BENCHMARKING ALGORITHMS FOR DETECTING ANOMALIES IN LARGE DATASETS

Uriel Carrasquilla

Abstract

After reviewing state-of-the art anomaly detection algorithms and their effectiveness in dealing with both scattered and cluster anomalies, this research benchmarks the following algorithms based on their anomaly detection capabilities and their poly-logarithmic time-space complexity: Isolation Forest, Random Forest, ORCA, Artificial Neural Networks and C4.5. Challenges faced during the benchmark were issues such as memory constraints, many attributes, and large file size. The best results for isolating anomalies might be a combination of these algorithms.

Chapter 1: Introduction

Data mining for the discovery of knowledge in databases has been generating a significant amount of interest in research (Ramaswamy, Rastogi & Shim, 2000). The focus has been on finding "large patterns" that can't be easily found using more conventional statistical methods. Many businesses have been applying these new tools to gain a competitive advantage in the market place by, for example, understanding buying patterns from customers. The converse problem of finding small patterns (a.k.a. outliers or anomalies) has also gained a significant amount of research as stated by Yang, Latecki and Pokrajac (2009): "Outlier detection has recently become an important problem in many data mining applications." Again, businesses have been applying these tools for applications such as credit card fraud detection, network intrusions, and virus detection. Outliers, exceptions, peculiarities, surprise, novelties, anomalies, rare events, black swans, noise, etc. are some of the names often used to refer to a pattern in the data that does not conform to the expected behavior. Outlier problems have been studied under headings such as rare class mining, chance discovery, novelty detection, exception mining, noise removal, and data cleansing. The one common characteristic describing outliers is that their behavior occurs relatively infrequently but with a noticeable impact. Or better put by Ting, Tan and Liu (2009), "The detection of anomalies often provides critical actionable information and brings about significant impact on the task at hand." For instance, credit card fraud, cyber intrusions, virus detection, faults in mechanical components are often caused by anomalies that are subtle and difficult to detect. Bad data from faulty monitors can be detrimental to Capacity Planning because it can lead to incorrect sizing of hardware acquisitions.

Problem Statement and Objectives

The problem is "detecting both scattered and clustered anomalies in large datasets." The hypothesis is that by sampling, isolating, and reducing the number of relevant attributes and by working with trees, distances and densities the best algorithms will produce high detection rates, low false alarm rates and acceptable poly-logarithmic time-space complexities. During the literature review, many algorithms were found but few can deal with the challenges in detecting anomalies in large datasets in a poly-logarithmic

time and space complexity (NP-Complete problem). From the literature review, the selected algorithms for benchmarking are Isolation Forest, Random Forest, ORCA, Artificial Neural Networks and J48. For benchmarking purposes, this paper proposes using detection rate, false alarm rate, and time-space complexity as the criteria for evaluating results from the selected algorithms. Next are the objectives to be investigated.

The three objectives for this research, each of which establishes a research question, are now stated. The first objective is to benchmark best-in-class anomaly detection algorithms found during the literature review. The research question to be investigated is the efficiency of these selected algorithms based on their time and space complexity in dealing with large number of attributes, dimensions, or both. The second objective is the assessment of data quality (Type I and II errors) in a cleansed dataset produced from the different algorithms. Assessing data quality is a challenge to measure quantitatively unless labeled datasets are available. In labeled datasets, it is known ahead of time which records are outliers. Therefore, some labeled datasets will be required such as those from KDDCUP99 (2010), UCI (2010) and WEKA-UCI (2010). The third objective is the evaluation of the feasibility of automating the data cleansing process with a scoring method that can be used to sort anomalies from most likely to least likely. This would allow the researcher to apply different thresholds to control tolerance for Type I and II errors. As Ting, Tan and Liu (2009) pointed out, "ranking measure is of prime importance in anomaly detection tasks because it is required to rank the instances from the most anomalous to the most normal."

Chapter 2: Review of the Literature

The purpose of this section is to review what other researchers have been doing to solve the anomaly detection problem.

Cleansing data from anomalies is of interest not only in academic circles but crucial for data collection, data processing and data maintenance in most organizations that collect data as part of their business. In capacity planning, data is normally collected from multiple monitors operating under multiple technical architectures that often is normalized and aggregated into significantly large OLAP databases. Data anomalies and impurities are estimated at up to 5% of all the terra bytes of data currently stored in most commercial databases (Redman, 1998). Domain experts often spend a significant part of their time in data cleansing activities. The experts such as performance analysts rely on a combination of both automated and manual procedures to purify the data. This semi-automatic process is prone to inconsistencies in data quality and varied degree of success depending on the ability of the in-house domain experts (Muller, H. and Freytag, J. C., 2003). The challenge facing organizations following this semi-automatic strategy is the exponential growth in the volumes of data to be cleansed compared to the development of in-house domain experts. This realization has fueled the development of Data Mining tools that can be seen as additional experts that may complement available human experts. The R (2010), WEKA (2010) and SAS (2010) software tools contain Data Mining algorithms that can assist in detecting anomalies. These tools require a learning curve but they are easier to master compared to writing new software or trying to visually impact charts for errors in the data.

Muller and Freytag (2003) concluded that data cleansing targets errors in data but what constitutes an error is not a standard term. Worse, what may be an error in one application may not be an error in another application in either the same or different domains of knowledge. This inconsistency has led to methods and solutions that concern themselves with domain specific heuristics and algorithms that cannot easily be generalized to other sources and domains. For instance, algorithms for finding anomalies in a zOS SMF/RMF data may not work for Unix SAR collected data. It seems that the most thrust in the research of data cleansing have come from the data warehousing, data mining and data quality management disciplines. From the Chandola, Banerjee and Kumar (2009) work, the most common methods applied are statistical, cluster, pattern or rules based. It may very well be the case that for some applications it might make more sense to apply different methods concurrently to detect anomalies and based on the results from the different methods, vote on which suspects should be considered anomalies. In other words, all the methods are treated as experts in their own right and their

input is taken to another level for a final decision as to what instance records to include in the final cleansed dataset. The arbiter for this final decision could be an expert or yet another algorithm.

Nearest Neighbor Algorithms

Distance based algorithms are normally quadratic in their time-space complexity with respect to the number of instances and attributes in the dataset. As indicated by Schwabacher (2005), Bay and Schwabacher developed ORCA to detect anomalies in near linear time using the nearest-neighbor technique. The reduction in time complexity for large datasets is achieved by the implementation of pruning rules. In their algorithm, distance calculations of k nearest neighbors are responsible for the largest demand of processing time. The parameter k determines the number of nearest neighborhood. Computational time is sensitive to this parameter. The parameter N controls the number of reported anomalies. This parameter controls the pruning cutoff so it also affects computational time. Liu, Ting and Zhou (2008) found that N=n/8 is a good guess estimate (n is the number of instances). ORCA is an ideal candidate for our benchmark purposes.

Classification Algorithms

Machine learning models make it possible to use non-linear approximations of the parameters in the functions to separate anomalies from normal instances. For the purpose of approximating the parameters in the function, the coefficients of the function decomposition are obtained from the input–output data pairs, some chosen model structure and systematic learning rules. Once trained, the machine learning model becomes a parametric description of the function. Learning a general principle from a set of specific training examples that were provided are achieved by trying out different model structures and the related parameters. For this paper, out of several viable possible methods considered, Artificial Neural Networks (a.k.a. ANN) models are applied to the datasets to be selected for testing. The models produce fitness to the data statistics for comparison purposes. ANN is the most widely used Machine Learning model. ANN is a broad term covering a large variety of network architectures, the most common of which is the multi-layer perceptron (MLP), the one selected for this paper. Such a network is trained by the so-called error-back-propagation method, which is a specialized version of the gradient-based optimization algorithm. In MLP, each target vector z is an unknown function f of the input vector x as in Equation 1.

$$z=f(x)$$

Equation 1 - MLP Model

The task of the network is to learn the function f. The network includes a set of parameters (weights vector), the values of which are varied to modify the generated function “f”, which is computed by the network to be as close as possible to “f.” The weight parameters are determined by training (calibrating) the ANN based on the training dataset. An often used parameter is 66% of the data reserved for training and the remaining for testing the predictive capabilities of the model.

Density Based Algorithms

LOF and LOCI are two anomaly detection techniques that originated from the field of knowledge discovery in databases and the claim is that they are best at finding outliers in large datasets (Janssens & Postma, 2007). The LOF (Local Outlier Factor) algorithm requires only one parameter “k” to indicate the number of neighbors constituting the neighborhood needed to access local density. The LOF algorithms construct the neighborhood, estimate the neighborhood density and compare the neighborhood densities. The LOCI (Local Correlation Integral Method) is supposed to be an improvement over LOF but as Janssens and Postma (2007) proved, this is not really the case and depends on the data. LOCI will not be further considered in this analysis since LOF is a good representation of the best-in-class. LOF is

another ideal candidate for our benchmark but unfortunately the tool could not be secured in time for this research.

Trees and Forests Algorithms

Random Forests, one of the most efficient misuse detection techniques, can be applied to real time applications such as network intrusion detection systems (Zhang & Zulkernine, 2005). Random Forests can also be directly applied to the problem of anomaly detection since the R (2010) Random Forest package has such option. These authors state that Random Forests is an ensemble classification and regression technique which is unsurpassable in accuracy. Sub-sampling is used by building trees for each sample and then voting on a given classification. The classification can be normal versus anomaly. Both the R (2010) and WEKA (2010) version of the Random Forest algorithm offer the option of attribute selection based on the contribution of each attribute to the classification of normal versus abnormal. This is important for large datasets with a large number of attributes where some of the attributes are irrelevant. In Random Forests, a training dataset is used off-line to build the pattern-detector model needed to detect anomalies in the on-line system (Figure 1).

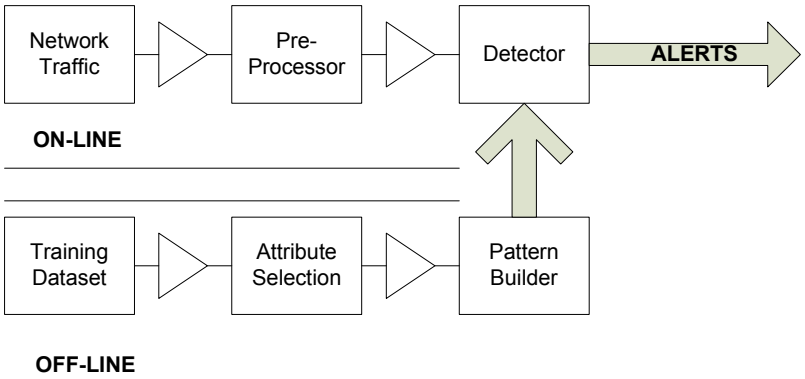

Figure 1 - Data Mining Model

The most important contribution of Random Forests is the fact that by working with a fixed number of trees and a subset of the attributes computational time and memory requirements can be kept almost constant regardless of the number of instances and attributes in the training dataset during the off-line processing phase. Then, in the real time phase, finding patterns is based on the same fixed number of trees built during training resulting in time complexities that change in a liner relationship to the number of instances in the on-line dataset. Another significant contribution of Random Forests is the technique available to calculate the importance value of the attributes in the input dataset. This leads to a reduction of attributes to be included for analysis which can significantly reduce computational complexity for datasets with many irrelevant attributes or attributes that don't add information needed to build patterns. Random Forest is an ideal candidate for benchmarking.

A competing approach to Random Forest is the Isolation Forest (a.k.a. iForest) proposed by Liu, Ting and Zhou (2008). In their approach they use isolation of anomalies, sub-sampling, attribute selection and the scoring of anomalies. The authors demonstrate that their algorithm is capable of a nearly linear time complexity with a low memory foot print. The author compared iForest to ORCA, LOF, and Random Forest but ignored Fast-RF. Their strategies were promising for dealing with large datasets in terms of detection rate and processing CPU time for a group of widely available datasets (results can be found in Appendix A of this paper). Their focus is on continuous variables with little indication on how to handle categorical data. The take away from this research is the authors' insights on the use of small samples of no more than 256 instances and on keeping the number of trees to no more than 100 regardless of the size of the dataset. The explanation for their choices is masking and swamping. Masking is the existence of too many anomalies that may conceal their own presence. Swamping refers to false alarms caused when normal instances have a close proximity to anomalies making it hard to distinguish

anomalies from normal instances. The authors believe that the problem of clustered anomalies can be overcome by their algorithm but no proof is given. In addition, the authors use a two phase approach to the iForest implementation similar to the one in Figure 1. However, in the on-line stage a score is produced indicating the level of belief from the vote of all the trees about how abnormal a candidate instance might be. The approach proposed for attribute selection, a separate activity before running the model, is based on the use of the Kurtosis statistical test to determine which attributes to include in the samples before constructing each tree. The authors indicated that iForest has a time complexity of $O(t\psi log\psi)$ in the training phase and $O(ntlog\psi)$ in the evaluation phase where t=100 (number of trees), ψ=256 the number of instance per sample (one sample per tree), and "n" is the number of instances in the dataset. Therefore, iForest is another ideal candidate to benchmark.

Another "tree" based algorithm is the C4.5. J48 is a Java implementation of the C4.5 algorithm in WEKA (2010). This algorithm builds trees during the training phase by measuring the uncertainty associated with each random variable; a process known as entropy. The training data is a set $S = s_1,s_2,... s_n$ of pre-classified samples where each $s_i = x_1,x_2,... x_n$ is a vector and $x_1,x_2,... x_n$ represent attributes. When working with labeled datasets, the training data also contains the vector $C = c_1,c_2,... c_n$ where $c_1,c_2,... c_n$ represent the class to which each sample belongs (e.g. 0-normal, 1-anomaly). During the training phase, at each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The criteria for the C4.5 decision is based on the estimated normalized information gain, i.e. the difference in entropy, which results from choosing an attribute for splitting the data. The decision is based on the attribute with the highest normalized information gain.

Case Datasets

For our benchmark purposes, some of the UCI (2010), WEKA-UCI (2010) and KDDCUP99 (2010) datasets can be used. The UCI repository is explained by Bay et-al (2000). Its purpose is to combine data from a variety of fields. Datasets such as shuttle, satellite, pima, breast, arrhythmia, ionosphere and others are labeled (meaning they contain known anomaly classes). The KDDCUP99 (2010) network intrusion data has to be prepared using the specifications are described by Yamanishi et-al (2004).

Dataset	**# instances**	**# attributes**	**Anomaly class**
ForestCover	581,012	55	Class 4 (0.5%)
Shuttle	58,000	10	Classes 5, 6, 7 (6%)
Mammography	830	6	Class 1 (49%)
Annthyroid	3,772	22	Classes 1, 2 (7%)
Satellite	6435	37	3 smallest classes (32%)
Pima	768	9	Pos (35%)
Breastw/Wisconsin	683	11	Malignant class 4 (35%)
Ionosphere	351	35	Bad (36%)
Poker	1,025,010	11	Class 6, 7, 8, 9 (0.2%)
http (KDDCUP99)	567,488	4	Intrusion (0.39%)
Smtp (KDDCUP99)	95,156	4	Intrusion (0.03%)

Table 1 - Dataset Properties

Chapter 3: Methodology

Design Specifications

From the CRIPS-DM (2010) methodology, the following steps are followed: domain understanding, data understanding, data preparation, modeling, results evaluation and results deployment. Both domain and data understanding are downplayed in this research by working with algorithms that have minimum, if any, dependency from experts to manipulate the data, select the samples, isolate anomalies, and reduce the number of attributes to be included in the samples. To this end, the algorithms are required to select attributes, select samples, build trees, and select parameters for their models based on the information available from the training dataset. Data understanding, important but not compulsory, should be explored using tools that provide data exploration and visualization. To this end, the proposed tools should provide such facilities but only R (2010) and WEKA (2010) have such facilities built in. During data preparation, the CSV dataset format is required. The modeling step consists of adapting the labels to represent normal and abnormal instances and then based on the algorithm, select the appropriate representation of the data. The results evaluation step is performed by estimating the quality of the models by comparing Type I and II errors after running each of the algorithms. During the results deployment step, the alerts and scores for anomalies are triggered. This methodology is expected to be iterative until acceptable results are obtained.

Business Understanding

The project objective and requirement is to remove anomalies from the data. This translates into a data mining problem where patterns are learned during the training stages for alerting purposes when the real data is used.

Data Understanding

The datasets to be used are those found during the literature review and summarized in Table 1. The datasets selected, as discussed by Bay et-al (2000) have a wide range of instances and attributes making them ideal for benchmarking algorithms. All the datasets contain an anomaly class indicating if the instance is abnormal or normal. As indicated in Table 1, we know ahead of time the percentage of anomalies present in each dataset. The percentage range is anywhere from 0.03% to 36%. Most of the anomalies are scattered but some of them are clustered anomalies. The datasets are representative of network intrusions, medical analysis, scientific analysis, forestry, and gambling. The plan is to test the ability of the different algorithms when working with very different data domains.

Data Preparation

The data preparation is driven by the requirements of the different tools and algorithms. The common denominator is to convert all the datasets to CSV format. The KDDCUP99 datasets required a program to be written to break up the original UCI dataset into the HTTP and SMTP datasets. The processing requirements for this program are as documented by Yamanishi et-al (2004). The ORCA tool requires its input data to be in binary format. Conveniently, the ORCA tool comes with a binary conversion utility named DPREP that reads the CSV file along with a NAMES file that must be prepared. In addition to the NAMES file, there should also be a WEIGHTS file indicating the weights each attribute is to have in terms of importance. To keep things simple, a weight of 1 can be assigned to each attribute. The NAMES file shows each attribute, one per instance, which tells the utility if the attribute is a nominal or numeric attribute. If it is nominal, valid values must be indicated.

Modeling

The modeling depends on the algorithm and tool to be used. The algorithms selected during the literature review are iForest, Random Forest (a.k.a. RF), ORCA,ANN and J48. When the R (2010) tool is used, the model relates the X matrix to the Y vector (Y ~ X). The following is an example for processing the SMTP dataset after splitting the original dataset into training (trainRF) and testing (testtRF) subsets. The datasets contain 4 attributes (duration, src_bytes, dst_bytes, and class). For the X matrix, the 4th attribute is ignored (-4). For the Y vector (the class containing the labels normal/abnormal), only the fourth attribute is selected. The other parameters used are to build with 100 trees and trace trees up to five levels deep.

```
randomForest (x=trainRF[,-4],xtest=testRF[,-4], y=trainRF[,4], ytest=testRF[,4], ntree=100))
```

The following is a sample when using the iForest algorithm on how to set up R for processing the HTTP dataset. The dataset is in datRF. There is no need to split the dataset into training and testing since the algorithm is very efficient in sub-sampling large datasets. The parameters are as the ones recommended by Liu, Ting, and Zhou (2008): the forest consists of 100 trees with a depth of 10 per tree and each tree built from samples of 266 instances.

```
IsolationTrees (datRF,ntree=100, hlim=10, nRowSamp=266))
```

In the case of the WEKA tool, a GUI is presented that allows the selection of each classifier (algorithm), the selection of the data (in CSV or ARFF format), and the parameters to run the model. The following is a sample when using the RF algorithm on how to set up WEKA for processing the HTTP dataset. The class variable (i.e. RLAST) must be changed from numeric to nominal.

```
Scheme:       WEKA.classifiers.trees.RandomForest -I 10 -K 0 -S 1
Relation:     http-WEKA.filters.unsupervised.attribute.NumericToNominal-Rlast
Instances:    567498
Attributes:   duration, src_bytes, dst_bytes, class
Test mode:    split 66.0% for training, remainder for testing
```

The following is a sample when using the J48 algorithm on how to set up WEKA for processing the HTTP dataset.

```
Scheme:       WEKA.classifiers.trees.J48 -C 0.25 -M 2
Relation:     http-WEKA.filters.unsupervised.attribute.NumericToNominal-Rlast
Instances:    567498
Attributes:   duration, src_bytes, dst_bytes, class
Test mode:    split 66.0% for training, remainder for testing
```

The following is a sample when using the ANN algorithm (technically the Multilayer-Perceptron algorithm) on how to set up WEKA for processing the SMTP dataset.

```
Scheme:       WEKA.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Relation:     smtp-WEKA.filters.unsupervised.attribute.NumericToNominal-Rlast
Instances:    95156
Attributes:   duration, src_bytes, dst_bytes, class
Test mode:    split 66.0% for training, remainder for testing
```

The ORCA tool is more involved requiring several steps as follows.

1. Prepare the NAMES file.
2. Convert the CSV input file to binary using the DPREP utility provided with ORCA.
3. Run ORCA with the “-n number” parameter where “number” is from the percentage of records known a priori to be labeled abnormal.

Evaluation/Experiments

Once the models have been built using scripts, they should be run to capture the quality metrics required for this research. The metrics are the prediction-rate, false-alarm-rate, CPU-Time, and Max-Memory required. Depending on the tool, the calculations may be already part of the tool. In the case of WEKA, this information is provided by the tool along with the AUC metric as used by Liu, Ting and Zhou (2008). But for ORCA and R, the scripts must be programmed to perform the calculations. In the case of ORCA, the script must track the "count" of those instances properly estimated to be an anomaly. The script should also sort from highest score to lowest score. The closer the score to 1, the higher the probability that the instance is an anomaly. The following is an excerpt of what the code may look like for estimating these two metrics.

```
my $predRate = sprintf ("%0.2f", 100 * (abs($#sorted - $count)) / $#sorted);
my $falseAlarmRate = sprintf("%0.2f",  100 * ($count) / $num_temp);
```

The following is a sample when using the iForest algorithm on how to use R for calculating the prediction-rate and false-alarm-rate when the threshold from the score is pre-set at 0.5. The iForest scoring is from 0 to 1 and anything over 0.5 is a candidate anomaly.

```
rf1.predicted<- as.numeric(as$outF > .5)
pred.rate <- round(sum(rf1.predicted == datRF$class) / dim(datRF)[1], 3)
false.alarm.rate <- round(sum(rf1.predicted == '1' &  datRF$class == '0') /   dim(datRF)[1], 3)
```

The following is a sample when using the Random-Forest algorithm on how to use R for calculating the prediction-rate and false-alarm-rate.

```
pred.rate <- round(sum(RF1$predicted == trainRF$class) / dim(trainRF)[1], 3)
false.alarm.rate <- round(sum(RF1$predicted == '1' &  trainRF$class == '0') /  dim(trainRF)[1], 3)
```

When using R, the CPU Time and Memory can be measured as follows. First reset and free up memory. Then run the model capturing the amount of CPU-time consumed. Then calculate MEM-MAX (memory used) and CPU-Time that includes user and system usage.

```
gc(reset=TRUE)
cpu.time <- system.time(RF1 <- get(my.algorithm)(datRF,ntree=100,hlim=10,nRowSamp=266))
mem.max <- memory.size(max = TRUE)
cpu.time <- sum(cpu.time[c('user.self', 'sys.self')])
```

Deployment/Experiments

Once the scripts are prepared and the datasets downloaded then it is time to execute the scripts and keep track of the results. The results from each script can be summarized into an Excel spreadsheet to keep track of the algorithm, dataset, instances, attributes, CPU-time, Maximum-Memory, Prediction-Rate, and False-Alarm-Rate. Based on those results, R can be used to build linear regressions and charts to complete the time-space complexity analysis required by this research. The following code can be used to generate such linear models for the case of the iForest and RF algorithms.

```
iForest.model <- lm(iForest$CPUTime~iForest$Instances+iForest$Attributes)
RF.model <- lm(RF$CPUTime~RF$Instances+RF$Attributes)
```

Then based on these two R models, the charts can be produced. For instance, to print the chart for the iForest model above, the following code can be used:

```
plot(iForest.ordered$Instances, iForest.ordered$CPUTime, col="red", type="l", ylab="CPU Time",
xlab="Instances")
abline(lm(iForest$CPUTime~iForest$Instances),col='green')
title(main="iForest CPU vs Instances", col.main="blue")
```

Equipment/Experiments

The experiments can be run on a Microsoft Windows XP machine with the following characteristics: XP professional, version 2002, service pack 3, Intel core 2 quads Q9400 at 2.66 GHz with 3.49 GB of RAM.

Research Questions

The approach to measure and answer the research questions should be as follows:

1. The issue of scalability and polynomial behavior for the anomaly detection algorithm can be demonstrated by showing the CPU time and memory allocated to process each dataset by each of the algorithms being benchmarked.
2. The quality of the algorithms being benchmarked can be demonstrated by showing the anomaly prediction rate and the false alarm rate. These rates are representative of Type I and II errors. That is, type I (α): reject the null hypothesis when the null hypothesis is true, and type II (β): accept the null hypothesis when the null hypothesis is false. The prediction rate is calculated by dividing the number of correctly detected anomalies to the total number of anomalies known based on the labels for each of the datasets. The false alarm rate can be calculated by dividing the number of wrongly assigned anomalies to the total number of records.
3. The sensitivity to the number of instances and attributes can be demonstrated by showing the linear regression on CPU time against the number of instances and attributes for each dataset.

This chapter presented the methodology and methods to calculate and compare the results from the selected algorithms to process the datasets in Table 1. The next chapter shows the results after the experiments described in this chapter are executed.

Chapter 4: Findings

The results from running the Isolation Forest (iForest), Random Forest (RF), Artificial Neural Networks (ANN) and J48 algorithms against the UCI datasets from Table 1 are summarized in Table 2, 3 and 4.

The headings in Table 2 have the following meaning. The algorithm heading refers to the pre-selected models being benchmarked. The file name is the dataset used from Table 1. The number of instances, the number of attributes, the experiment's CPU Time (in seconds), memory used, prediction rate and false alarm rates are calculated as described during the methodology. In the case of RF, the number of instances shown is for the "training" dataset for the given file name. As described in the methodology, the training dataset size is 66% of the total dataset with the remaining instances reserved for testing the model. In the case of iForest, the entire number of instances is shown. The separation between training and testing was compulsory for large datasets due to the fact that R requires the entire dataset to be in memory. In a 32 bit machine such as the one used in this research, that was a limitation. In both Table 3 and 4, the headings have the same meaning and the only difference is the algorithm used. In the case of ORCA, only the large datasets were benchmarked and it proved to be efficient in the handling of memory. In the case of the WEKA tool, it experienced the same memory limitations as R when executing RF. The ANN algorithm proved to be the slowest and J48 the fastest under WEKA. RF was significantly slower under WEKA compared to R.

For easy access and comparison purposes, the results from Liu, Ming and Ting (2008) experiments with some of the same datasets are available in Appendix A. As a side note, the results from our benchmark is different from the results in Appendix A. One explanation is that the RF algorithm worked for more of

the datasets in our research by changing the model to work with training and test datasets. There are other differences between the results in this research compared to Appendix A and that is a concern that will need more scrutiny. Some of those differences can be explained by the method used to estimate quality. In this research, prediction-rates and false-alarm-rates were used instead of the area-under-the-curve (AUC). The WEKA tool reports AUC and the AUC but even those results were different. The most significant finding in this research compared to the reported findings from Liu, Ming and Ting (2008) is that the Random Forest algorithm seems to out-perform the iForest algorithm in every respect: CPU time, Memory usage, prediction rate and false alarm rate. Another interesting finding is that the ANN and J48 algorithms produce high quality results compared to iForest, ORCA, and RF. Their only limitation is the size of the datasets they can handle is limited making them poor choices for large datasets. As an idea for future work is to see how ANN and J48 behave outside the WEKA tool. Another idea is to combine these algorithms in a two phase process: pre-select instances for more analysis using a sampling method such as iForest followed by the ANN or J48 algorithm against the pre-selected instances.

In Table 3, the memory usage could not be measured due to the lack of proper tools when running ORCA. This can be overcome manually by looking at the task manager but there would be no audit trails to support those readings.

In Table 4, it can be seen that the WEKA tool ran into memory issues when running large datasets. This problem could be overcome with 64 bit machines and more memory. Eventually, and for datasets large enough, this could also become a problem, but this conjecture will require more work to prove.

Algorithm	FileName	Instances	Attributes	CPUTime	MemMax	PredRate	FalseAlarm
iForest	annthyroid	3772	22	0.66	36.12	0.914	0.016
iForest	breastw	683	11	0.21	24.69	0.927	0.035
iForest	forestCover	581012	55	708.24	685.06	0.994	0
iForest	ionosphere	351	35	0.15	20	0.806	0.057
iForest	mammography	830	6	0.22	27.19	0.508	0.099
iForest	pima	768	9	0.2	25.94	0.667	0.042
iForest	poker	1025010	11	393.64	321.38	0.145	0.851
iForest	satellite	6435	37	1.54	43.69	0.719	0.007
iForest	shuttle	58000	10	10.39	61.19	0.944	0.001
iForest	http	567498	4	42.03	162.06	1	0
iForest	smtp	95156	4	3.9	53.62	0.994	0.006
RF	annthyroid	3772	22	14.94	382.94	0.997	0.003
RF	breastw	683	11	0.61	29.31	0.971	0.018
RF	forestCover	406011	55	589.56	965	0.997	0
RF	ionosphere	351	35	0.24	19.88	0.932	0.023
RF	mammography	830	6	0.72	36.06	0.806	0.101
RF	pima	768	9	0.68	33.31	0.766	0.095
RF	poker	299999	11	157.66	843	0.998	0
RF	satellite	4435	37	1.93	23.56	0.927	0.021
RF	shuttle	43500	10	5.52	40.94	1	0
RF	http	200000	4	24.2	231.56	1	0
RF	smtp	50000	4	3.25	45.38	1	0

Table 2 – R Tool

In terms of scoring each instance between 0 and 1, with 1 being the most likely anomaly, only iForest and ORCA produce such results. Since we worked with labeled datasets we knew a priori the percentage of records that were anomalies in each dataset. Therefore, the models were built to report anomalies for that same percentage of records after sorting from most likely anomaly to least likely.

Algorithm	FileName	Instances	Attributes	CPUTime	MemMax	PredRate	FalseAlarm
ORCA	shuttle	58000	10	4.70	?	0.108	0.051
ORCA	smtp	95156	4	4.03	?	0.0	0.030
ORCA	http	567498	4	23.45	?	0.038	0.004
ORCA	poker	1025010	11	83.87	?	0.978	0

Table 3 - ORCA Tool

Algorithm	FileName	Instances	Attributes	CPUTime	MemMax	PredRate	FalseAlarm
ANN	shuttle	58000	10	327	?	0.999	0.003
ANN	breastw	691	10	15	?	0.952	0.047
ANN	ionosphere	351	35	59	?	0.912	0.088
ANN	smtp	95,156	4	128	?	0.999	0.001
ANN	http	567,498	4	FAILED	RAN OUT	?	?
ANN	forestCover	581012	55	FAILED	RAN OUT	?	?
ANN	poker	1025010	11	FAILED	RAN OUT	?	?
J48	shuttle	58000	10	3	?	0.999	0.002
J48	breastw	699	10	1	?	0.946	0.005
J48	ionosphere	351	35	1	?	0.915	0.008
J48	smtp	95156	4	1	?	0.999	0
J48	http	567,498	4	14	?	0.999	0
J48	forestCover	581012	55	FAILED	RAN OUT	?	?
J48	poker	1025010	11	FAILED	RAN OUT	?	?
RF	shuttle	58000	10	14	?	0.999	0.039
RF	breastw	699	10	1	?	0.961	0.071
RF	ionosphere	351	35	1	?	0.929	0
RF	smtp	95,156	4	11	?	0.999	0
RF	http	567,498	4	114	?	1	0
RF	forestCover	581012	55	FAILED	RAN OUT	?	?
RF	poker	1025010	11	FAILED	RAN OUT	?	?

Table 4 - WEKA Tool

Note: *breast and ionosphere used "cross-validation/folds=10", all other 66% for training.*

In figure 2, CPU time is shown against the number of attributes for the iForest and RF algorithms. Both algorithms are sensitive to not only the number of attributes but also the number of instances. The straight trend line is a linear regression between attributes (independent variable) and CPU time

(dependent variable). The relationship between attributes and CPU Time is distorted by the significant differences in the number of instances for the datasets. But in general, as the number of attributes goes up, the CPU Time goes up.

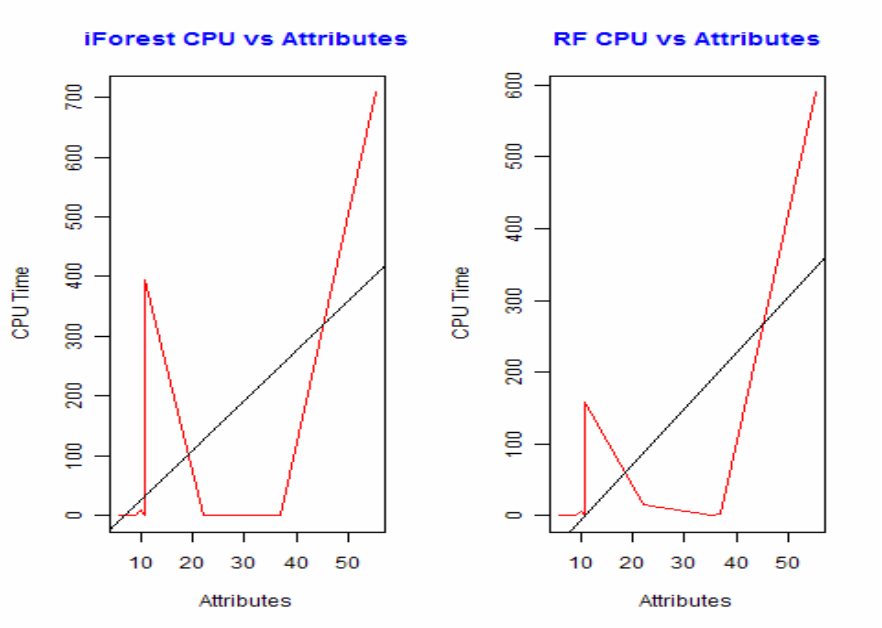

Figure 2 - CPU versus Attributes

In figure 3, CPU time is shown against the number of instances for each of the two algorithms. Both iForest and RF algorithms are sensitive to not only the number of instances but also the number of attributes. The straight trend line is a linear regression between attributes (independent variable) and CPU time (dependent variable). The relationship is more significant than the one in Figure 2 due to the weaker influence of attributes. Noticeable is the not so linear relationship between CPU Time and number of instances with iForest exhibiting more independence with respect to the number of instances.

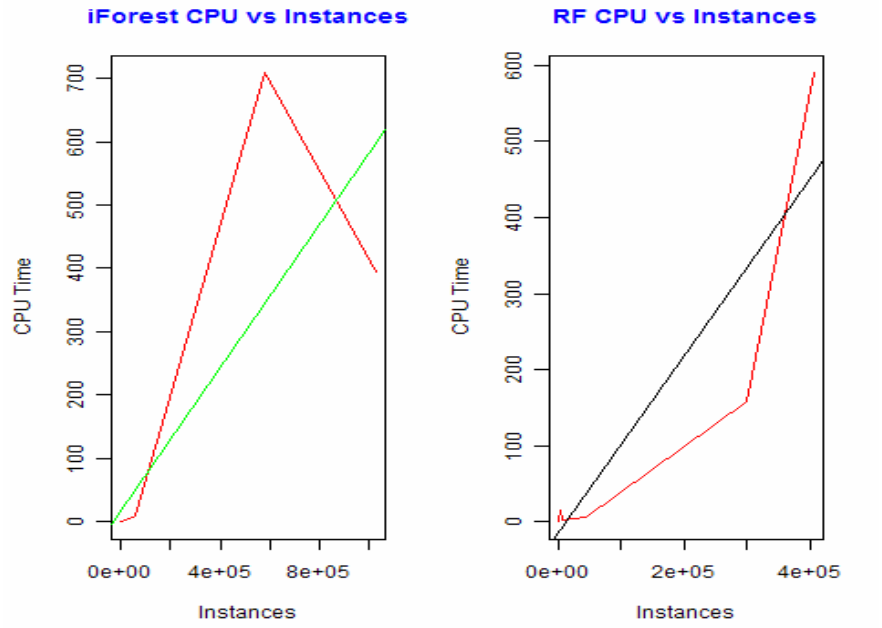

Figure 3 - CPU versus Instances

In Table 5, the log from R when trying to fit a linear regression model is shown. In this log, the linear model relates CPU Time (dependent variable) to both number of instances and number of attributes (the two independent variables). The resulting linear model is shown in Equation 2. Equation 2 shows the lack of sensitivity to the number of instances and more sensitivity to the number of attributes. Therefore, reducing the number of attributes before running any of the models is a worthwhile effort to reduce time-space complexities.

CPU-Time = -170 + 0.001 * Instances + 8.31 * Attributes

Equation 2- Regression Model

```
> #  linear regression to find CPU time based on Instances and Attributes by Model
> iForest.model <- lm(iForest$CPUTime~iForest$Instances+iForest$Attributes)
> RF.model <- lm(RF$CPUTime~RF$Instances+RF$Attributes)
> iForest.model
Call:
lm(formula = iForest$CPUTime ~ iForest$Instances + iForest$Attributes)
Coefficients:
      (Intercept)   iForest$Instances  iForest$Attributes
       -1.095e+02          5.057e-04          6.211e+00
> RF.model
Call:
lm(formula = RF$CPUTime ~ RF$Instances + RF$Attributes)
Coefficients:
  (Intercept)   RF$Instances  RF$Attributes
   -1.706e+02      1.349e-03      8.315e+00
```

Table 5 - Results from Models (R Tool)

Table 6 shows the results of running ORCA on the Shuttle dataset. What is a concern is the low prediction rate, high false alarm rate and the large amount of processing time.

```
--(1644:Thu,15 Apr 10:$)-- ./ORCA_shuttle.pl
04-15-2010:16:46:25 ORCA_shuttle RMUJC-HP CYGWIN:  "
      ORCA_shuttle Started
       debug     = 0
       help      = 0
       verbose   = 0
       directory  = /cygdrive/d/NOVA/DCIS-750-DBMS/HW4/datasets
       app       = UCI.csv
"
……….

04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "input file with 3289 anomalies"
04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "we selected top 3289 anomalies using
ORCA"
04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "2932 records do not match"
04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "Prediction Rate = 10.85"
04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "False Alarm Rate = 5.06"
04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "126 wallclock secs ( 4.55 usr  0.01 sys +
0.06 cusr  0.08 csys =  4.70 CPU)"
04-15-2010:16:48:31 ORCA_shuttle RMUJC-HP CYGWIN:  "ENDED PROCESSING"
```

Table 6 - Listing from running ORCA against Shuttle dataset

This concludes the findings chapter. The next chapter presents the conclusions from this research.

Chapter 5: Conclusions

This chapter presents the conclusions and recommendations from this research. The objectives for this study as stated in Chapter 1 were to select best-in-class algorithms to be benchmarked, compare their quality in detecting anomalies using prediction rate and false alarm rate metrics, study their CPU time and memory complexity to draw conclusions about their poly-logarithmic behavior, and to investigate the possibility of automating the data cleansing process using a scoring method that can be used to sort anomalies. From the previous findings chapter, it was found that ANN, J48 and RF produce the best quality results but ANN, J48 and to an extent RF; exhibit a non poly-logarithmic behavior. In terms of scoring, only ORCA and iForest produce such results. The findings also suggest that both R and WEKA have deficiencies in their memory management due to their requirement to keep the entire dataset in memory. This can limit their usability for extremely large datasets. It was also found that iForest can overcome this limitation by sampling and isolation. We also found that RF could be made to work by carefully selecting the training dataset to be only as large as memory would allow it to be. This put some extra burden when setting up the experiments and makes it very difficult to automate the data cleansing process. For automation purposes, iForest is probably the best way to go for numeric attributes since it cannot handle nominal attributes unless converted to numeric. Another shortcoming of the iForest algorithm is that attribute selection must be done a priori.

It seems that the best approach for isolating anomalies might be to perform the following steps:

1. Using J48 with multiple training datasets, select the attributes to be used for the iForest algorithms.
2. Using the iForest algorithm, pre-select candidate anomalies from the dataset by taking those instances that score over 0.50.
3. Using the ANN, J48 and RF as an ensemble, process the candidate anomalies.
4. The common denominator instances that are flagged by the ensemble (previous step) are most likely to be anomalies.

APPENDICES

Appendix A: Results from Liu, Ming and Ting's Comparison

Area Under the Curve (AUC)

Dataset	iForest	RF	ORCA	LOF
ForestCover	0.88	NA	0.83	NA
Shuttle	1.0	NA	0.60	0.55
Mammography	0.86	NA	0.77	0.67
Annthyroid	0.82	NA	0.68	0.72
Satellite	0.71	NA	0.65	0.52
Pima	0.67	0.65	0.71	0.49
Breast/W	0.99	0.97	0.98	0.37
Arrhythmia	0.80	0.60	0.78	0.73
Ionosphere	0.85	0.85	0.92	0.89

CPU Time (seconds)

Dataset	iForest	RF	ORCA	LOF
ForestCover	16.33	NA	6995.17	NA
Shuttle	3.13	NA	156.66	7489.74
Mammography	0.66	NA	4.49	14647.00
Annthyroid	0.51	NA	2.32	72.02
Satellite	1.63	NA	8.51	217.39
Pima	0.28	4.98	0.06	1.14
Breast/W	0.28	3.10	0.04	1.77
Arrhythmia	2.98	2.32	0.49	6.35
Ionosphere	0.48	0.83	0.04	0.64

References

Bay, S. D. and Schwabacher, M. (2003). Mining Distance-Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule. SIGKDD 2003.

Bay, S. D., Kibler, D., Pazzani, M. J., Smith, P. (2000). The UCI KDDD Archive of Large Data Sets for Data Mining Research and experimentation. ACM SIGKDD Explorations, Volume 2, Issue 2, pp 14 – 18.

Chandola, V., Banerjee, A. and Kumar, V. (2009). Outlier Detection – A Survey. ACM Computing Surveys (CSUR). Volume 41, Issue 3. Article 15. ISSN: 03600-0300.

CRISP-DM. (2010). The Cross-Industry Standard Process Model for Data Mining" or CRISP-DM for short. HTTP: www.crisp-dm.org

Janssens, J.H.M., Postma, E.O. (2007). One-Class Classification with LOF and LOCI : An Empirical Comparison. Proceedings of the 18th Annual Belgian Dutch Conference on Machine Learning BENELEARN

KDDCUP99. (2010). The Third International Knowledge Discovery and Data Mining Tools Competition. http://kdd.ics.uci.edu/databases/kddcup99/task.html

Liu, F.T., Ting, K.M. and Zhou, Z. (2008). Isolation Forest. Eighth IEEE International Conference on Data Mining. pp.413-422. (ICDM 08). ISBN: 978-0-7695-3502-9.

Müller, H., Freytag, J.C. (2003). Problems, Methods, and Challenges in Comprehensive Data Cleansing. HUB-IB-164, Humboldt University Berlin.

Perl. (2010). The Perl programming language. Http://www.perl.org/

R. (2010). R project for statistical computing. Http://www.r-project.org

Ramaswamy, S., Rastogi, R. and Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. ACM Volume 29 , Issue 2 (June 2000) pp 427 – 438. ISSN:0163-5808.

Redman, T. (1998). The impact of poor data quality on the typical enterprise. Communications of the ACM, volume 41, issue 2.

SAS. (2010). SAS Institute with the SAS tool. Http://www.sas.com

Schwabacher, Mark. (2005). Machine learning for Rocket Propulsion Health Monitoring. Machine Learning for Rocket Propulsion Health Monitoring. SAE World Aerospace Congress.

Ting, K.M., Tan, S.C., and Liu, F.T. (2009). Mass: A New Ranking measure for Anomaly Detection. Gippsland School of Information Technology. TR 2009/1. Submitted to IEEE Transaction on Knowledge and Data Engineering.

UCI. (2010). UCI Machine Learning Repository. Http://archive.ics.uci.edu/ml/datasets.html

WEKA. (2010). University of Waikato Web site in New Zeeland dedicated to WEKA tool. Http://www.cs.waikato.ac.nz/ml/WEKA

WEKA-UCI. (2010). Collection of datasets in the WEKA ARFF format. http://www.cs.waikato.ac.nz/~ml/WEKA/index_datasets.html.

Yamanishi, K., Takeuchi, J. I., Williams, G. and Milne, P. (2004). On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. Kluwer Academic Publishers. The Netherlands..

Yang, X., Latecki, L.J. and Pokrajac, D. (2009). Outlier Detection with Globally Optimal Exemplar-Based GMM. *SIAM Int. Conf. on Data Mining (SDM09)* Sparks, Nevada, April/May, 2009.

Zhang, J. and Zulkernine, M. (2005). Network Intrusion Detection Systems Using Random Forests. MSc Thesis, School of Computing, Queen's University, Kingston, Canada.