K-means instead of IGK).

There was a clear improvement with the proposed method, which consistently had a much lower mean square error on the cluster centroids. The datasets generated for the experiment were made up of normally distributed round clusters, and so the MSE measure is a fair judgement of the cluster and therefore outliers' quality, however this is only feasible because of the specific shape of the dataset. Some datasets might have complex shapes with clusters that K-means isn't able to detect well enough, where this method and this evaluation would be unsuitable. The bonus that comes with an outlier detection method like this is the rest of the data is classified into clusters, which might be useful depending on the problem. However, the quality of the outliers will lie entirely on the quality of the clusters found by the clustering algorithm.

3.4. Density-based Evolutionary Outlier Detection

Another technique to avoid the computationally expensive closest neighbour calculation is proposed by Banerjee [5] who declared an approach that used a density-based distance measure combined with evolutionary search algorithm. In this method, outliers are detected through a weighted distance measure that combines simple Euclidean distance with a density component. This component is related to how frequently the feature, or in the case of continuous data, bin occurs in the dataset.

The individual representation in the evolutionary algorithm is a string of integers the same length as the number of points in the dataset. Each integer represents the predicted closest neighbour based on this modified distance metric, so for example in a dataset of five samples, an individual [43513] would mean point 4 is the closest to point 1, point 3 is the closest to point 2 etc. During genetic crossover, distances are compared iteratively through both parents to select only the shortest from each one rather than random chance. Mutations are still performed randomly from a predetermined probability; however, a second mandatory mutation correction is also applied to ensure no genes refer to themselves as their own nearest point. In some experiments this random mutation step was disabled after a set number of generations to encourage exploitability. The fitness function was calculated inversely to the sum of the nearest distances, making this a minimisation problem with a fitness less than 1 near optimal. Outliers can then be selected from the final representation by taking the k points with the largest distances that occur in the final individual.

A first series of experiments was conducted on a set of five baseline datasets for parameter tuning, and results were compared to a Least-Squares genetic algorithm outlier detection technique [8]. A technique for population initialisation that involved using k-means clustering and assigning genes to others within the same cluster was tested and resulted in convergence being achieved up to two times as fast. The second set of experiments involved validating the parameters across two gaussian generated datasets of size 1600 (and dimensions 2 and 12). Finally, two real world datasets (yeast dataset of size 1484 and dimensionality 6, highway dataset of size 7089 with 4 dimensions) were tested. In both datasets, datapoints that were either unrepresentative or errors were able to be detected for low values of k, and the results were encouraging.

The computational issue with having to iterate through all datapoints to calculate the nearest neighbours in a large dataset has been significantly reduced with this method. The additional density weight also helps outliers in sparse regions be detected faster. However, as the individual representation is the length of the entire dataset, this still makes this method a challenge for very large datasets. A more variable chromosome design that only stores information for outlier distances and can change length during the algorithm could be more efficient in these cases. There is also still the challenge of choosing the best value for k, which in future work could be automatically identified through some other fitness measure.

3.5. Outlier Detection with Multi-objective GA

A similar genetic algorithm approach to unsupervised distance-based outlier detection was proposed by Duraj and Chomatek in 2017 [9]. Individuals were again represented as a string of integers of same length as the size of the dataset, however instead of storing the nearest point in each gene, simply a 1 or 0 was used to determine whether a point should be classified as outlier or not.

This algorithm made use of multi-objective optimisation, instead of having a single fitness function to optimise there were five different objectives. Firstly, the average distance from the k nearest neighbours of the outliers was aimed to be maximised, and there were three functions here for k values 1, 2, and 3. This had the goal of keeping outliers as far from other points in the data as possible. The fourth objective was to maximise the average distance of all the outliers from the centroid of the outliers, with the goal to keep outliers as far from each other as possible. The final objective was to minimise the number of outliers, as datasets generally have far fewer outliers than inliers.

The crossover was simply done by selecting a random point along the chromosome and switching half with another parent, and mutation was a small random chance to flip a gene from 1 to 0 or 0 to 1 on every datapoint. Because there were five objectives, a special GA fitness evaluation system needed to be used for ranking individuals. Three different genetic algorithms were tested in the paper; PESA-2, SPEA-2, and NSGA-II.

Two experiments were conducted using the first 100 observations of the Wisconsin Breast Cancer dataset from the UCI Machine Learning Repository, of which 18 were malignant cases and used as outliers. In the first experiment only the first 4 attributes were used, but in the second all 10 were used. Four metrics were used for evaluating the performance of the algorithm; percent of true outliers returned, accuracy of predicted outliers, percent of inliers identified as outliers, and the accuracy of incorrectly predicted outliers. Through these results it was concluded that NSGA-II performed the best in terms of identifying the most outliers, however it also misidentified the most too. Both PESA-2 and SPEA-2 identified less but were also the most accurate in their predictions. All algorithms performed better on the lower feature space, which is likely due to there being a smaller search area. One main disadvantage to this method is the expensive distance calculation that needs to be performed three times across different objectives with a progressively higher k value. The dataset used in the experiments was very small (only 100 samples), and this is likely because the algorithm doesn't scale well up to larger datasets.

3.6. Summary

Supervised methods are extremely reliant on quality training data, something often hard to find or not accessible in outlier detection applications. Most of the reviewed methods for outlier detection also involved manual parameter tuning, something which these algorithms are extremely sensitive to. The multi-objective approach by Duraj and Chomatek [9] has the most potential for a completely unsupervised autonomous algorithm, and was therefore the method we were most inspired to further develop.

4. Research Project

Our first outlier detection method involved taking a GA with the same individual representation as the proposed experiments done by Duraj and Chomatek [9], where each individual was a string of 0s and 1s same length as the number of samples in the dataset, where a 1 meant a sample was labelled as an outlier and 0 meant it was an inlier (**Figure 2**).

[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0]

Figure 2: Example Individual Representation

We chose this representation as it is simple and not too memory intensive, only storing one of two integers for each datapoint. We also used the NSGA-II ranking system which in the results [9] found to yield the highest accuracy. Unlike the five objectives that they used, we wanted a simpler algorithm that used fewer objectives for better performance and easier results visibility. In our first experiments we tried using the objective of maximising the average distance from each outlier to its closest non-outlier neighbour, similar to the first objective used by Duraj and Chomatek [9] but ignoring other outliers. However, because we were minimising the average distance, the genetic algorithm found that for this objective the optimal result was to have only one outlier: the outlier furthest from its closest neighbour, and therefore the highest average across all (one) points. This meant to return more outliers we needed a conflicting objective, so for our second objective we tried maximising the number of outliers. This returned many outliers across the approximate Pareto curve, even towards the lower end of the number of outliers objective, so we knew this set of objectives wouldn't be suitable. In a dataset the number of outliers is usually minimal and therefore we knew the first distance objective needed to conflict this so the number of outliers can be minimised instead.

4.1. Proposed Genetic Algorithm

To maximise nearest distances whilst also maintaining many outliers, we tried a slightly different objective function of taking the total distance of all outliers to their closest neighbours, rather than the average. This now meant the best possible selection was to include every point in the dataset as an outlier, so by minimising the number of outliers as our second objective (**Equation 4**) we were now able to generate a diverse range of solutions across the Pareto front with different numbers of outliers. To keep the number of outliers from getting too large for both performance issues and keeping it at a low reasonable level, we assigned a max number of outliers threshold to 10% of the size of the database. This required some domain knowledge as we knew there were fewer than 10% outliers in all three datasets we tested on, however this is a good estimate for outlier detection as datasets generally only have a very small number of outliers in them. The smaller the threshold, the higher quality the results across the Pareto curve are and therefore the higher the chance of automatically selecting a good individual, but too small a threshold and too much information will be lost from the fitness function cut-off. In our fitness calculation, if there were more outliers than the threshold, the distance fitness would simply be assigned to 0 (**Equation 5**).

$$Num\ Fitness = \frac{n}{\text{len}(X)}$$

$$Dist\ Fitness = \begin{cases} 0 & \text{if } n > t \\ \sum_{i \in X}^{n} \min_{d} d(P_i, P_{j \in Z}) & \text{if } n \leq t \end{cases}$$

Equation 4: Number of Outliers Fitness Function ***Equation 5: Total Distance Fitness Function***

where *n* is the number of outliers in the individual *X*, *t* is the maximum number of outliers threshold, P_i represents the value of outlier *i*, P_j represents the value of *j* in the non-outlier set *Z*, and *d* is the distance between them.

We ran this algorithm across three different datasets with different dimensionality and numbers of samples, a population size of 100, cross-over rate of 80% and mutation rate of 20%, and for 1000 generations (except for the first dataset which only needed 500). For the initial population representation, we generated a population of individuals with entirely 0s to encourage a start with no outliers and to add them slowly through mutation to overall encourage a small number of outliers. The mutation function was a simple bit flip which we assigned a 5% chance to randomly flip every gene. We kept the mutation rate this low to introduce new outliers into the individuals slowly and prevent too many from reaching over the 10% threshold. The crossover function was a one-point crossover, where a random midpoint is chosen and two individuals swap half of their genes with each other up to the midpoint, generating two new offspring.

On each dataset, the algorithm was run five times across five different seeds, where fitness, precision, and recall were stored and an average of each was calculated for evaluation.

Lymphography Dataset

We first ran on the lymphography dataset from the UCI machine learning repository [10], which had 148 records and a dimensionality of 18. It was a multi-class dataset that contained four classes however two of those classes have very few samples (2 and 4), so these are combined into a set of 6 outliers, with the rest of the data classed as inliers. This dataset was ideal for our investigations, as there was potential for some outliers to be within close proximity with each other within one of the small classes, but also separate between the two for a variety of outlier types.

The runtime was fast at only around 40 seconds, and the final Pareto curve contained 14 individuals (**Figure 4**). From this set we wrote a script to find the best individuals by precision and recall. The individual with the best precision predicted 2 outliers, and both were correctly identified for a 1.0 precision and 0.33 recall. The best individual by recall predicted 12 outliers, of which all 6 were correctly identified for a 0.5 precision and 1.0 recall.
To get an overall selected best from the curve, we normalised both fitness values across the Pareto curve between 0 and 1 and found the closest individual to the optimal fitness of [1, 0] (maximum distance, minimum number of outliers). On the first run this chose an individual that predicted 6 outliers, of which 4 were correct for a 0.67 precision and 0.67 recall. We then repeated the algorithm across 5 different seeds, and saved the fitnesses, precision, and recall into the table below (**Figure 3**).

	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Num Outliers Fitness	0.0405	0.0405	0.0405	0.0405	0.0405	0.0405
Distance Fitness	10.893	10.829	10.902	10.884	10.902	10.882
Precision	0.67	0.67	0.67	0.67	0.67	0.67
Recall	0.67	0.67	0.67	0.67	0.67	0.67

Figure 3: Lymphography Dataset Results Table

We used UMAP, a dimensionality reduction algorithm, to visualize the multi-dimensional dataset in two dimensions. This enabled us to make a scatter plot to compare the predicted outliers with the true outliers in the dataset visually on one of the runs (**Figure 5**).

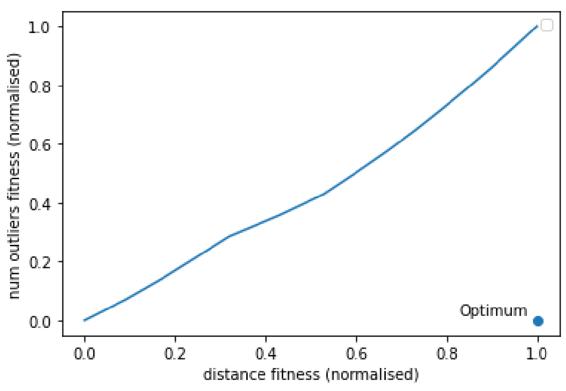

Figure 4: Lymphography Dataset Pareto Curve

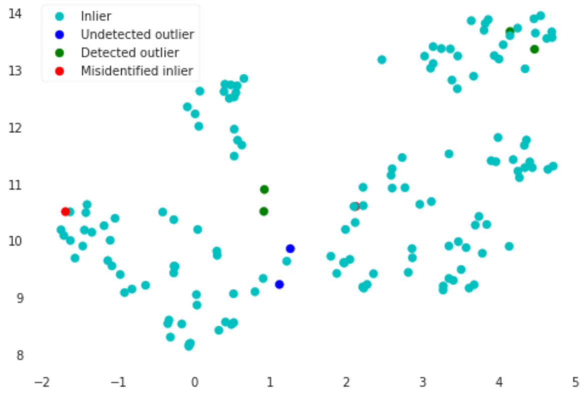

Figure 5: Lymphography Dataset Prediction Plot

The results on this dataset were consistent, always returning the same number of predicted outliers (6), but simultaneously only correct on 4 of them. There was slight variation in the distance fitness across the five runs also, with only runs 3 and 5 having the same. This means each run was choosing a slightly different selection of outliers, however the 4 outliers correctly identified was always consistent. The remaining two outliers proved to be a challenge to detect, however the inconsistency of the two misidentified inliers shows the algorithm wasn't converging on anything specific.

Wisconsin-Breast Cancer Dataset

We ran then ran the same algorithm on the Wisconsin-Breast Cancer Dataset [11], also from the UCI machine learning repository. This dataset was larger, having 278 data points and 30 dimensions. Because the dataset was larger and the individual representation was almost twice as long, we ran the algorithm for 1000 generations instead of 500 (which when tested on the previous dataset had no change in results). The malignant cases in the two-class dataset were sampled down to 29 datapoints, with benign breast cancer cases being the inliers. This was to make the purpose of the outlier detection to be detecting malignant cases of breast cancer when amongst a dataset of benign cases.

This time the algorithm took around 2 minutes to train, and the Pareto curve was larger with around 36 records (**Figure 6**). The shape of the curve was also interesting, having a noticeable flat section around the middle part. This meant there was a small region of around 0.2 to 0.4 number of outliers fitness where the algorithm struggled to find individuals with high distance fitness, which when looking at the scatter plot of the data (**Figure 7**) could be because once all the distant outliers were detected the rest of the data was too close together to offer much improvement. From this curve we found the best individual by precision predicted 13 outliers of which all 13 were correctly identified for a 1.0 precision and 0.62 recall. The individual with best recall predicted 23 outliers of which 15 were correctly identified for a 0.65 precision and 0.71 recall.

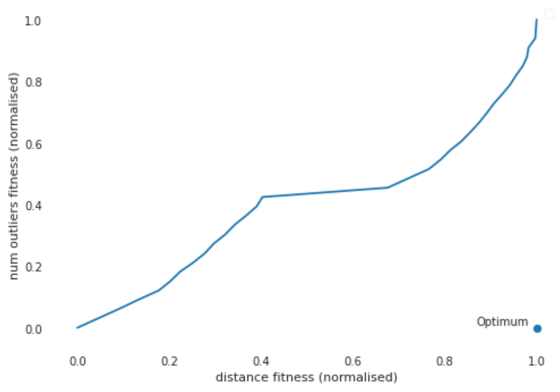

Figure 6: WBC Dataset Pareto Curve

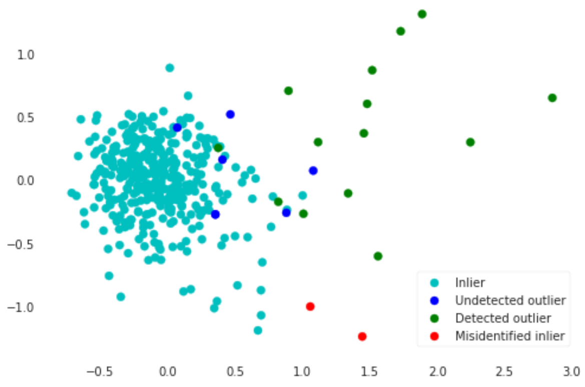

Figure 7: WBC Dataset Prediction Plot

We once again auto selected the best individual by distance from the optimal [1, 0], and repeated across 5 seeds. Results on this dataset as shown in **Figure 8** performed significantly better on the precision, with the best run (Run 2) returning an individual with 0.875 precision, whilst also having the best recall at 0.67 (plotted on **Figure 7**). The average recall however was 0.1 less than the lymphography dataset at 0.57. This lower overall recall is likely because of the larger dataset size with more outliers to detect, which despite running for twice as many generations still has trouble assigning the right bits into outliers when there is such a long individual representation and a low mutation rate. Even when some outliers are correctly assigned and found in an individual, there might be too many outliers over the threshold, assigning the distance fitness value to 0 and losing this potentially useful information.

This dataset was also used in the multi-objective genetic algorithm investigations done by Duraj and Chomatek [9]. However, in their investigations they only used 100 rows of the dataset with 18 malignant cases as outliers. It wasn't clear exactly which data points they used so we weren't able to perform an investigation with a completely accurate comparison, although from the results we have we can see our algorithm did achieve a higher average precision than all but two results. Average recall was not particularly high either, only performing better than the PESA2, although our best run was only beaten by two NSGA-II recall results and had the best precision. Overall, our algorithm was more accurate in the points it classed as outliers; however, it isn't possible to confidently say one method is objectively better without an identical set of data to compare on.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Num Outliers Fitness	0.0397	0.0423	0.0423	0.0423	0.037	0.0407
Distance Fitness	18.52	18.99	15.24	19.66	16.31	17.74
Precision	0.87	0.875	0.56	0.75	0.86	0.783
Recall	0.62	0.67	0.43	0.57	0.57	0.572

Figure 8: WBC Dataset Results Table

Satimage Dataset

There was one more dataset we tested running the algorithm on, and this one was much larger in size. Also from the UCI machine learning repository, the Satimage dataset [12] contained pixel information from satellite imagery classed into 36 different categories, with class 2 down sampled into the outlier class and grouping the rest as inliers. The dataset had 5803 datapoints with dimensionality 36, and there were 71 outliers.

This algorithm took significantly longer to run, taking 2 hours to train. The returned Pareto curve (**Figure 9**) contained 249 individuals, and the best precision was one that returned 42 outliers, all of them correct for a 1.0 precision and 0.59 recall. However, the automatically selected best based on distance to optimum had only 0.19 precision and 0.67 recall, plotted in **Figure 10**. The Pareto curve was returning too many individuals with little improvement over each other, and this made it difficult to automatically select the best performer. The number of outliers in relation to the dataset size was also significantly smaller than the previous two datasets. The lymphography and WBC had 4.1% and 5.6% of their datasets as outliers respectively, whereas in the Satimage only 1.2% of the dataset were outliers. Due to the large Pareto front size and lower ideal number of outliers fitness, we knew we would need to adjust the optimal position for the auto selected best individual.

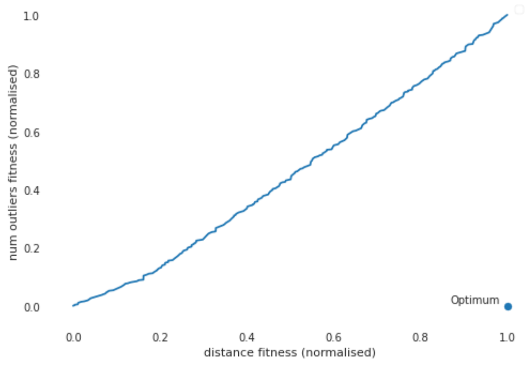

Figure 9: Satimage Dataset Pareto Curve

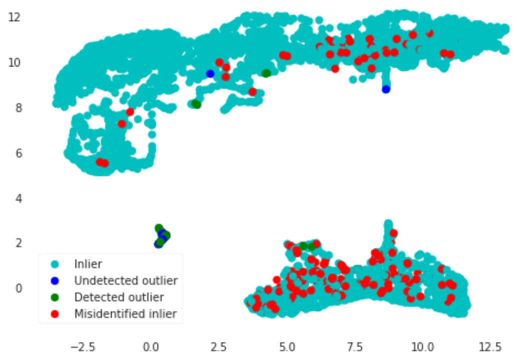

Figure 10: Satimage Dataset Prediction Plot

At 2 hours runtime, we wanted to see if we could optimise the algorithm further before testing and saving results across 5 seeds, and simultaneously see if we could improve the precision and recall of the returned results.

4.2. Version 2.0 Genetic Algorithm

The slowest part of the algorithm is the distance measure in the fitness evaluation, which scales tremendously with the size of the dataset, in both number of instances and dimensionality. The program must find the closest point from all other non-outliers in the dataset and do this for every single outlier adding up the score as the fitness value. This fitness evaluation is applied to every individual at every generation in the algorithm, whether it has had mutation or cross-over applied, and this was slowing it down.

One method we investigated for optimising the distance calculation was to pre-compute every datapoint's closed neighbour distance and store it in an array before training. This could then be simply referenced during the fitness evaluation. However, because we were only looking at the closest distance to a non-outlier, and the datapoint labels were constantly being switched during the training, this distance was dynamic and would require a large matrix for every possible outlier combination.

Therefore, for our improved genetic algorithm, we decided to change the individual representation in such a way as to reduce the number of closest neighbour calculations required. Instead of being represented as a simple string of 1s and 0s, we changed it to store the closest neighbour distances. This now meant each gene was a float value rather than a simple integer, we kept the same idea that a 0 would mean a non-outlier however outliers were now represented by the distance to their nearest non-outlier instead of a 1 (**Figure 11**).

[0.0, 1.12434, 0.0, 0.0, 7.34675, 0.0, 0.0, 0.0, 4.76607]

Figure 11: Example of New Individual Representation

This made the fitness calculation for total distance fitness much simpler, only needing to sum up all the values in an individual (**Equation 6**).

$$Dist\ Fitness = \begin{cases} 0 & \text{if } n > t \\ \sum_{i \in X}^{n} X_i & \text{if } n \leq t \end{cases}$$

Equation 6: New Total Distance Fitness Function

Now that the distance measure wasn't being calculated in the fitness function, and was part of the representation, we had to make it part of the mutation function. We kept the applied mutation and parameters the same, having a 5% chance to flip every gene in the chromosome, however instead of simply switching to a 1 when becoming an outlier, we needed to calculate the closest distance. Because we were only calculating the distance to the nearest non-outlier, and it was possible that points further down the individual might change causing a different result, we replaced all outlier distances with a -1 until the mutation was complete. Once we had the new individual, we could then loop through all the -1s and change them to their closest distance values.

Another step we added to the mutation was a function that reduced the number of outliers to beneath the threshold. The purpose of this was to try and include as many potentially useful individuals as possible without them being assigned 0 fitness and cut from the next generation. We initially tried iterating through the outliers removing them based on their closest distance value from smallest to largest, as the larger the value the further away the outlier. While this worked in most cases on datasets with spread apart outliers, we found this would hinder results when clumps of outliers close together are present, as it would only take one point labelled as a non-outlier for them all to be misidentified. Therefore, we simply selected random outliers in the individual to be converted to inliers, repeatedly until the count was less than the max outliers threshold. This did mean it was possible that good outliers could still be lost from the individual if randomly chosen, but overall, it kept as much of the original information as possible whilst keeping below the fitness threshold.

The crossover function we left the same, selecting a random point to switch two individuals around to generate two new offspring. However, on each of the two offspring we added in the same reduce outliers step to keep them below the threshold. If the genetic crossover resulted in one individual containing too many outliers, they would be randomly switched back into non-outliers iteratively until the number of outliers was low enough.
We then ran the new algorithm across the same three datasets with the same parameters as before.

Lymphography Dataset

On the lymphography dataset, the runtime was similar to the previous algorithm at 49 seconds, only 9 seconds slower, and the Pareto curve was the same size with 14 individuals (**Figure 13**). The initial returned results however were a big improvement, the best individual by precision predicted 4 individuals with all of them correct for a 1.0 precision and 0.67 recall. The best individual by recall got 5 of its 7 outliers correct, for a precision of 0.71 and recall 0.83. The automatically selected best by distance to [1.0, 0.0] was the same as before however, at 0.67 precision, 0.67 recall. We once again repeated across 5 different seeds and saved the results to a table (**Figure 12**).

	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Num Outliers Fitness	0.0405	0.0473	0.0405	0.0405	0.0405	0.0419
Distance Fitness	11.31	12.67	11.17	11.13	10.798	11.416
Precision	**0.67**	**0.57**	**0.67**	**0.83**	**0.67**	**0.682**
Recall	**0.67**	**0.67**	**0.67**	**0.83**	**0.67**	**0.702**

Figure 12: Lymphography Dataset Results Table (Version 2.0 Algorithm)

Overall, there was a small improvement in both average precision and recall, an increase of 0.012 and 0.032 respectively. The results this time had more of a variety, with one individual performing exceptionally well at 0.83 precision and 0.83 recall (Run 4), plotted in **Figure 14**. It only had one misidentified inlier, and one undetected outlier. Once again, the number of predicted outliers was always consistent at predicting 6 outliers, with the exception of one run which predicted 7 for a lower precision at 0.57 (Run 2).

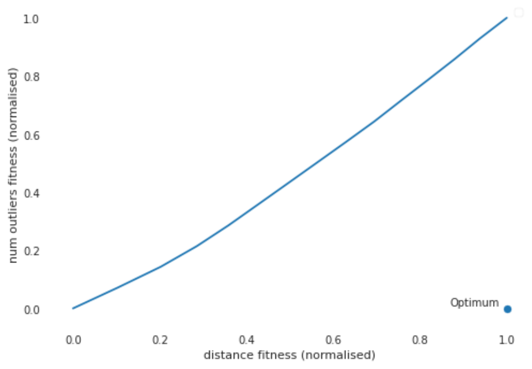

Figure 13: Lymphography Dataset Pareto Curve

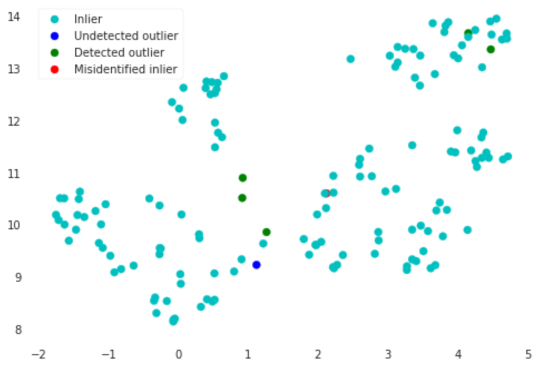

Figure 14: Lymphography Dataset Prediction Plot

Wisconsin-Breast Cancer Dataset

Running the improved algorithm on the WBC dataset, there was a significant time improvement with it only taking around 1min 20sec, 40 seconds faster than the previous. The size of the Pareto curve was similar again with around 36 individuals, however the shape of the curve was visibly smoother than before (**Figure 16**). The individual by best precision was worse on recall than the previous algorithm, at 1.0 precision and 0.38 recall, however the best by recall was better overall at 0.71 precision 0.71 recall. The automatically selected was similar but slightly worse at 0.8 precision 0.57 recall, which was interesting because the number of outliers fitness was the same, yet the distance fitness was 0.58 higher at 19.1. This is likely because there are outliers present which are close to each other, and when not both identified will cause a smaller closest distance on the ideal outlier.

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Average**
Num Outliers Fitness	0.0397	0.0344	0.0423	0.0423	0.0397	0.0397
Distance Fitness	19.1	17.86	15.96	19.57	14.71	17.44
Precision	**0.8**	**0.85**	**0.69**	**0.81**	**0.53**	**0.74**
Recall	**0.57**	**0.52**	**0.52**	**0.62**	**0.38**	**0.52**

Figure 15: WBC Dataset Results Table (Version 2.0 Algorithm)

Saving the results across 5 seeds (**Figure 15**), both average precision and recall was slightly lower than the previous algorithm, so there was no improvement on results there. Neither of the two best runs based on precision (Run 2) and recall (Run 4) performed as well as the best individual on the previous program either, however Run 4 (**Figure 17**) was not far behind (0.065 lower precision, 0.05 lower recall). Looking at the fitness values, there was also very little difference on average with this algorithm only 0.001 lower on number of outliers and 0.3 lower on distance fitness. Other than the one run that performed terribly (Run 5 with only 0.38 recall), the overall results were very similar, and although they were slightly worse on our new algorithm, we managed to cut the training time by a third.

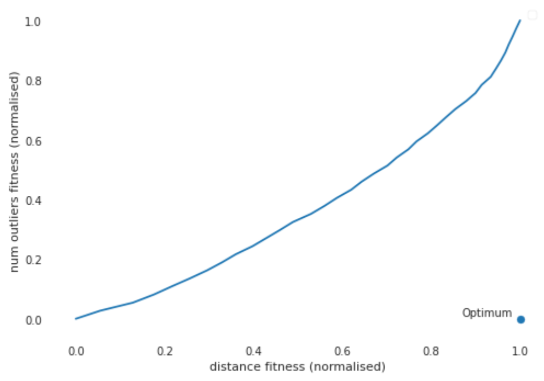

Figure 16: WBC Dataset Pareto Curve

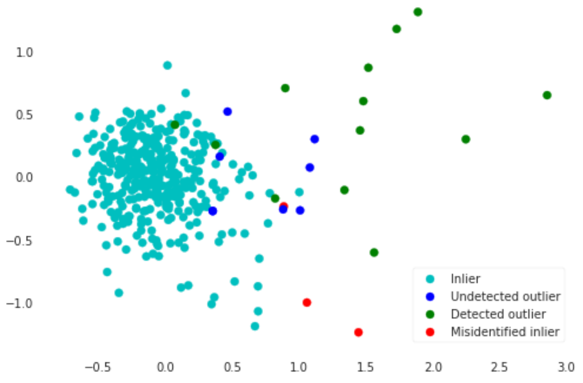

Figure 17: WBC Dataset Prediction Plot

Satimage Dataset

On the Satimage dataset initial results looked extremely similar to the previous algorithm, however the main difference was the much faster new runtime. It now took only 52 minutes, which was less than half the previous time of 2 hours. The Pareto curve (**Figure 19**) was large once again with a similar number of individuals, with the best by precision predicting 38 outliers correctly for a 1.0 precision and 0.535 recall.

	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Num Outliers Fitness	0.0112	0.0103	0.0122	0.0103	0.0107	0.0109
Distance Fitness	24.996	24.969	26.481	23.311	24.415	24.834
Precision	**0.6**	**0.67**	**0.31**	**0.67**	**0.61**	**0.572**
Recall	**0.55**	**0.56**	**0.31**	**0.56**	**0.54**	**0.5**

Figure 18: Satimage Dataset Results Table (Version 2.0 Algorithm)

This time we ran the algorithm 5 times with different seeds like we did on the smaller datasets and saved to the table **Figure 18**. Two runs (Runs 2 and 4) performed equally best with precision 0.67 and recall 0.56, and they also had the same number of outliers fitness. However, the distance fitnesses were not the same. When comparing the plots of both (one shown in **Figure 20**), the selection of outliers they each found was slightly different despite returning the exact same number.

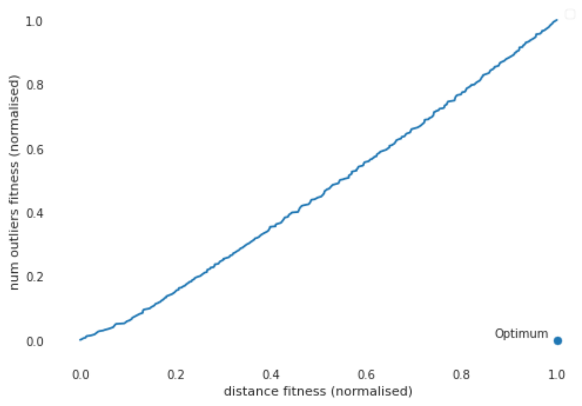

Figure 19: Satimage Dataset Pareto Curve

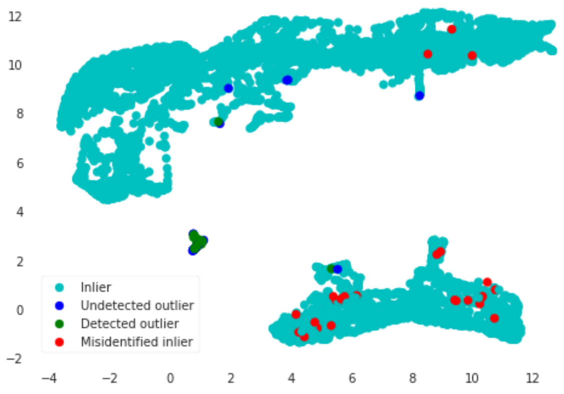

Figure 20: Satimage Dataset Prediction Plot

5. Conclusion

5.1. Evaluation

Unsupervised outlier detection is certainly a difficult task, and particularly when using GA, it gets significantly harder the larger the dataset in both size and dimensionality. In our experiments we managed to achieve reasonably high precision and recall on the smaller datasets, but we found that when the dataset increased in scale, not only did the individual representation increase and become more computationally expensive to perform operations on, but it was also much harder for the right bits to be assigned correctly for detecting the outliers themselves. This is because the nature of a GA relies on pure random chance to find new promising individuals, and as the size increases the more options there are and the lower the chance of these ideal candidates being found. There is potential here for a more direct optimisation, making use of a local search through a memetic algorithm could help the GA find outliers more efficiently, but with such large individual representation these could quickly become expensive.

Keeping the number of outliers low is another challenge of its own, particularly for our algorithm where the more outliers in an individual the more closest-distance computations needed to be performed. Having a cut-off threshold in the fitness function worked in most cases on the smaller datasets. However, when the chromosome size gets too long or the mutation rate gets too high, it can become a problem when too many potential individuals are lost for simply having as low as just one extra outlier over the threshold. The solution we came up with worked on the first two datasets in our experiments but there is plenty of room for future improvement, particularly on the Satimage dataset that could only detect half of the outliers (**Figure 18**). Iterating through the outliers removing them based on a quality or probability metric (i.e. change in fitness upon removal) from worst to best would ensure the most promising outliers are saved across each generation, however like other local search techniques this is challenging to achieve efficiently.

Something else that should be considered is the memory usage and storage when performing large scale GA. In our first version the individual representation was made up of integers, which despite only using values 0 and 1 were still being stored as a 32-bit integer. The second version used float values, which each take up 64-bits, twice the memory usage. The way the individuals in our programs were being stored in their array format however, both datatypes took up the same amount of memory (1272 bytes on lymphography dataset). This is still something that needs to be considered for future optimisation, as we believe the representation on our first algorithm (that used 0s and 1s) might be able to be optimised further if some sort of Boolean variable format was used instead.

5.2. Future Work

Going forward, we think evolutionary techniques that use individual representations which don’t scale significantly with the size of the dataset could be the better option, and therefore a different approach such as Genetic Programming (GP) is worth investigating. Using GP, a program designed to determine whether a given sample is an outlier based on its data values could be used. A terminal set would include the set of variables or columns in the dataset, and an output value could be classed as outlier or inlier depending on a threshold, for example < 0 inlier and >= 0 outlier. Such a continuous value could even be used to determine the likelihood of a given point being an outlier, rather than being strictly classed as one or the other. The flexibility of evolutionary computation allows such an individual representation to be compatible with our previously used multi-objective fitnesses, and investigations could be performed and directly compared. Despite not having the issue of scaling with the dataset size, the distance measure fitness would still have to be calculated at every evaluation step – something we improved with the second version program, and therefore a different fitness function might be needed for GP.

Despite not achieving results as high and reliable as we would’ve liked, we were still able to take an existing program and significantly improve its computation time. Evolutionary algorithms show strong potential for unsupervised outlier detection, but extensive work needs to be done to choose and optimise the right representation and method. There are many algorithms for performing EC and parameters to fine tune, and therefore plenty of room for future improvement.

6. References

[1] V. L. Cao, M. Nicolau, J. McDermott, "One-Class Classification for Anomaly Detection with Kernel Density Estimation and Genetic Programming", *European Conference on Genetic Programming*, University College Dublin, Dublin, Ireland, March 2016

[2] A. M. Jabbar, "Local and Global Outlier Detection Algorithms in Unsupervised Approach: A Review", *Iraqi Journal for Electrical and Electronic Engineering*, Shatt Al-Arab University College, Basra, Iraq, June 2021

[3] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, April 2002

[4] G. Folino, F. S. Pisani, L. Pontieri, P. Sabatino, M. A. Haeri, "Using Genetic Programming for Combining an Ensemble of Local and Global Outlier Algorithms to Detect New Attacks", *Genetic and Evolutionary Computation Conference (GECCO '19)*, Prague, Czech Republic, July 2019

[5] A. Banerjee, "Density-based evolutionary outlier detection", *Genetic and Evolutionary Computation Conference (GECCO '12)*, Philadelphia, Pennsylvania, USA, July 2012

[6] M. H. Marghny, A. I. Taloba, "Outlier Detection using Improved Genetic K-means", *International Journal of Computer Applications*, August 2011

[7] K. Krishna and M. N. Murty, "Genetic K-Means Algorithm", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, June 1999

[8] K. D. Crawford, R. L. Wainwright, "Applying Genetic Algorithms to Outlier Detection", *The Sixth International Conference on Genetic Algorithms*, Pittsburgh, Pennsylvania, USA, July 1995

[9] A. Duraj, Ł. Chomatek, "Outlier Detection Using the Multiobjective Genetic Algorithm", *JOURNAL OF APPLIED COMPUTER SCIENCE*, Vol. 25, No. 2, 2017

[10] I. Kononenko, B. Cestnik, "Lymphography dataset", *UCI Machine Learning Repository*, 1988

[11] Dr. W. H. Wolberg, W. N. Street, O. L. Mangasarian, "Wisconsin-Breast Cancer (Diagnostics) dataset (WBC)", *UCI Machine Learning Repository*, 1995

[12] A. Srinivasan, "Statlog (Landsat Satellite) dataset", *UCI Machine Learning Repository*, 1993