Imbalanced Classification using Genetically Optimized Cost Sensitive Classifiers

Todd Perry
School of Computing
University of Portsmouth
Portsmouth PO1 3HE, UK
Email: todd.perry@myport.ac.uk

Mohamed Bader-El-Den
School of Computing
University of Portsmouth
Portsmouth PO1 3HE, UK
Email: mohamed.bader@port.ac.uk

Steven Cooper
Seagate Technology
Langstone Road
Havant PO9 1SA, UK
Email: steven.cooper@seagate.com

***Abstract*—Classification is one of the most researched problems in machine learning, since the 1960s a myriad of different techniques have been proposed. The purpose of a classification algorithm, also known as a 'classifier', is to identify what class, or category an observation belongs to. In many real-world scenarios, datasets tend to suffer from class imbalance, where the number of observations belonging to one class greatly outnumbers that of the observations belonging to other classes. Class imbalance has been shown to hinder the performance of classifiers, and several techniques have been developed to improve the performance of imbalanced classifiers. Using a *cost matrix* is one such technique for dealing with class imbalance, however it requires a matrix to be either pre-defined, or manually optimized. This paper proposes an approach for automatically generating optimized cost matrices using a genetic algorithm. The genetic algorithm can generate matrices for classification problems with any number of classes, and is easy to tailor towards specific use-cases. The proposed approach is compared against unoptimized classifiers and alternative cost matrix optimization techniques using a variety of datasets. In addition to this, storage system failure prediction datasets are provided by Seagate UK, the potential of these datasets is investigated.**

I. Introduction

As processing power and storage become cheaper, machine learning is quickly making it's way from a mathematical curiosity to a powerful decision making tool. Some of the more well known applications of machine learning in the real-world include targeted advertising [1] and email spam filters [5]. Machine learning is appealing to businesses as it allows problems to be solved that would previously be infeasible, however it can rarely solve these problems correctly every time. Classification is an example of a machine learning problem, classifiers attempt to identify an unseen object as part of a certain category, or class. In order to do this, classifiers must be trained using a dataset, made up of a number of records, or instances. Each instance of a dataset has many attributes and a class, classifiers learn from the dataset and build a model that can be used to predict a class based on the attributes alone. A classification problem is said to suffer from 'class imbalance' when the number of instances belonging to one class outnumbers that of any other class(es). Class imbalance can be measured using an imbalance ratio, for example in a scenario where there are four times as many examples of one class than another, the imbalance ratio will be 4:1. When the imbalance ratio becomes larger, classifying becomes more problematic, as the minority class(es) become a smaller part of the dataset. Class imbalance has been shown to adversely effect the performance of many classification algorithms [17]. Several techniques have been developed for dealing with class imbalanced datasets [23][10][11], and either involve modifying the dataset (re-sampling) or making the classifier 'cost sensitive'. *Re-sampling* involves modifying an imbalanced dataset to change the imbalance ratio (majority class / minority class). There are two types of re-sampling: undersampling, and oversampling. Undersampling involves reducing the size of the majority class by removing records. Records can be removed at random, or pseudo-randomly [23]. Pseudo-random techniques involve running clustering algorithms on the majority class to find any clusters, a small number of records are then randomly picked from each cluster, to ensure that the majority class is still properly represented despite data loss. Oversampling involves increasing the size of the minority class with the insertion of synthetic data. Synthetic data can be inserted at random, or pseudo-randomly using algorithms such as SMOTE (Synthetic Minority Oversampling Technique) [10]. Oversampling can be problematic when the minority set is made up of a small number of instances, as the synthetic data can be poorly representative of the class.

Classifiers can be made cost sensitive with the use of a *cost matrix* [11]. Cost matrix M is defined as:

$$M = \begin{pmatrix} M_{11} & \cdots & M_{1N} \\ \vdots & \ddots & \vdots \\ M_{N1} & \cdots & M_{NN} \end{pmatrix} \quad (1)$$

Where N is the number of classes and M_{ij} is the cost of classifying an instance with actual class i as class j. This makes it possible to penalize classifiers more for misclassifying the minority class, which solves a common problem when working with class imbalance, where classifiers simply classify all instances as the majority class. The advantage of making classifiers cost sensitive is the fact that no data is removed from the dataset, and no potentially harmful synthetic data is added. However when using a cost matrix, the values of an optimal cost matrix are often unknown. The aim of this paper is to create and evaluate an algorithm for automatically producing an optimized cost matrix. The rest of the paper is structured as follows: Section II reviews existing work done on optimizing cost matrices. Section III contains several performance metrics that are commonly used to measure the performance of imbalanced classifiers. Section IV proposes a solution that uses a genetic algorithm to automatically optimize cost matrices. Section V introduces a case study that is investigated in this paper, and summarizes some datasets provided by Seagate UK. Section VI describes the experiments run on the proposed classifier, and shows the results, this is followed by a discussion in Section VII and the paper is concluded with Section VIII.

II. Existing Techniques For Cost Matrix Optimization

One problem with cost matrices is that in most cases the optimal cost matrix values are unknown. In most cost matrices, $M_{ij} = 0$ where $j = i$, so for binary classification problems manual optimization is simple, as only M_{21} and M_{12} need to be investigated. However for problems with a larger number of classes, optimization becomes a more laborious task, as the number of non-zero cost matrix cells is equal to $N^2 - N$. Krawczyk et al. [18] trained ensembles of cost sensitive decision trees. Their technique focused on binary classification, where the cost matrix is made up of 4 elements. They derived cost matrices by investigating the cost of misclassifying the minority class. ROC (Receiver Operation Characteristic) curve analysis was used to analyse matrix performance. Their results showed a link between the values in the optimal cost matrix and the imbalance ratio of a dataset. Results were compared against 6 other classifiers and the proposed algorithm outperformed all others in 9 out of 18 cases (6 datasets, each with 3 imbalance ratios). Genetic algorithms have been used to tune misclassification costs before [21], however as opposed to optimizing a cost matrix, a misclassification cost for each class was learned. Their algorithm encodes a cost matrix as a vector, where each element of the vector is the misclassification cost for one of the classes. In cost matrices produced by their algorithm, $M_{ij} = M_{ik}, \forall j,k \in \{1,..,N\}$ where $k \neq i$ and $j \neq i$. This is problematic as large portions of the possible solution space go unsearched. They evaluate the performance of their optimized classifier against two unoptimized classifiers, their results show the optimized classifier performs better across 3 datasets in terms of two common performance metrics for imbalanced classification: Geometric Mean and F-Measure. Cao et al [9] proposed a framework for improving the performance of SVM (Support Vector Machines) classifiers by optimizing a number of parameters. and among these parameters was the cost matrix. Like [18], this paper only focuses on binary classification, so the only value that was optimized was the misclassification cost of the minority class. PSO (Particle Swarm Optimization) is used to optimize the parameters. Their results showed their proposed technique performed significantly better than 5 other SVM variants on 6 out of 11 imbalanced binary classification problems. Another example of a cost matrix optimization technique that focuses on binary classification is [22]. Thai-Nghe et al. used grid search [16] to optimize the *CostRatio* - the cost of misclassifying the majority class over the cost of misclassifying the minority class. This technique of optimizing a *CostRatio* is akin to optimizing the misclassification of the minority class, as [9] [18] did. Their results show that in terms of Geometric Mean, their proposed technique outperforms or is equal to a selection of other classifiers in 13 of 18 datasets. Although a large amount of interest has been generated around optimizing cost matrices, to the authors knowledge, no works exist that optimize a cost matrix by learning a cost for each value in the matrix. Existing works either learn a single cost [16] [9] [18], or learn a misclassification cost for each class [21]. Learning a single cost is only suitable for binary classification, and learning a misclassification cost for each class avoids searching large parts of solution space.

III. Measuring The Performance Of Imbalanced Classifiers

Once a classifier has been trained, it can be evaluated by using a *test set*, which contains instances that have not yet been encountered by the classifier. Once the test set has been evaluated, it produces a confusion matrix, and many performance metrics can be described in terms of the confusion matrix. The confusion matrix C can be defined as:

$$C = \begin{pmatrix} C_{11} & \cdots & C_{1N} \\ \vdots & \ddots & \vdots \\ C_{N1} & \cdots & C_{NN} \end{pmatrix} \tag{2}$$

Where N is the number of classes, and matrix element C_{ij} is the number of instances belonging to class i that were classified as j. For a binary classification problem, the number of false positives can be defined as C_{12}, and the number of false negatives can be defined as C_{21}. Traditionally classification accuracy is used to measure classifier performance, the classification accuracy, a, can be defined as:

$$a = \frac{\sum_{i=1}^{N} C_{ii}}{\sum_{i=1}^{N} \sum_{j=1}^{N} C_{ij}} \tag{3}$$

The classification accuracy gives a good idea of classifier performance when the dataset is balanced, however when the dataset is unbalanced some problems emerge. For example, in a binary classification problem, if the majority class outnumbered the minority class 9:1, and all instances were classified as the majority class, the classifier would have an accuracy of 90%, despite 0% of the minority class being classified correctly. Geometric mean [4] is a useful measure of classifier performance. The geometric mean g is defined as:

$$g = \sqrt[N]{\prod \frac{C_{ii}}{\sum_{i=1}^{N} C_{ij}}} \tag{4}$$

The geometric mean is a good performance measure for imbalanced problems as the performance of each class is weighted equally, meaning that if any of the classes show poor performance, the geometric mean will be low. Two popular performance metrics for imbalanced problems are sensitivity s and precision p. The sensitivity s of class i is defined as:

$$s(i) = \frac{C_{ii}}{\sum_{j=1}^{N} C_{ij}} \tag{5}$$

The precision p of class i is defined as:

$$p(i) = \frac{C_{ii}}{\sum_{j=1}^{N} C_{ji}} \tag{6}$$

Where $1 \leq i \leq N$. Sensitivity describes the percentage of instances belonging to class i that were correctly classified, when $s(i) = 1$ all instances belonging to class i have been classified correctly. Precision describes the percentage of instances classified as class i that were correctly classified. When $p(i) = 1$, and i is the 'positive class', the false positive rate of the classifier will be 0, ie. all instances assigned to class i are correctly classified. For classification problems with more than 2 classes S and P need to be defined to handle multiple minority classes. This is assuming that there is a single majority class. S and P are defined as:

$$S = \frac{1}{(N-1)} \sum_{i=2}^{N} s(i) \tag{7}$$

$$P = \frac{1}{(N-1)} \sum_{i=2}^{N} p(i) \tag{8}$$

Where class 1 is the majority class, hence precision and sensitivity of class 1 is not taken into account. Sensitivity and precision can be combined using a metric known as the F-Measure [8]. The F-Measure F is defined as:

$$F = 2\Big(\frac{SP}{S+P}\Big) \tag{9}$$

IV. Proposed Solution

A. Algorithm Description

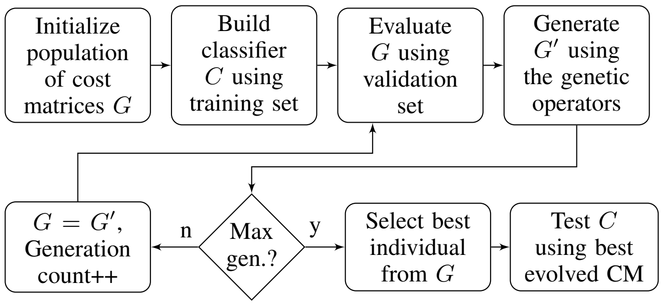
Initialize population of cost matrices G
Build classifier C using training set
Evaluate G using validation set
Generate G' using the genetic operators
$G = G'$, Generation count++
n
Max gen.?
y
Select best individual from G
Test C using best evolved CM

Fig. 1. A flowchart depecting the proposed algorithm

A GA (Genetic Algorithm) is employed to improve classification performance by exploring a solution space of possible cost matrices. The GA consists of a population of individuals (cost matrices), an inividual is expressed as a genome. The algorithm takes a dataset, and splits it into two sections, the 'training' set and the 'validation' set. Next, a population of randomly generated cost matrices is created. Each of these cost matrices is applied to a 'base classifier', the 'base classifier' type is supplied as a parameter. The classifiers are trained using the training set, and then their fitness is evaluated using the validation set. An empty population is then created, and filled with the offspring of the current population. If the maximum number of generations hasn't been reached, then this process repeats, but with the new population. If however, the maximum number of generations has been reached, The best individual (cost matrix) from the new population is applied to a base classifier, and this classifier is trained using the entire dataset (both train and validate). This classifier is then output by the algorithm. The whole process is illustrated in Figure 1 and Algorithm 1.

1) Genome: The genome (chromosome) is expressed as a vector of doubles, where the length is always N^2 with N being the number of classes. Each gene in the genome represents a different element of the cost matrix M, for example, a binary classification problem will have the genome: $\langle M_{11}, M_{12}, M_{21}, M_{22} \rangle$. When an individual is created, the genome is generated at random, genes associated with misclassification (M_{ij} where $i \neq j$) are initialized between 0 and 100, and genes associated with correct classification (M_{ij} where $i = j$) are initialized as 0.

2) Fitness Evaluation: Three fitness functions are proposed: *F-measure* (Equation 9), *Accuracy* (Equation 3) and *Geometric Mean* (Equation 4). These fitness functions are compared in Section VI.

3) Genetic Operators: Three types of genetic operator are used in the algorithm, *mutation*, *crossover* and *reproduction*. Genetic operators are how genetic information is passed from one generation to the next. Mutation involves selecting genes from an individual's genome and changing their value. As the genome is stored as a vector of doubles the mutation operator picks a random double from an individuals genome, and randomly generates a new double between 0 and 100 to replace it. Despite the fact matrix values associated with correct classification are initialized as 0, they can be mutated to become non-zero values. Crossover involves selecting two different individuals and 'mixing' their genomes, the type of crossover used in this algorithm is known as 'one-point crossover' and was selected due to the potentially small genome size. One point crossover involves picking a random crossover point in the genome, and swapping the genetic information of the two individuals after that point creating two new 'child' genomes. Reproduction selects an individual and copies it into the next generation. Individuals are selected using a technique known as tournament selection [14]. Tournament selection involves k individuals being picked from a population at random to be 'contenders'. The 'contender' with the highest fitness is selected as the winner. Mutation and reproduction requre one tournament to be run, whilst crossover requres two tournaments, as two parents are required. In the case of crossover, if both parents are the same, another tournament will be run to find a new parent.

V. Case Study: Storage System Failure Prediction

One real-world application of classification is the prediction of product failures before they happen. Failure prediction strategies offer a lot of opportunity for saving money in a business environment. Product failure prediction is a challenging problem because in such an environment datasets almost always suffer from class imbalance. As a case study, storage

Algorithm 1 Cost Matrix Optimization Algorithm

```
Split the dataset into ‘training’ and ‘validation’ sets
Randomly initialize the Cost Matrix population $G$.
for $i = 1$ to noGenerations do
  for $j = 1$ to noIndividuals do
    for $k = 1$ to noTrainingSet do
      train a classifier($C$) using record($k$) and matrix ($G_j$)
    end for
    for $k = 1$ to noValidationSet do
      evaluate classifier($G_j$) on record($k$)
    end for
    calculate fitness for individual($j$)
  end for
  create new empty generation $G'$
  for $m = 1$ to crossoverRate*100/2 do
    select two individual from $G$ based on fitness {parents}
    crossover the individuals creating two new individuals {children}
    insert new individuals in $G'$
  end for
  for $m = 1$ to mutationRate*100 do
    select one individual from $G$ based on fitness
    mutate the individual
    insert new individual in $G'$
  end for
  for $m = 1$ to reproductionRate*100 do
    select one individual from $G$ based on fitness
    insert that individual in $G'$
  end for
  $G = G'$
end for
build a ‘final’ classifier($C_F$) using the best individual from $G'$
train $M_F$ using the entire dataset
return $M_F$
```

system failure prediction datasets provided by Seagate UK are investigated in this paper. There is a wealth of literature on disk drive failure prediction using data collected whilst the drives are running, some of it very successful. Murray et al. [19] achieved a sensitivity of over 50% using SVM (Support Vector Machines) with a false positive rate of less than 1%. More recently Zhu et al. [24] used ANN (Artificial Neural Networks) and achieved a failure prediction rate of over 90% with a false positive rates of less than 1%. In spite of this, there has been little work done on predicting failures based on data gathered in manufacturing and little work has been published on predicting the failure of entire storage systems as opposed to individual disk drives.

A. *Enterprise Storage System Datasets*

Two datasets have provided by Seagate UK, the datasets were generated from Seagate manufacturing test data, they are made up of 18 months worth of data taken from one manufacturing plant, concerning a single product type. The product is an enterprise storage system, consisting of a disk drive array and two drive controllers. One dataset consists of data gathered at the controller level, and the other consists of data gathered at the drive level. The datasets contain attributes measured in the manufacturing test process, and a class, which corresponds to the component’s status in the field. The controller level dataset has 40,676 instances and 43 features, the features include the values of many environmental sensors on the controller. The dataset has 7 classes, either ‘No Defect’, or a category of field failure. The class distribution can be seen in Table I. The drive level dataset has 239,608 instances and 48 features. The features include performance measurements and data pulled from error counters on the drive. The dataset has 6 different classes, where the most common class ‘No Defect’ vastly outnumbers the other classes, which like the controller dataset represent different field failure categories. The class distribution can be seen in Table I.

TABLE I. Class Distribution of the Controller and Drive Datasets

Dataset	Class	Instances
Controller	No Defect	38770
	Connector - Pins Defective	1886
	Other	22
Drive	No Defect	236838
	Performance	1233
	G-List	782
	Corrupt Format	464
	Other	291

B. *Preprocessing*

Due to the way the data was initially extracted, a small number of duplicate records were present in the data, which have been removed during the preprocessing stage. In addition, to improve the performance of classification algorithms, the problems have been split up into binary classification problems. Each of the binary classification problems contain the instances concerning one of the defects, as well as all of the ‘No Defect’ instances. This reduces the probability of an instance being misclassified by reducing the possible number of classes it could fall into.

C. *Experiments On Failure Prediction Datasets*

To Determine which of the many datasets are suitable for failure prediction, some experiments were run. To compare the performance of different classifiers, 3 binary classification datasets were used: ‘Controller_DefectiveConnectors’, ‘Drive_Performance’ and ‘Drive_GList’, each one comprised of ‘No Defect’ records and records belonging to their respective failure category. The other datasets were not considered as they have a very small number of instances concerning the ‘failure’ class, this makes it hard for meaningful classifiers to be trained. The ‘DefectiveConnectors’ failure corresponds to one specific connector on the storage system being faulty. ‘Performance’ failures correspond to all performance related drive problems. The ‘GList’ failures correspond to drives with a very high number of G-List entries. A G-List entry occurs every time a ‘bad’ sector is discovered on the drive media, it is normal for a small number of G-List entries to occur, but a large number could imply the drive is faulty or damaged. *Naive Bayes* [20], *Bayesian Networks* [13], *CART* (Classification And Regression Tree) [7] and *Random Forests* [6] were evaluated on these datasets, and the results were compared. The *Random Forest* classifier used 9 trees, as this gave good results, but also had a reasonable training time. All experiments used

10 fold cross validation and the results shown are the mean of 5 models. Each model used a different random seed for generating cross validation partitions. The *precision*, *sensitivity* and *F-Measure* were measured, the results are displayed in Table II.

TABLE II. COMPARISON OF CLASSIFIERS USING CONTROLLER DATASET

Dataset	Classifier	Prec	Sens	FM
DefectiveConnectors	Naive Bayes	0.092	0.642	0.161
DefectiveConnectors	Bayesian Network	0.273	0.347	0.305
DefectiveConnectors	CART	0.414	0.367	0.386
DefectiveConnectors	Random Forest	0.883	0.456	0.601
Drive_Performance	Naive Bayes	0.008	0.082	0.016
Drive_Performance	Bayesian Network	0.014	0.005	0.008
Drive_Performance	CART	0.000	0.000	0.000
Drive_Performance	Random Forest	0.000	0.000	0.000
Drive_GList	Naive Bayes	0.007	0.305	0.014
Drive_GList	Bayesian Network	0.012	0.005	0.007
Drive_GList	CART	0.000	0.000	0.000
Drive_GList	Random Forest	0.000	0.000	0.000

Table II shows the results of the initial experiments on the failure prediction datasets. The controller dataset yields far better results than the drive datasets. These results show that the drive datasets are not suitable for failure prediction, as a high precision is required for this. The highest precision measured on either of the drive datasets is 0.014, this would imply a very high false positive rate. Because of this, the only Seagate dataset that will be investigated further is the defective connectors dataset. All three of the Seagate datasets suffer from class imbalance. The imbalance ratio of the controller dataset is around 20:1, whilst the imbalance ratios of the Drive_Performance and Drive_GList datasets are 192:1 and 302:1 respectively. One approach for improving classifier performance in the case of an imbalanced dataset is using a cost matrix [11]. Cost matrices have advantages over re-sampling methods as no data is lost and no synthetic data is added. In the case of a binary classification problem, the cost matrix can be defined as:

$$M = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \tag{10}$$

Assuming class 1 is the majority class, and class 2 is the minority class, as M_{21}/M_{12} increases, missclassifying the minority class becomes more and more costly, thus penalizing classifiers that classify most, or all instances as the majority class. Using 'Controller_DefectiveConnectors' as a benchmark, the classifiers used earlier in the paper were made cost sensitive, and changing the value of M_{21}/M_{12} was investigated.

Figure 2 shows how changing the cost matrix affects the performance of classifiers on the Controller_DefectiveConnectors dataset. Different classifiers appear to react differently to the cost matrix being changed. *Bayesian Networks*, *CART* and *Random Forests* appear to follow one trend, where the F-Measure improves as M_{21}/M_{12} is increased. *Naive Bayes* however behaves differently, the F-Measure imrpoves slowly as M_{21}/M_{12} decreases. In section VI the cost matrix will be optimized using the proposed solution and the performance of the optimized classifiers will be evaluated against their unoptimized counterparts using both the Seagate controler dataset, and publicly available datasets.

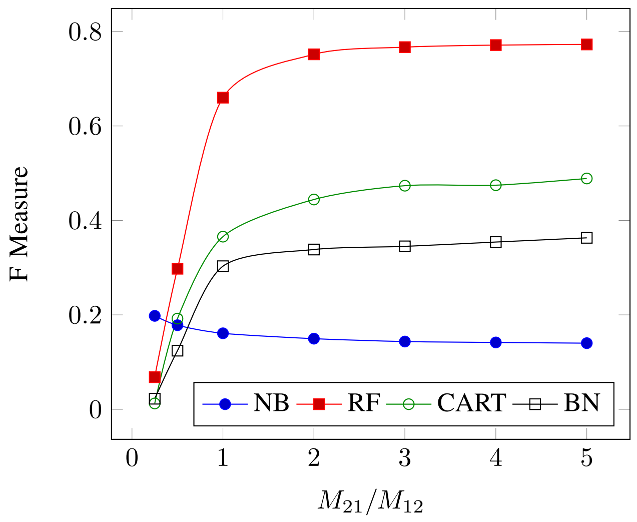

Fig. 2. F-measure of classifiers vs M_{21}/M_{12}. All classifiers showed an improved F-Measure for values of M_{21}/M_{12} that are not 1

VI. INVESTIGATON

A. *Experimental Setup*

Three experiments are run, the first experiment is designed to test the proposed algorithms effectiveness in a range of different scenarios. Seven different datasets are tested, the controller dataset extracted from the Seagate data, and six publicly available datasets from the KEEL repository [2]. Classifiers are evaluated using a 60/40 train/test set split. Genetic parameters can be seen in Table III. The parameters were chosen based on the time taken for the algorithm to complete and the rate at which the population looses diversity.

TABLE III. GENETIC PARAMETERS

Parameter	Expr 1	Expr 2	Expr 3
Mutation Rate	20%	20%	20%
Crossover Rate	60%	60%	60%
Reproduction Rate	20%	20%	20%
Population Size	30	80	80
Number Of Generations	12	50	50
train/validate split	70/30	70/30	70/30
Tournament Size	2	2	2

A Breakdown of the datasets can be seen in able IV. The imbalance ratio is given as: number of instances in majority class / number of instances in minority class.

TABLE IV. DATASET BREAKDOWN

Dataset	#Instances	#Features	#Classes	Imbalance Ratio
page-blocks0	5472	10	2	8.78
thyroid	7200	21	3	40.16
magic	19020	10	2	1.84
shuttle	57999	9	7	4558.6
coil2000	9822	85	2	15.76
letter	20000	16	26	1.14
Controller	40657	43	2	20.56

The second experiment is designed to show how effective the proposed algorithm is at different levels of class imbalance. The 6 datasets used in [18] were taken from the KEEL repository [2]. Random undersampling was run to achieve different levels of class imbalance (1:10, 1:25 and 1:50). A breakdown of the datasets can be seen in table V.

TABLE V. DATASET BREAKDOWN

Dataset	#Instances	#Features	#Classes
page-blocks0	5472	10	2
pima	768	8	2
segment0	2308	19	2
shuttle-c0-vs-c4	1829	9	2
vehicle2	846	18	2
yeast1	1484	8	2

In [18], Krawczyk et al. derive an optimal value of M_{21}/M_{12} for each of the datasets at each of the imbalance levels using ROC curve analysis. The final experiment compares ROC curve analysis to the proposed solution. All experiments were written using the *Weka Data Mining Software* [15].

B. Experimental Results

1) Experiment One: The target in this experiment is to evaluate the effectiveness of the evolved cost matrix in classifying imbalanced datasets. The performance of each classifier method is evaluated with and without cost matrix. The results in the NCM column are collected from a classifier with no cost matrix and the results in the other 3 columns were obtained using the proposed algorithm with the different fitness functions. The CM_A column contains data collected using the *Accuracy* fitness function, the CM_F contains data collected using the *F-Measure* fitness function, and the CM_G column contains data collected using the *Geometric Mean* fitness function. The results in the NCM column were collected from a classifier with no cost matrix. Each method is evaluated using F-Measure (FM), Precision (Perc) and Sensitivity (Sen). 'Std' corresponds to the standard deviation in F-Measure. The results are shown in Table VI. The results shown are the mean of 10 runs.

2) Experiment Two: This experiment is designed to investigate the effectiveness of the proposed algorithm at different levels of class imbalance. The algorithm is run using *Naive Bayes* as it's base classifier, with a 60/40 train/test split and *F-Measure* is used as the fitness function. The *Naive Bayes/F-Measure* combination is used due it's consistent improvement over the NCM classifier in experiment 1. The results are compared against an unoptimized Naive Bayes classifier, and an ensemble of *Naive Bayes* classifiers using *AdaBoost* [12]. Results can be seen in Table VII, all performance metrics shown are the mean of 10 runs.

3) Experiment 3: The final experiment compares the proposed genetic algorithm to ROC curve optimization, which is used in [18]. The ROC curve optimization uses *CART* as base classifier and modifies the value of M_{21}. Values between 5 and 100 are used, with a step size of 5. For a fair comparison, the genetic algorithm has been modified in this experiment to use the area under the ROC curve as a fitness function. The genetic algorithm uses *CART* as a base classifier. The results are also compared with an unoptimized *CART* classifier, with no cost matrix used. *CART* is used as the classifier used in [18] is based on decision trees. Results under the CM_{GA} correspond to results collected using the proposed algorithm. Results under the CM_{ROC} correspond to results collected using ROC curve optimization. The results can be seen in Table VIII. All performance metrics shown in this experiment are the mean of 10 runs.

C. Analysis Of Results

In experiment 1, the optimized *Naive Bayes* classifier shows the best performance gain on the 'magic' and 'page blocks' datasets, both of which have a fairly low number of attributes and are binary classification problems. Cost matrix optimization had little effect on the 'letter' and 'coil2000' datasets, 'letter' is balanced so a cost matrix shouldn't change classification performance much, however 'coil2000' is a binary classification problem. What separates 'coil2000' from the other binary classification problems is it's large number of attributes. The most effective fitness function appears to be *F-Measure*, which outperforms NCM in all cases. The other fitness functions (*Geometric Mean* and *Accuracy*) perform inconsistently. In contrast to *Naive Bayes*, *Bayesian Networks* show the best performance gain on the binary classification datasets with a large numbers of features ('coil2000' and the Seagate 'controller' dataset). The *F-Measure* fitness function does not perform as consistently here as it did with *Naive Bayes*, although in most cases it outperforms the NCM classifier. *Geometric Mean* is the most consistent fitness function here. Like *Bayesian Networks*, *CART* shows large performance gains on the 'coil2000' and 'controller' datasets. The *Accuracy* fitness functions performs very poorly, with the NCM classifier outperforming it in 6 of the 7 datasets. The other two fitness functions perform better, both of them outperform the NCM classifier in 5 of the 7 datasets. The *Random Forests* perform very well on the datasets, specifically the optimized *Random Forest* using *F-Measure* as a fitness function. *F-Measure* always performs better than, or on par with the NCM classifier. Like *Bayesian Networks* and *CART*, *Random Forests* see the best performance gains on the 'coil2000' and 'controller' datasets. *Random Forest* optimized using *F-Measure* is an extremely competitive classifier, it outperforms all 15 other configurations (4 classifiers * 4 cost matrix types) in 3 of the 7 datasets ('page blocks', 'magic' and 'controller').

From the results in experiment 2, it is obvious that in most cases performance deteriorates as the imbalance ratio is increased. In most cases, the unoptimized and optimized classifiers deteriorate at a steady rate, with the proposed solution performing better. The proposed solutions performance relative to *Adaboost* appears to change from dataset to dataset, in some cases the proposed solution outperforms *Adaboost* by a large margin ('yeast'), but in other cases performs very poorly in comparison ('page blocks'). The datasets the proposed solution performs best on ('pima', 'yeast' and 'shuttle') have a low number of attributes compared to the ones it performs poorly on. These results could indicate that the proposed algorithm outperforms *Adaboost* in scenarios where the number of attributes is very small.

In the final experiment 18 separate datasets were investigated, in 11 out of 18 cases, the proposed solution performs at least as well as the ROC curve analysis. As well as this, 'brute force' optimization techniques like the ROC curve analysis quickly become infeasible when the number of classes rises above 2, this scalability problem does not affect genetic algorithms, so the proposed solution will scale well.

TABLE VI. RESULTS OBTAINED FROM EXPERIMENT 1

		NCM				CM_A				CM_F				CM_G			
	Dataset	FM	Std	Prec	Sen	FM	Std	Prec	Sen	FM	Std	Prec	Sen	FM	Std	FM	Sen
Naive Bayes	blocks	0.506	0.044	0.543	0.474	0.496	0.052	0.609	0.419	**0.557**	0.057	0.511	0.611	0.531	0.045	0.491	0.577
	thyroid	0.668	0.035	0.772	0.589	**0.678**	0.055	0.749	0.620	0.677	0.071	0.727	0.633	0.651	0.044	0.536	0.829
	magic	0.496	0.007	0.710	0.381	0.463	0.021	0.796	0.326	**0.609**	0.051	0.531	0.714	0.603	0.015	0.538	0.687
	shuttle	0.559	0.048	0.435	0.782	0.583	0.039	0.456	0.808	**0.591**	0.035	0.467	0.804	0.569	0.036	0.441	0.802
	coil	0.202	0.016	0.130	0.454	**0.206**	0.017	0.171	0.259	0.205	0.036	0.141	0.376	0.181	0.019	0.107	0.596
	letter	0.646	0.005	0.651	0.641	**0.648**	0.010	0.653	0.644	**0.648**	0.006	0.653	0.643	**0.648**	0.005	0.653	0.644
	conn	0.165	0.009	0.094	0.642	0.014	0.027	0.126	0.008	**0.216**	0.030	0.160	0.330	0.162	0.016	0.093	0.618
Bayes Net	blocks	0.768	0.022	0.721	0.821	**0.795**	0.049	0.735	0.865	0.791	0.029	0.715	0.884	0.774	0.049	0.671	0.914
	thyroid	0.936	0.024	0.924	0.949	**0.952**	0.015	0.930	0.976	0.937	0.019	0.917	0.958	0.948	0.015	0.922	0.976
	magic	0.721	0.013	0.806	0.653	0.724	0.036	0.816	0.650	**0.738**	0.050	0.712	0.765	0.737	0.023	0.694	0.785
	shuttle	**0.792**	0.094	0.775	0.809	0.745	0.089	0.740	0.751	0.760	0.073	0.750	0.771	0.790	0.073	0.798	0.783
	coil	0.147	0.038	0.243	0.105	0.013	0.040	0.026	0.008	**0.225**	0.049	0.166	0.349	0.194	0.020	0.116	0.601
	letter	**0.754**	0.003	0.760	0.748	0.749	0.006	0.756	0.743	0.753	0.003	0.760	0.747	**0.754**	0.005	0.760	0.748
	conn	0.268	0.021	0.267	0.269	0.007	0.015	0.048	0.004	**0.344**	0.019	0.248	0.561	0.320	0.054	0.212	0.653
CART	blocks	0.820	0.006	0.826	0.814	0.821	0.020	0.801	0.843	**0.827**	0.023	0.807	0.848	0.794	0.049	0.716	0.892
	thyroid	**0.968**	0.010	0.967	0.970	0.949	0.021	0.944	0.954	0.958	0.026	0.940	0.976	0.960	0.025	0.946	0.974
	magic	0.739	0.006	0.743	0.735	0.738	0.019	0.822	0.669	**0.748**	0.017	0.780	0.719	0.745	0.031	0.745	0.746
	shuttle	0.938	0.015	0.980	0.899	0.936	0.012	0.973	0.902	0.897	0.080	0.912	0.882	**0.951**	0.031	0.964	0.938
	coil	0.109	0.024	0.251	0.070	0.003	0.005	0.042	0.001	0.163	0.009	0.111	0.304	**0.181**	0.011	0.121	0.361
	letter	0.807	0.008	0.808	0.806	0.804	0.004	0.805	0.803	**0.811**	0.002	0.812	0.811	0.804	0.012	0.806	0.803
	conn	0.278	0.029	0.355	0.229	0.000	0.000	0.000	0.000	**0.372**	0.041	0.288	0.525	0.341	0.079	0.232	0.643
Random Forest	blocks	0.858	0.014	0.873	0.843	**0.868**	0.031	0.865	0.872	**0.868**	0.037	0.866	0.871	0.831	0.040	0.747	0.936
	thyroid	0.954	0.023	0.948	0.960	**0.955**	0.018	0.942	0.968	**0.955**	0.018	0.934	0.977	0.952	0.023	0.933	0.972
	magic	0.795	0.006	0.846	0.749	0.792	0.017	0.870	0.726	**0.797**	0.026	0.815	0.779	0.793	0.022	0.795	0.791
	shuttle	0.908	0.073	0.954	0.867	0.886	0.061	0.938	0.840	0.916	0.048	0.960	0.875	**0.924**	0.071	0.968	0.883
	coil	0.102	0.025	0.187	0.070	0.004	0.008	0.063	0.002	0.187	0.032	0.143	0.270	**0.197**	0.064	0.126	0.452
	letter	**0.920**	0.004	0.921	0.920	0.918	0.002	0.919	0.918	0.919	0.005	0.921	0.918	0.918	0.003	0.919	0.917
	conn	0.449	0.020	0.817	0.310	0.595	0.038	0.769	0.485	**0.609**	0.029	0.727	0.524	0.508	0.136	0.406	0.680

TABLE VII. RESULTS OBTAINED FROM EXPERIMENT 2

		NCM		CM_F		AdaBoost	
Dataset	IR	FM	Std	FM	Std	FM	Std
bloc	10	0.486	0.036	0.523	0.033	**0.558**	0.090
bloc	25	0.342	0.060	0.350	0.035	**0.480**	0.068
bloc	50	0.278	0.069	0.287	0.097	**0.383**	0.083
pima	10	0.292	0.072	**0.305**	0.080	0.268	0.094
pima	25	0.120	0.122	**0.127**	0.076	0.090	0.100
pima	50	0.084	0.059	**0.094**	0.055	0.081	0.075
segm	10	0.549	0.026	0.619	0.022	**0.650**	0.107
segm	25	0.301	0.041	0.399	0.069	**0.465**	0.256
segm	50	0.217	0.061	0.269	0.068	**0.398**	0.249
shut	10	0.998	0.004	**1.000**	0.000	0.997	0.005
shut	25	**0.998**	0.005	**0.998**	0.005	0.994	0.010
shut	50	**0.997**	0.010	0.996	0.013	0.993	0.015
vehi	10	0.427	0.135	0.439	0.086	**0.565**	0.123
vehi	25	0.304	0.165	0.247	0.109	**0.395**	0.109
vehi	50	0.059	0.104	0.061	0.109	**0.145**	0.113
yeas	10	0.214	0.057	**0.322**	0.051	0.159	0.047
yeas	25	0.092	0.067	**0.113**	0.044	0.058	0.042
yeas	50	0.175	0.077	**0.186**	0.073	0.013	0.040

TABLE VIII. RESULTS OBTAINED FROM EXPERIMENT 3

		NCM		CM_{GA}		CM_{ROC}	
Dataset	IR	FM	Std	FM	Std	FM	Std
bloc	10	0.825	0.026	**0.826**	0.023	0.816	0.016
bloc	25	0.715	0.034	**0.737**	0.040	0.732	0.029
bloc	50	0.564	0.084	**0.610**	0.054	0.569	0.044
pima	10	0.185	0.158	**0.311**	0.093	0.288	0.064
pima	25	0.024	0.058	0.106	0.061	**0.128**	0.054
pima	50	0.000	0.000	0.105	0.098	**0.135**	0.084
segm	10	0.939	0.029	**0.959**	0.019	0.938	0.029
segm	25	**0.911**	0.044	0.908	0.066	0.889	0.046
segm	50	0.815	0.153	0.874	0.063	**0.900**	0.050
shut	10	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
shut	25	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
shut	50	**1.000**	0.000	**1.000**	0.000	**1.000**	0.000
vehi	10	0.685	0.119	**0.765**	0.088	0.701	0.100
vehi	25	0.235	0.244	**0.553**	0.106	0.537	0.134
vehi	50	0.133	0.161	0.314	0.126	**0.326**	0.164
yeas	10	0.127	0.067	0.271	0.034	**0.279**	0.045
yeas	25	0.047	0.073	0.137	0.066	**0.150**	0.059
yeas	50	0.015	0.038	**0.099**	0.064	0.088	0.066

TABLE IX. CONTROLLER DATASET RESULTS BREAKDOWN

Classifier	Mean s	Mean p	Best s	Best p
Naive Bayes (NCM)	0.642	0.094	0.649	0.106
Naive Bayes (CM_F)	0.330	0.160	0.342	0.167
Bayesian Network (NCM)	0.269	0.267	0.330	0.283
Bayesian Network (CM_F)	0.561	0.248	0.590	0.258
CART (NCM)	0.229	0.355	0.247	0.402
CART (CM_F)	0.525	0.288	0.528	0.300
Random Forest (NCM)	0.310	0.817	0.332	0.869
Random Forest (CM_F)	0.524	0.727	0.537	0.740

D. Analysis Of Seagate Datasets

Of the three Seagate datasets, only the Controller dataset appears to show a connection between the manufacturing data and field failures. A detailed breakdown of the results from this dataset can be seen in table IX. Unoptimized classifiers are compared against the proposed solution using the best fitness function (which in all cases is F-Measure). These results show a trade-off between sensitivity and precision, such a trade-off is common in classification problems [3], the optimization algorithm finds the best trade-off between the two, and that's how the overall performance is improved. No previous work has investigated a similar failure prediction problem, so there is nothing to compare these results with, however *Random Forests* appear to perform the best, with the untopimized classifier having a precision of 0.869 in the best case. The optimized algorithm has a lower precision, however the sensitivity is higher, meaning a larger number of failures would be correctly predicted, at the cost of a higher false alarm rate. As for the other two Seagate datasets, both concerning the prediction of drive failures, they appear to show very poor classification performance (shown in table II). This implies either there is little or no connection between the data recorded in manufacturing test and the 'field status' of the drive or the algorithms discussed in this paper are not suitable for solving this problem.

VII. Discussion

Experiment 1 shows that genetically optimizing the cost matrix has potential to improve classification performance. The results from experiment 1 along with Figure 2 show that different classifiers are effected differently when a cost matrix is applied. Another interesting finding is that using *F-Measure* as the fitness function doesn't always produce the highest *F-Measure* in the results, which is counter-intuitive. Experiment 2 shows that the performance of the proposed algorithm deteriorates as the imbalance ratio increases. The experiment also shows that in some cases, the proposed algorithm can outperform *Adaboost* [12]. Experiment 3 compares the proposed algorithm to 'manual' optimization using ROC curve analysis [18]. It was shown that in the majority of cases, the proposed algorithm performs at least as well as ROC curve analysis. One of the key advantages of the proposed solution is how easy it is to modify for different classifiers or fitness functions, this allows the proposed solution to be modified to work in many different use-cases. However, as the solution is based around a genetic algorithm and fitness evaluation involves training classifiers, a very large number of classifiers must be trained. Because of this the training time for the proposed solution is far longer than that of a single base classifier. As well as this the proposed solution has many additional parameters on top of the base classifiers parameters, which may require optimization. As the number of classes (N) in a classification problem grows, number of cost matrix elements, and therefore the genome size is equal to N^2. The dimensionality of solution space rapidly increases as the number of classes increases, meaning that larger populations are required to search the solution space effectively. Future research could focus on finding a method of storing a cost matrix as genome with a smaller size, whilst still retaining all of the information. As well as this, the way different classifiers react to cost matrices warrants an investigation, learning why this happens could be the key to an improved approach to cost matrix optimixation.

VIII. Conclusion

This paper demonstrates that the performance of classifiers suffering from class imbalance can be improved with the use of a cost matrix, and that an effective cost matrix can be derived using a heuristic search algorithm. One such heuristic technique is proposed by this paper and uses a genetic algorithm for cost matrix optimization. The algorithm takes a classifier and derives an optimized cost matrix, optimized classifiers are compared against their unoptimized counterparts as well as boosting algorithms, both cases show promising results. It is important to improve the performance of imbalanced classification because it affects many real-world problems, once such real-world example is investigated in this paper: storage system failure prediction. Classification algorithms were run on three datasets provided by Seagate UK, and one of the datasets showed good potential for failure prediction, accurately predicting over 50% failures, with a precision of over 70%.

References

[1] Charu Chandra Aggarwal, Joel Leonard Wolf, and Philip Shi-lung Yu. Method for targeted advertising on the web based on accumulated self-learning data, clustering users and semantic node graph techniques, March 30 2004. US Patent 6,714,975.

[2] J Alcalá, A Fernández, J Luengo, J Derrac, S García, L Sánchez, and F Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17:255–287, 2010.

[3] Sergio A Alvarez. An exact analytical relation among recall, precision, and classification accuracy in information retrieval. *Boston College, Boston, Technical Report BCCS-02-01*, pages 1–22, 2002.

[4] Ricardo Barandela, José Salvador Sánchez, Vicente Garcıa, and Edgar Rangel. Strategies for learning in class imbalance problems. *Pattern Recognition*, 36(3):849–851, 2003.

[5] Enrico Blanzieri and Anton Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, 2008.

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[8] Michael K. Buckland and Fredric C. Gey. The relationship between recall and precision. 1994.

[9] Peng Cao, Dazhe Zhao, and Osmar Zaiane. An optimized cost-sensitive svm for imbalanced data learning. In *Advances in Knowledge Discovery and Data Mining*, pages 280–292. Springer, 2013.

[10] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *arXiv preprint arXiv:1106.1813*, 2011.

[11] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Citeseer, 2001.

[12] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[13] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[14] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[16] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[17] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[18] Bartosz Krawczyk, Michał Woźniak, and Gerald Schaefer. Cost-sensitive decision tree ensembles for effective imbalanced classification. *Applied Soft Computing*, 14:554–562, 2014.

[19] Joseph F Murray, Gordon F Hughes, and Kenneth Kreutz-Delgado. Machine learning methods for predicting failures in hard drives: A multiple-instance application. In *Journal of Machine Learning Research*, pages 783–816, 2005.

[20] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[21] Yanmin Sun, Mohamed S Kamel, Yang Wang, et al. Boosting for learning multiple classes with imbalanced class distribution. In *ICDM*, volume 6, pages 592–602, 2006.

[22] Nguyen Thai-Nghe, Zeno Gantner, and Lars Schmidt-Thieme. Cost-sensitive learning methods for imbalanced data. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[23] Show-Jane Yen and Yue-Shi Lee. Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications*, 36(3):5718–5727, 2009.

[24] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. Proactive drive failure prediction for large scale storage systems. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pages 1–5. IEEE, 2013.