Data Science for Class Imbalanced and Cost-Sensitive Data and its Application to Software Defect Prediction

Author: Michael J. SIERS

Principal Supervisor: A/Prof. Md Zahidul ISLAM

Co-Supervisor: Prof. Terry BOSSOMAIER

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Computing and Mathematics

April 21, 2019
“The more that you read, the more things that you’ll know. The more that you learn, the more places you’ll go.”

Dr. Seuss
Abstract

Faculty of Business, Justice and Behavioural Sciences
School of Computing and Mathematics

Doctor of Philosophy

Data Science for Class Imbalanced and Cost-Sensitive Data and its Application
to Software Defect Prediction

by Michael J. SiERS

Class imbalance and cost-sensitivity are two prominent challenges in classification. The overwhelming majority of techniques which address these issues only focus on predictive performance rather than suitability for knowledge discovery. This thesis focuses on addressing both issues. This thesis proposes the design for an approach with four important characteristics. Firstly, a cost-sensitive decision forest is generated which avoids the negative effects of class imbalance. Secondly, the forest is generated using the entirety of the original training dataset which means that the knowledge it contains directly matches the original data. Thirdly, a clear process is proposed which automatically extracts, ranks, and values the forest’s discovered knowledge. Lastly, the resulting classifier achieves competitive performance compared to several existing techniques. The knowledge discovery approach is demonstrated by discovering patterns in software bugs present in several NASA programs (National Aeronautics and Space Administration). The conceptual design of a tool for real-time integration of the proposed techniques into the software development process is also presented at the end of this thesis.
Publications Arising from this Thesis

During the candidature of this PhD, a number of peer-reviewed articles were produced. They are listed here for reference.

  URL: https://doi.org/10.1016/j.ins.2018.05.035.

  URL: https://doi.org/10.1007/978-3-319-49586-6_11

  URL: https://goo.gl/jF5pdJ

  URL: https://doi.org/10.1007/978-3-319-26350-2_46


Publications Produced by Candidate Outside the Scope of this Thesis

During and prior to PhD candidature, three publications were produced by the candidate which are outside the scope of this thesis. They are listed here for reference.

  URL: http://ausdm.org/ausdm16.pdf

  URL: https://doi.org/10.1016/j.is.2015.02.006.

  URL: https://doi.org/10.1007/978-3-319-13560-1_80
Acknowledgements

My supervisor A/Prof Md Zahidul Islam is endlessly committed to his students’ growth. His endless enthusiasm for data mining is something that has been transferred to each of his students including myself. I am ever grateful for the wisdom he imparted to me during my candidature. Thank you Zahid.

There was no bigger inspiration to complete this work than my other half Laura. She has kept me sane when the work has been tough, and provided me with more work-life balance advice than I thought possible.

My parents and sisters consistently encouraged my work which did wonders for my perseverance towards the completion of this thesis. I am also grateful to the members of my extended family for their encouragement and support.

I’m so grateful to be part of such a talented team of data professionals at Flight Centre. They have been so supportive of my efforts during this thesis’ revisions.

I am thankful to my friends who helped me to wind down after long days of research and writing. I will not forget the many movies seen, games played and dinners eaten together. Thank you Mitch, Jon, Tim, Garret, Hannah, Anthony, and Natalie.

Finally, I am thankful to my loyal Australian Shepherd Tinker who was always by my side during the writing of this thesis.
## Contents

Abstract iii

Publications Arising from this Thesis v

Publications Produced by Candidate Outisde the Scope of this Thesis vii

Acknowledgements ix

Declaration of Authorship xv

Part I Exordium 1

1 Introduction 3
   1.1 Research Questions ........................................... 4
   1.2 Overview of this Thesis ........................................ 7

2 Data Science: Core Concepts 11
   2.1 Classification .................................................... 12
      2.1.1 Decision Trees ............................................. 13
      2.1.2 K-Nearest-Neighbours Classifiers .......................... 14
      2.1.3 Ensembles .................................................. 15
      2.1.4 Decision Forests ............................................ 16
      2.1.5 Classifier Performance Evaluation .......................... 17
   2.2 Clustering .......................................................... 19
   2.3 Data Resampling .................................................. 20
   2.4 Supervised, unsupervised, and active learning ............ 21

3 Literature Review 23
3.1 Introduction ................................................. 23
3.2 The Fundamental Similarity: Bias Adjustment .................. 26
  3.2.1 Main Goal of Class Imbalance Research .................. 26
  3.2.2 Main Goal of Cost-Sensitive Classification Research ..... 28
  3.2.3 Common Goal of Class Imbalance and Cost-Sensitive Tech-
       niques: Bias Adjustment .................................. 30
  3.2.4 The Proposed Bias Adjustment Taxonomy ................ 30
  3.2.5 Cost-Sensitive Techniques Which Also Address Class Imbalance 32
3.3 Dataset Modification ....................................... 32
  3.3.1 Resampling ............................................ 34
    Resampling: Class Specific .................................. 35
    Resampling: All Classes ..................................... 43
  3.3.2 Feature Selection ....................................... 44
3.4 Algorithm Modification Properties ............................. 46
  3.4.1 Splitting Criteria Modification ........................... 46
  3.4.2 Wrapper Methods ....................................... 48
3.5 Direct Bias Adjustment ...................................... 50
  3.5.1 Class Level ............................................ 50
  3.5.2 Record Level .......................................... 51
  3.5.3 Model Level ............................................ 54
  3.5.4 Cluster Level .......................................... 56
3.6 Example of Inter-Field Learning ............................... 56
  3.6.1 Experimental Evaluation of the Proposed SCSRoulette .... 57
3.7 Summary of Experimental Results in the Surveyed Studies ..... 60
3.8 Identified gaps which motivate this thesis ..................... 61
  3.8.1 Lack of Delineation and Definition of Cost Types .......... 61
  3.8.2 Existing Class Imbalance Methods Are Not Designed for Knowl-
       edge Discovery ............................................. 63
  3.8.3 Unexplored ability of producing low cost classifiers by taking
       a unified approach ......................................... 64
3.9 Concluding Remarks ........................................ 65
Part II  Classification in Imbalanced Data: Cost-Insensitive  67

4  Standoff Balancing: Classification in Class Imbalanced Datasets  69
   4.1  Introduction .................................................. 69
   4.2  Main Contributions ......................................... 71
   4.3  Lanchester’s Laws ......................................... 72
   4.4  The Proposed Method ...................................... 72
      4.4.1  Analogy Between Class Imbalance and War ............ 72
      4.4.2  Standoff-Balancing .................................. 73
   4.5  Experimental Results ....................................... 76
      4.5.1  Sensitivity to Outliers ................................. 78
      4.5.2  Sensitivity to High Dimensions ....................... 79
   4.6  Conclusion .................................................. 80

5  RBClust: Fast Discovery of Class Specific Clusters  83
   5.1  Introduction .................................................. 84
   5.2  Main Contributions ......................................... 85
   5.3  Related Work ................................................ 85
   5.4  The Proposed Method: RBClust ............................ 87
   5.5  Experiments .................................................. 90
      5.5.1  Setup ..................................................... 90
      5.5.2  Discussion of Results ................................ 90
      5.5.3  Time improvement to Standoff ....................... 92
      5.5.4  Discussion about setting $\theta$ ....................... 93
   5.6  Conclusion .................................................. 94

Part III  Classification in Imbalanced Data: Cost-Sensitive  97

6  BCF: Balanced Costs Framework  99
   6.1  Introduction .................................................. 99
   6.2  BCF: A cost-sensitive classification technique ............ 102
   6.3  Experiments .................................................. 107
      6.3.1  Experimental Setup ................................. 107
6.3.2 Note About Previously Used Datasets ........................................ 109
6.3.3 Results and Discussion .......................................................... 109
6.3.4 Cost Spread Analysis .............................................................. 110
Which algorithm most commonly achieves the lowest total cost on average? ......................................................... 114
Which algorithms have the least/most consistent performances on average? .......................................................... 114
6.3.5 Variance Analysis ................................................................. 118
6.4 Conclusions ............................................................................ 118

7 Cost-Sensitive Knowledge Discovery Using The Proposed Methods ................................................................. 123
7.1 Introduction ............................................................................. 124
7.1.1 SibCost: A Cost-Sensitive Rule Set Interestingness Measure .... 124
7.1.2 SCORE: An Algorithm for Automatic Extraction of the Most Interesting Rule Sets from Decision Trees and Forests .... 126
7.1.3 Breadth-First-Mining: A process for cost-sensitive knowledge discovery ......................................................... 128
7.1.4 Experimental Evaluation of SCORE ......................................... 129
7.2 Knowledge Discovered by Applying the Proposed Methods to NASA Software .................................................. 130
7.2.1 The Dataset: MDP’ ............................................................... 130
7.2.2 Extracted Knowledge ............................................................ 132
7.3 Conclusions ............................................................................ 136

Part IV Denouement .................................................................... 137

8 Discussion .................................................................................. 139
8.1 IDE Integration Concept .......................................................... 139
8.1.1 Back-End Architecture ....................................................... 140
8.1.2 Front-End Interface ............................................................. 141
8.2 Challenges with implementing the described concept ............ 147
8.2.1 Converting rule sets to everyday language ......................... 147
8.2.2 Meaningfulness of expected cost ....................................... 147
Declaration of Authorship

I hereby declare that this submission is my own work and to the best of my knowledge and belief, understand that it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at Charles Sturt University or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by colleagues with whom I have worked at Charles Sturt University or elsewhere during my candidature is fully acknowledged. I agree that this thesis be accessible for the purpose of study and research in accordance with normal conditions established by the Executive Director, Library Services, Charles Sturt University or nominee, for the care, loan and reproduction of thesis, subject to confidentiality provisions as approved by the University.

Name: Michael J Siers

Signed: 21/04/2019

Date: \\text{\textasciitilde}Siers
List of Figures

2.1 A popularity comparison between the terms data mining and data science. Data source: Google Trends. ................................. 11

2.2 A decision tree classifier trained from the example data in Table 2.1. ................................. 13

2.3 An illustration of how a decision forest classifies a new unseen record. This general process is the same as for ensemble classifiers. ....................... 17

2.4 A visual example of what a clustering algorithm does. In the clustered dataset, a broken green circle denotes a discovered cluster. ....................... 20

3.1 The proposed taxonomy for the properties of bias adjustment techniques in decision trees. Properties of class imbalance treatment algorithms and cost-sensitive classification algorithms can be categorised using this taxonomy. ................................................................. 25

3.2 The AUC of classifiers before and after treating class imbalance. AUC is a measure of classification performance (the higher the better). A linear regression line is included in both figures. .................................................. 27

3.3 A cost-sensitive decision tree which predicts whether or not a C function contains bugs. Trained on data collected from source code data from a NASA satellite’s flight software. Algorithm: CSTree (Sheng et al., 2014). ................................................................. 29

3.4 SCSRoulette vs CSRoulette for Dataset 1. .................................................. 59

3.5 SCSRoulette vs CSRoulette for Dataset 2. .................................................. 59

3.6 SCSRoulette vs CSRoulette for Dataset 3. .................................................. 60

3.7 SCSRoulette vs CSRoulette for Dataset 4. .................................................. 60

3.8 Average Results for SCSRoulette vs CSRoulette. .................................................. 60

4.1 The major steps in the proposed Standoff Balancing algorithm. ....................... 71
4.2 AUC Comparison of the Methods (Higher the Better) visualised. Those who are reading this in black-and-white are referred to Table 4.3 which presents the same information but in further detail. .......................... 77

4.3 Standoff Improvement over Original vs Percent Minority .................. 78

4.4 Standoff’s AUC on the Pima datasets that have 0%, 5%, 10%, and 15% added outliers. ................................................................. 79

4.5 Standoff’s AUC on the Arcene datasets that have 50, 100, 150, and 200 attributes. ................................................................. 80

5.1 The major steps of Standoff Balancing proposed in Chapter 4. ........ 84

5.2 The major steps of Standoff Balancing where the clustering steps have been replaced by RBClust. ........................................... 84

5.3 Methods Comparison - Silhouette Coefficient visualised. Those who are reading this in black-and-white are referred to Table 5.1 which presents the same information but in further detail. ....................... 91

5.4 Time to Find all CSCs (ms) visualised. Those who are reading this in black-and-white are referred to Table 5.2 which presents the same information but in further detail. ................................. 92

5.5 The average execution time (seconds) of a fold in 10 cross-fold validation. Results shown are for Standoff with EM (proposed in Chapter 4) and Standoff with RBClust (proposed in this Chapter). visualised. Those who are reading this in black-and-white are referred to Table 5.3 which presents the same information but in further detail. ......................... 93

5.6 The time taken to execute on each dataset. Each point along the x-axis is the number of records within that dataset. ................................. 94

6.1 The proposed framework: BCF. .............................................. 102

6.2 Combined Cost-Matrices in CSForest, Domain Cost-Matrix in CSVoting 105

6.3 Total Cost Comparison for Different Weight Combinations ............... 106

6.4 How the 10-fold cross validation strategy is used to evaluate total cost for a given dataset. This is the procedure used to evaluate the compared techniques in Table 6.6. .............................................. 108
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>Total cost comparison (lower is better) visualised. Those who are reading this in black-and-white are referred to Table 6.6 which presents the same information but in further detail.</td>
<td>110</td>
</tr>
<tr>
<td>6.6</td>
<td>The minimum, average, and maximum results of the cost spread analysis for KC3. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.</td>
<td>111</td>
</tr>
<tr>
<td>6.7</td>
<td>The minimum, average, and maximum results of the cost spread analysis for MC2. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.</td>
<td>112</td>
</tr>
<tr>
<td>6.8</td>
<td>The minimum, average, and maximum results of the cost spread analysis for MW1. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.</td>
<td>112</td>
</tr>
<tr>
<td>6.9</td>
<td>The minimum, average, and maximum results of the cost spread analysis for CM1. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.</td>
<td>113</td>
</tr>
<tr>
<td>6.10</td>
<td>The cost spread results visualised. The technique with the lowest mean is written in bold and illustrated in blue.</td>
<td>113</td>
</tr>
<tr>
<td>6.11</td>
<td>The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised.</td>
<td>115</td>
</tr>
<tr>
<td>6.12</td>
<td>The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised.</td>
<td>116</td>
</tr>
<tr>
<td>6.13</td>
<td>The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised.</td>
<td>116</td>
</tr>
</tbody>
</table>
6.14 The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised. ........................................ 117

6.15 Variance of total cost over 10x10 crossfold validation for each method on dataset KC3'. ................................................................. 118

6.16 Variance of total cost over 10x10 crossfold validation for each method on dataset MC2'. ................................................................. 119

6.17 Variance of total cost over 10x10 crossfold validation for each method on dataset MW1'. ................................................................. 119

6.18 Variance of total cost over 10x10 crossfold validation for each method on dataset CM1'. ................................................................. 120

7.1 How Contributions 1, 2, 3, and BCF can be used together to discover cost-sensitive knowledge from a dataset. Inputs/Outputs are represented as orange parallelograms. Each blue rectangle represents one of the proposed contributions. ........................................ 124

7.2 A cost-sensitive tree generated by BCF which classifies modules as defective or non-defective. ................................................................. 125

7.3 The proposed process for knowledge discovery which uses the proposed algorithms: Breadth-First-Mining. ........................................ 128

7.4 Average SibCost for the top 5 most interesting rule sets (higher the better). ................................................................. 130

7.5 Rule set for Insight 1. ................................................................. 133

7.6 Volume vs expected function cost. .................................................... 133

7.7 Rule set for Insight 2. ................................................................. 133

7.8 Effort vs function cost. ................................................................. 133

7.9 First rule set for Insight 3. ................................................................. 133

7.10 Second rule set for Insight 3. ................................................................. 134

7.11 Executable lines vs function cost. .................................................... 134

7.12 Rule set for Insight 4. ................................................................. 134

7.13 A box plot showing the distribution of Halstead’s estimated number of bugs for both defective modules and non-defective modules. ........................................ 135
7.14 An extended version of the rule set from Insight 1. 135

8.1 The back-end architecture of how the proposed work can be integrated into a software development environment. 140

8.2 The user has selected the function information tip for the function: ClassifyInstance. 143

8.3 The user has selected the function information tip for the function: BuildClassifier. 144

8.4 The user has requested an explanation of how the suggestion was calculated. 145

8.5 The user has opened the dialog which displays an ordered list of all functions and their expected costs. 146

9.1 AUC Comparison of the Methods (Higher the Better) visualised. Those who are reading this in black-and-white are referred to Table 9.1 which presents the same information but in further detail. 152

9.2 Methods Comparison - Silhouette Coefficient (The higher the better) - visualised. Those who are reading this in black-and-white are referred to Table 9.2 which presents the same information but in further detail. 153

9.3 Total cost comparison (lower is better) visualised. Those who are reading this in black-and-white are referred to Table 9.3 which presents the same information but in further detail. 156

9.4 Average SibCost for the top 5 most interesting rule sets (higher the better). 157

List of Tables

2.1 An example of historical data which can be used to train a classifier which can predict whether or not it will rain on any given day. 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>An example confusion matrix with two possible class values: Rain and No Rain.</td>
</tr>
<tr>
<td>3.1</td>
<td>The experiment settings used to generate the results shown in Figure 3.2</td>
</tr>
<tr>
<td>3.2</td>
<td>The presence of algorithm properties for both class imbalance and cost-sensitive classification algorithms. ✓ indicates that the corresponding algorithm has the corresponding algorithm property.</td>
</tr>
<tr>
<td>3.3</td>
<td>Bagging/Boosting-inspired approaches to mitigate the loss of information problem. Sorted by date.</td>
</tr>
<tr>
<td>3.4</td>
<td>Distance-based approaches to mitigate the loss of information problem. Sorted by date.</td>
</tr>
<tr>
<td>3.5</td>
<td>Summary of sampling methods - The presence of algorithm properties for both class imbalance and cost-sensitive classification algorithms. ✓ indicates that the corresponding algorithm has the corresponding algorithm property.</td>
</tr>
<tr>
<td>3.6</td>
<td>The number of times each base classifier was used when implementing a proposed wrapper method. (The implementation in the original proposing paper.)</td>
</tr>
<tr>
<td>3.7</td>
<td>Metadata on the datasets used in this section.</td>
</tr>
<tr>
<td>3.8</td>
<td>For any given cell, + indicates that the method corresponding to the row outperforms the method corresponding to the column. The converse is true for −. = denotes that the methods are considered equal in performance. This data is collected from the papers that propose the method corresponding to each row. Cells which are empty indicate that no comparison was made in the paper between the two corresponding methods.</td>
</tr>
<tr>
<td>4.1</td>
<td>The Proposed Minority and Majority Cluster Cost-Matrixes</td>
</tr>
<tr>
<td>4.2</td>
<td>Number of Majority and Minority records in each dataset.</td>
</tr>
<tr>
<td>4.3</td>
<td>AUC Comparison of the Methods (Higher the Better)</td>
</tr>
<tr>
<td>4.4</td>
<td>Execution Time Comparison of the Methods - Seconds to complete a 10x10 cross fold validation</td>
</tr>
</tbody>
</table>
5.1 Methods Comparison - Silhouette Coefficient ............................................. 90
5.2 Time to Find all CSCs (ms) ........................................................................ 91
5.3 The average execution time (seconds) of a fold in 10 cross-fold valida-
tion. Results shown are for Standoff with EM (proposed in Chapter 4) 
and Standoff with RBClust (proposed in this Chapter). .............................. 93

6.1 The different types of predictions in SDP and their associated costs. .... 100
6.2 MC2' Total Cost Comparison (Lower the better) ................................. 104
6.3 MC2' Total Cost Comparison (Using only Domain Costs in CSVoting) . 104
6.4 Combining Only FP and FN Costs ................................................................. 106
6.5 Metadata on the SDP datasets used in this study. Sorted by the num-
er of lines of code used in the project’s source code. ............................... 108
6.6 Total cost comparison (lower is better). Highlighted cells indicate 
the best performing methods for the corresponding dataset. The upper 
portion of this table indicates the properties of the corresponding 
method. ✓ indicates the presence of a property and X denotes the 
opposite. .............................................................. 109
6.7 The minimum, average, and maximum results of the cost spread anal-
ysis. The lower the mean, the better. ......................................................... 114
6.8 The maximum cost minus the minimum cost achieved in the cost 
spread analysis. A lower value indicates that the algorithm achieves 
more consistent performance. ................................................................. 115

7.1 The major contributions of this paper. ..................................................... 124
7.2 Metadata for the combined dataset MDP' .................................................. 131

9.1 AUC Comparison of the Methods (Higher the Better) ......................... 152
9.2 Methods Comparison - Silhouette Coefficient (The higher the better.) . 153
9.3 Total cost comparison (lower is better). Highlighted cells indicate 
the best performing methods for the corresponding dataset. The upper 
portion of this table indicates the properties of the corresponding 
method. ✓ indicates the presence of a property and X denotes the 
opposite. ................................................................. 155
A.1 Characteristics of the datasets used in Chapter 3. - Part 1 . . . . . . . . . . 160
A.2 Characteristics of the datasets used in Chapter 3. - Part 2 . . . . . . . . . . 161
List of Abbreviations

\[ A = \{A_1, A_2, ..., A_{|A|}\} \] A set of attributes
ACO Ant Colony Optimisation
AP Affinity Propogation
AUC Area Under the Curve
BCF Balanced Costs Framework
\( C_FN \) Cost of one False Negative prediction
\( C_FP \) Cost of one False Positive prediction
\( C_TN \) Cost of one True Negative prediction
\( C_TP \) Cost of one True Positive prediction
\( C_N \) Cost of labelling all records as Negative
\( C_P \) Cost of labelling all records as Positive
\( C(j|x) \) The cost of classifying record \( x \) as class \( j \)
\( C^\delta_FN \) The balancing cost of one \( C_FN \)
\( C^\delta_FP \) The balancing cost of one \( C_FP \)
\( C^\delta_TN \) The balancing cost of one \( C_TN \)
\( C^\delta_TP \) The balancing cost of one \( C_TP \)
\( C^\delta_FN \) The domain cost of one \( C_FN \)
\( C^\delta_FP \) The domain cost of one \( C_FP \)
\( C^\delta_TN \) The domain cost of one \( C_TN \)
\( C^\delta_TP \) The domain cost of one \( C_TP \)
\( C^\gamma_FN \) The combined cost of one \( C_FN \)
\( C^\gamma_FP \) The combined cost of one \( C_FP \)
\( C^\gamma_TN \) The combined cost of one \( C_TN \)
\( C^\gamma_TP \) The combined cost of one \( C_TP \)
CPRS Cost Proportionate Roulette Sampling
CS Cost Sensitive
CSC Class Specific Cluster
\(|D|\) The number of records in dataset \( D \)
\( D_{maj} \) All majority records in \( D \)
\( D_{min} \) All minority records in \( D \)
\( D_i \) All records in dataset \( D \) with the \( i \)th class value
\( D_j \) All records in dataset \( D \) with the \( j \)th class value
\( D_T \) Training Dataset
\( D' \) A sample of dataset \( D \)
DT Decision Tree
\( E \) Expected Classification Cost
ExampleCSDT Example Cost Sensitive Decision Tree
\( f \) A function in some source code
FAST Feature Assessment by Sliding Thresholds
FN False Negative
FNR False Negative Rate
FP False Positive
FPR False Positive Rate
HDDT Hellinger Distance Decision Tree
IBM International Business Machines
IDE Integrated Development Environment
IR Imbalance Ratio
IRUS Inverse Random Undersampling
KDD Knowledge Discovery from Data
KNN K-Nearest Neighbours
\( L = \{L_1, L_2, \ldots, L_L\} \) A set of possible class labels
\( M \) A set of majority class records
MUTE Majority Undersampling Technique
\( N \) A set of minority class records
\( N_{FN} \) Number of False Negative predictions
\( N_{FP} \) Number of False Positive predictions
\( N_{TN} \) Number of True Negative predictions
\( N_{TP} \) Number of True Positive predictions
NASA National Aeronautics and Space Administration
\( P(j|x) \) Probability that record \( x \) belongs to class \( j \)
\( R = \{R_1, R_2, \ldots, R|R|\} \) A set of records
\( R_i = \{V_1, V_2, \ldots, V|V|\} \) A set of values comprising a record
\( r_x \) The \( x \)th record in dataset \( D \)
RBClust Rule Based Clustering
ROC Receiver Operating Characteristic
ROS Random Oversampling
RUS Random Undersampling
\( S_i \) A Sample of \( i \)th class records
SBC Undersampling Based on Clustering
SCORE Stack of Cost Ordered Ruleset Extractions
SDP Software Defect Prediction
SibCost Sibling Ruleset Cost
\( sl(r_x) \) The safe level of \( r_x \)
SLS Safe Level SMOTE
SMOTE Synthetic Minority Oversampling Technique
SQL Structured Query Language
TN True Negative
TNR True Negative Rate
TP True Positive
TPR True Positive Rate
In memory of my grandfather Fred Siers (1930-2017)
Part I

Exordium
Chapter 1

Introduction

Classification is a major task in data mining. It involves predicting the correct labels of objects or events (Han et al., 2011a). For example, classification can be used to predict whether a given section of programming code contains bugs or not (Sheng et al., 2014). Classification is performed using predictive models called classifiers. These models are constructed by applying a classification algorithm to a given dataset. As input, these algorithms require a training dataset denoted as $D_T$. This dataset is comprised of a set of records $R = \{R_1, R_2, \ldots, R_{|R|}\}$ and a set of attributes $A = \{A_1, A_2, \ldots, A_{|A|}\}$. Each record is comprised of a set of values $R_i = \{V_1, V_2, \ldots, V_{|A|}\}$ where each value corresponds to an attribute in $A$. One $A_j \in A$ is designated the class attribute. Typically, the class attribute can only be equal to one of two different values. In general, these values are positive and negative. For example, the $A_j$ attribute could describe whether or not it will rain; in this example, $A_j = \text{positive}$ indicates that it will rain and $A_j = \text{negative}$ means it will not rain. This is called a binary class attribute. Class attributes can also be multi-class, that is, there are 3 or more possible values that the class attribute can be equal to. A classifier’s purpose is to predict a record’s class attribute value.

A dataset provides the measurements of many records. We refer to each measurement as an attribute. These attributes are analogous to column headings in a table or spreadsheet. To continue this analogy, each row is analogous to a record. For each cross section of row and column (record and attribute respectively) there is a value. This value is the result of measuring the corresponding record using the corresponding attribute. For example, consider an example where each row represents a person. The attributes might include: "Name", "Age", "Height" and "Favourite
Chapter 1. Introduction

Colour”. One record in the dataset could have the values: "Michael", "26", "182", "Blue".

Two challenges to classification are approached in this thesis. A brief overview of these challenges is given before stating the thesis’ research questions. The first is called the class imbalance problem (Japkowicz and Stephen, 2002).

**Class Imbalance**: Some training datasets contain many more records of one class than other classes. These datasets are referred to as class imbalanced (Chawla et al., 2002; Tahir et al., 2012). The class which occurs the most is called the *majority class*. Similarly, the class value which occurs the least is called the *minority class*. Classifiers trained on class imbalanced datasets often incorrectly predict minority class records as majority class (Chawla et al., 2002; Tahir et al., 2012).

**Cost-Sensitivity**: The traditional approach to classification is a cost-insensitive approach (Quinlan, 2014; Islam and Giggins, 2011) where the aim is to produce as many correct predictions as possible. An alternative approach to classification is a cost-sensitive approach (Ling et al., 2006; Sheng et al., 2014; Siers and Islam, 2014; Siers and Islam, 2015a). In the cost-sensitive approach, the aim is to make predictions which incur the lowest real-life cost. The real-life cost of predictions can be calculated if the classification costs are known. In a binary classification problem where there are two possible class values, positive and negative, there are four possible prediction types. A true positive prediction occurs when a record is correctly predicted as positive, a true negative when a record is correctly predicted as negative, a false positive when a record is incorrectly predicted as positive and a false negative when a record is incorrectly predicted as negative. Each of these predictions have an associated cost called a classification cost or misclassification cost. For example, consider a classifier which predicts whether or not a credit card transaction is fraudulent. The cost of a false negative might be the amount of the fraudulent transaction. The cost of a true positive could be the cost of an investigation and retrieval of the money.

1.1 Research Questions

This thesis focuses on three research questions. They are presented as follows.
**Research Question 1**: How can classification performance be further improved when the training data is class imbalanced?

Class imbalance is a widespread problem which negatively affects classification performance in many application areas. For example, class imbalance is a major issue in predicting the location of bugs in source code (Wang and Yao, 2013), detecting fraud (Kim et al., 2016), and social media spam detection (Liu et al., 2016). Application areas which commonly suffer from class imbalance typically have high costs associated with incorrect predictions. For example, the prediction of relapse in Breast Cancer suffers from class imbalance and it is highly costly to misdiagnose patients (Abreu et al., 2016). Thus, even marginal improvements to class imbalanced classification can have a large impact on society.

In Chapter 3, a thorough literature review of class imbalance and its relationship to cost-sensitivity is performed. To address Research Question 1, a novel technique in Chapter 4 called Standoff is proposed which achieves a higher performance than existing approaches without modifying the original dataset. Standoff had a barrier for further improvement; it was slow to execute and therefore took a long time to experiment with. Chapter 5 addresses this issue by proposing a new algorithm RB-Clust which replaces a major component of Standoff. This replacement allowed us to further experiment with Standoff and use it to explore Research Questions 2 and 3.

**Research Question 2**: For the purposes of building lower cost classifiers, how can strategy for addressing class imbalance be integrated into a cost-sensitive decision forest algorithm?

Algorithms which aim to negate the effects of class imbalance do so to achieve better classification performance (Tahir et al., 2012; Bunkhumpornpat et al., 2009). These algorithms are overwhelmingly used to improve the performance of cost-insensitive classifiers. (Later in this thesis a detailed experiment is performed which illustrates the performance improvement of class imbalance techniques over 99 datasets; see Figure 3.2.)

This thesis focuses on cost-sensitive classification algorithms due to two advantages that they have. Firstly, they directly report on the business costs. This allows the results to be presented in a more meaningful way. Secondly, the patterns that
are discovered using these algorithms can describe the best opportunities for cost-
savings to a business or organisation. However, not all classifiers can be manually
inspected to find useful patterns. Decision tree based methods are a type of classifier
which can be, thereby making them suitable for discovering knowledge (Islam and
Giggins, 2011). By building a collection of decision trees (known as a decision for-
est), the number of patterns that are found is greatly increased compared to a single
decision tree.

Chapter 3 describes how class balancing techniques aim to reduce classification
bias. Also described is how techniques for training cost-sensitive classifiers induce
classification bias. This has an interesting implication: if one was to use a class imbal-
ance algorithm such as SMOTE (Chawla et al., 2002) and then train a cost-sensitive
classifier, then the classification bias would be reduced, only to be induced again. In
that situation, applying a class balancing approach could possibly introduce classifi-
cation errors (Chawla et al., 2002), and similarly, applying a cost-sensitive approach
could also introduce classification errors (Sheng et al., 2014). Therefore classification
errors are possibly being introduced twice. Chapter 6 proposes BCF which does not
aim to reduce or induce bias, but instead adjusts the bias in one step. BCF is used to
address class imbalance whilst training a cost-sensitive decision forest algorithm.

**Research Question 3:** How can a cost-sensitive decision forest be better utilised
to perform cost-sensitive knowledge discovery on class imbalanced datasets?

Typically, class imbalance is treated by methods which change the original train-
ing dataset. This means that classifiers trained on the modified dataset do not di-
rectly describe patterns in the original dataset. The work so far in this thesis pro-
vides us with a tool (BCF) for generating a cost-sensitive decision forest from class
imbalanced data without modifying the original dataset. That is, each tree in the
decision forest is generated using the entirety of the original training dataset.

To discover cost-sensitive knowledge using BCF, a data scientist can inspect pat-
terns in the decision forest. This process would be manual and unguided. A guided
method should have the following three characteristics. First, there should be an
equation for quantifying how useful a decision forest’s pattern is. Second, there
should be a method for extracting and sorting the existing patterns based on their
usefulness. Thirdly, there should be a process for taking each pattern, enhancing it,
then reporting on it.

The above-mentioned characteristics are addressed in Chapter 7 by proposing a cost-sensitive interestingness measure, a pattern extraction algorithm, and a knowledge discovery process respectively. The proposed concepts are demonstrated by discovering cost-sensitive patterns which lead to software defects over several projects undertaken at NASA.

Main Contributions of this Thesis:

- A combined survey of cost-sensitive classification and class imbalance techniques for decision trees. The survey considers techniques from these two fields as belonging to a single family of techniques called bias adjustment techniques.

- A novel technique for classification in class imbalanced datasets which is designed through an analogy between class imbalance and warfare.

- Definition of the class-specific clustering problem and proposal of a novel technique to quickly and accurately handle it.

- A classification framework for cost-sensitive classification of class imbalanced datasets.

- Novel methods for cost-sensitive knowledge discovery from class imbalanced datasets with an application to NASA software defect data.

1.2 Overview of this Thesis

This thesis is structured into four parts. This section describes the purpose of each part, and the chapters within.

This section is part of Part I - Exordium. The exordium aims to provide the basic knowledge required to read this thesis. It first provides a introductory chapter: Chapter 1 - Introduction. Within that chapter, the research questions are described in detail. This current section is also a part of the introductory chapter.
The second chapter of Part 1 is called **Chapter 2 - Data Science: Core Concepts**. In that chapter, the data science concepts which are related to this thesis are described in beginner-friendly detail. This is to ensure that all readers have access to the required background knowledge to understand the thesis.

The final chapter in Part 1 is the literature review for this thesis: **Chapter 3 - Literature Review**. The literature review identifies a novel categorisation of the existing research. This enables us to discover unique insights from the literature.

**Part II - Classification in Imbalanced Data: Cost-Insensitive** presents novel approaches to cost-insensitive classification for class imbalanced datasets. This thesis’ proposed approach for cost-insensitive classification is presented in **Chapter 4 - Standoff Balancing: Classification in Class Imbalanced Datasets**. An advantage of this approach is that the original dataset is not changed, and the whole dataset is used for classifier training.

The approach proposed in Chapter 4 called **Standoff Balancing**, requires a clustering algorithm to be executed twice. Since clustering algorithms typically require a long time to execute (Rahman and Islam, 2014), a clustering algorithm is designed which only needs to be executed once to provide the results required for Standoff Balancing. This clustering algorithm is presented in **Chapter 5 - RBClust: Fast Discovery of Class Specific Clusters**.

In **Part III - Classification in Imbalanced Data: Cost-Sensitive**, the work from Part II is extended to address cost-sensitivity. This is achieved through two chapters.

The first of these chapters is **Chapter 6 - BCF: Balanced Costs Framework**. In that chapter, a method is proposed for combining the output of the approach in Chapter 4 with traditional domain costs. The experiments in that chapter find that this method produces a classifier with a better performance than the other studied methods.

The second chapter of Part III is **Chapter 7 - Cost-Sensitive Knowledge Discovery Using the Proposed Methods**. This chapter proposes three techniques which when used together with the work from Chapter 6, allow a data miner to perform cost-sensitive knowledge discovery. The strength of this approach is threefold: class imbalance is mitigated without affecting the original dataset, the whole dataset is
used for training, and the discovered knowledge is in terms of actual monetary value.

This thesis concludes in **Part IV - Denouement**.

Part IV first addresses discussion points which have arisen from this thesis in **Chapter 8 - Discussion**. A component of this discussion is a concept for integration of the work from Chapters 4 to 7 into the software development process. That chapter also discusses the possible research directions that may arise from this thesis.

Finally, concluding remarks are provided in **Chapter 9 - Conclusion**.
Chapter 2

Data Science: Core Concepts

According to IBM, 2.5 quintillion bytes of data are created everyday (Winans et al., 2017). That is equal to 2.5 exabytes or 2.5 million terabytes. This means that on average, each human creates one third of a gigabyte every day\(^1\). As the amount of collected data in the world increases, so does the demand for effective ways to utilise it. This demand is addressed by data mining. A newer term data science is becoming increasingly popular and is extremely similar to data mining. For the purposes of consistency and simplicity in this thesis, data mining and data science are considered synonymous. Figure 2.1, shows the results of a popularity comparison of data mining against data science which was generated using Google Trends (Google, 2012).

\[\text{Figure 2.1: A popularity comparison between the terms data mining and data science. Data source: Google Trends.}\]

\(^1\)Using the United Nations 2017 estimate of the world population: 7.5 billion
Chapter 2. Data Science: Core Concepts

In Figure 2.1, one can observe how the term data science has increased in popularity since 2012, whilst the term data mining has been declining in popularity since 2004. Since 2015, data science has been a more popular term than data mining.

Data science practitioners utilise a diverse multidisciplinary skill-set, including techniques from machine learning, data preprocessing, statistics, business analysis, and software engineering. From these fields, this thesis is most closely associated with machine learning and data preprocessing. To provide a background on the core concepts of these two fields this chapter describes the following:

- Classification (See Section 2.1)
- Clustering (See Section 2.2)
- Data Re-sampling (See Section 2.3)

2.1 Classification

Two major tasks in machine learning are classification and regression (Han et al., 2011b). Both tasks involve creating predictive models. For classification, these models predict categorical values and are called classifiers. For example, a classifier might predict whether tomorrow it will rain or not rain. To build a classifier, historical data is used as input to a classification algorithm. To continue the example, the historical data may describe the weather for the last 500 days using data such as humidity and temperature. The historical data will also include an attribute which describes whether or not it rained on each of the 500 days. Using classification terminology, humidity, temperature, and cloud liquid water content are called attributes, and whether or not it rained is called the class attribute. The historical data can therefore be thought of as a 2-dimensional table where each row represents one day and each column represents an attribute. This example historical data is illustrated in Table 2.1.

When used to train a classifier, the historical data is referred to as the training dataset. To train the classifier, the training data is used as input to a classification algorithm such as C4.5 (Quinlan, 2014). There are many families of classification algorithms to choose from. Each family has their own set of unique advantages and
disadvantages. The following subsections provide a brief overview of four popular families of classification algorithms: *Decision Trees, Nearest Neighbours, Ensembles, Decision Forests.*

### 2.1.1 Decision Trees

A great advantage of decision tree classifiers is that they are easily illustrated and as a result, easy to understand. For example the tree in Figure 2.2 is a decision tree which is trained using the example data from Table 2.1.

![Figure 2.2: A decision tree classifier trained from the example data in Table 2.1.](image)

This decision tree can be used to predict whether it will rain for any given day, as long as the humidity and temperature of that day is known. For example, consider that it is 10am on Monday, humidity is measured as 0.36 and it is 32 degrees. To predict whether it will rain on Monday, start at the root node of the decision tree. The attribute written inside the root node is *Humidity* and there are two branches coming
from the root node labelled $\leq 0.12$ and $> 0.12$. Since the humidity on Monday is 0.36, the latter branch is followed. This leads to the node which tests temperature. The process of testing attribute values is followed until a leaf is found. A leaf is a node with no child nodes. Monday falls into Leaf 3. The path it took to reach this leaf is illustrated using a broken orange line. Each leaf tells us how many records from the training data also fall into it. In leaf 3, there was rain on 120 days and no rain on 40 days. This means that for Monday, it is more likely to rain than to not rain. Therefore, it is predicted to rain on Monday.

Decision trees are especially useful for discovering insights from the training data. This is performed by extracting logic rules from the decision tree. A logic rule is extracted by following a path through the tree from the root node to any leaf node. The path taken can be written as the attribute tests that were performed along the path. For example, in Figure 2.2, the path from the root node to Leaf 3 can be written as: IF Humidity $> 0.12$ AND Temperature $> 29$ THEN Did Not Rain = 40, Rained = 120. From this logic rule, one can discover that days which fit these preconditions are 3 times more likely to rain than to not rain.

### 2.1.2 K-Nearest-Neighbours Classifiers

This type of classifier is simple to understand. The basic technique for K-Nearest-Neighbours (KNN) classification does not require the training of a predictive model. Rather, the training data itself is used to classify new records. Consider that the $k$ parameter of KNN is set to 5. To classify a new record $r$, the 5 closest records to $r$ within the training dataset are found. $r$ is then classified as the most common class of the 5 closest records. To measure closeness, a suitable distance metric needs to be chosen. The most well-known of which is Euclidean distance. The formula for Euclidean distance is shown in Equation 2.1. In this equation, $\bar{X}$ and $\bar{Y}$ are two records represented as a vector of attribute values: $\bar{X} = \{x_1, x_2, ..., x_{|X|}\}$ and $\bar{Y} = \{y_1, y_2, ..., y_{|Y|}\}$.

$$
\text{dist}(\bar{X}, \bar{Y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}
$$

(2.1)
2.1. Classification

One drawback of the KNN neighbours approach is that for categorical attributes, distance is harder to calculate. For example, it is intuitive to say that the distance between \( temperature = 25 \) and \( temperature = 29 \) is 4. However, the distance between \( colour = red \) and \( colour = blue \) is not intuitive. In the case of categorical attributes, a typical strategy is to measure the distance as 1 if the categorical values are different, and 0 if they are the same (Han et al., 2011b). When calculating distance, it is important to normalise each attribute’s values to be between 0 and 1.

2.1.3 Ensembles

An ensemble is a collection of classifiers that are used together to make predictions. The classifiers that comprise an ensemble may disagree on how to classify a new record. Therefore, a strategy must be employed to combine the predictions of each classifier into a single prediction. These strategies are called voting (Islam and Giggs, 2011). The two most common methods for producing an ensemble are bagging (Breiman, 1996) and boosting (Freund and Schapire, 1995). To provide a brief overview of ensembles, these two techniques are described in the following two paragraphs.

From a dataset \( D \), a subset of records \( D' \subseteq D \) can be created by randomly sampling with replacement from \( D \). This is referred to as a bootstrap sample. Bagging (Breiman, 1996) creates several bootstrap samples and builds a classifier on each to form an ensemble. Typically, the prediction of the ensemble is the most common prediction of the ensemble’s classifiers. Bagging has been shown to increase the performance of many classification algorithms including decision trees (Quinlan, 1996).

Boosting is a general concept which extends the concept of bagging. In boosting (Freund and Schapire, 1995), every record \( R_i \in D \) is assigned a weight \( w_i \). All weights are initialised as \( w_i = \frac{1}{|D|} \). Similar to bagging, boosting creates many samples with replacement. Unlike bagging, the chance that any given \( R_i \in D \) is chosen to be added to \( D' \) is equal to \( w_i \). Each time a sample is created from \( D \), a classifier is trained on the sample. Each weight is then adjusted: reduced if the record was correctly classified, increased if the record was incorrectly classified. Therefore, the next sample is more likely to sample records which were misclassified by the classifier trained on the previous sample.
A well-known implementation of boosting is AdaBoost (Freund and Schapire, 1995). Adaboost has three inputs: the training dataset which includes the class values, a classification algorithm such as C4.5 (Quinlan, 2014), the number of iterations T. Adaboost uses the boosting notation where $D_t(i)$ represents the weight of sample $D$ on record $i$ for the $t$’th iteration. On each iteration, the classification algorithm aims to find the best classifier on sample $D_t$. The error of the classifier is recorded and is measured as the number of records that are misclassified by the classifier ($\varepsilon_t$). The coefficient of the $t$’th classifier is denoted as $\alpha_t$ and is calculated as $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$. In each iteration, $\alpha_t$ is used to update the weights of each record such that the next sample will be different. This is performed using equation 2.2; where $y_i$ is the class value of record $i$, $h_t$ is the classifier trained in iteration $t$ and $h_t(x_i)$ is the classification of classifier $h_t$ on record $i$ given it’s non class attributes $x_i$. $Z_t$ is used as a normalisation factor.

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$  \hspace{1cm} (2.2)

### 2.1.4 Decision Forests

In the previous subsection, ensembles were described. This subsection describes a special type of ensemble which only contains decision trees, aptly named a decision forest. Decision forest algorithms often achieve better prediction accuracy than decision tree algorithms (Islam and Giggins, 2011; Siers and Islam, 2015a). Some decision forest algorithms such as SysFor (Islam and Giggins, 2011) and CSForest (Siers and Islam, 2015a) are designed to maximise the knowledge which can be extracted from the produced decision trees. Therefore, decision forests are extremely useful for discovering patterns and knowledge from datasets. The process of classifying a new unseen record using a decision forest is illustrated in Figure 2.3. This figure shows how the voting algorithm is used to combine the predictions of each individual tree to classify a single record.
2.1. Classification

2.1.5 Classifier Performance Evaluation

There exist many methods for evaluating the performance of a classifier. These methods are based on values which are extracted from a structure called a confusion matrix. This matrix considers two main ideas. The first is the predicted value and the second is the actual value. For example, a classifier might predict that it will rain on Monday, but it actually does not rain on Monday. Therefore, if the number of possible class values is $k$, then there are $k^2$ combinations of predicted and actual class values.

When testing the performance of a classifier, a test dataset is typically used. This test dataset does not share any records with the training dataset. The predictions made by the classifier on the test dataset is compared against the actual class values. Thus, the number of predictions for each combination of predicted and actual values can be represented by a confusion matrix as shown in Table 2.2.

<table>
<thead>
<tr>
<th>Predicted</th>
<th>No Rain</th>
<th>Rain</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Rain</td>
<td>56</td>
<td>5</td>
</tr>
<tr>
<td>Rain</td>
<td>27</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 2.2: An example confusion matrix with two possible class values: Rain and No Rain.

For example, from Table 2.2, one can calculate that when the classifier was tested using the test dataset, 27 days were predicted to rain when it did not actually rain. If Rain is considered to be the positive class and No Rain to be the negative class,
then the confusion matrix represents the number of true positives, false positives, true negatives, and false negatives. The false/true part refers to whether or not the predicted value matches the actual value, whilst the positive/negative part refers to the predicted value.

This thesis denotes the number of TP, FP, FN, and TN predictions as \( N_{TP} \), \( N_{FP} \), \( N_{FN} \) and \( N_{TN} \) respectively. Using the confusion matrix in Table 2.2, one can observe that \( N_{TP} = 12 \), \( N_{TN} = 56 \), \( N_{FP} = 27 \) and \( N_{FN} = 5 \). As mentioned earlier in this section, the confusion matrix is used to derive many different performance measures. A brief overview is now given for 5 measures used regularly in the research area of this thesis. These are precision, recall, true negative rate, accuracy, f-measure and AUC.

**Precision** measures the fraction of positive predictions that were correct.

\[
\text{Precision} = \frac{N_{TP}}{N_{TP} + N_{FP}} \quad (2.3)
\]

**Recall** or true positive rate is a measure of a classifier’s ability to correctly predict the class of actually positive records.

\[
\text{Recall} = \frac{N_{TP}}{N_{TP} + N_{FN}} \quad (2.4)
\]

**True negative rate** (TNR) is a measure of a classifier’s ability to correctly predict the class of actually negative records.

\[
\text{TNR} = \frac{N_{TN}}{N_{TN} + N_{FP}} \quad (2.5)
\]

**Accuracy** is an easy to understand measure. It is equivalent to asking: "Out of all the predictions made, which were correct?".

\[
\text{Accuracy} = \frac{N_{TP} + N_{TN}}{N_{TP} + N_{FN} + N_{TN} + N_{FP}} \quad (2.6)
\]

**F-Measure** is a combination of precision and recall. More specifically, it is the harmonic mean between precision and recall.

\[
\text{F-Measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (2.7)
\]
AUC, sometimes referred to as AUROC is the Area Under the Receiver Operating Characteristic curve (ROC curve). Along with a class prediction, some classifiers also output a probability that a record belongs to that class. Typically, if the probability is at least 50%, then that prediction is used. However, this threshold can be modified. For example, a record is only predicted as positive if the classifier gives at least 80% probability of it being true. An ROC curve can be generated using Algorithm 1.

**Algorithm 1:** Algorithm for generating an ROC curve.

```
Input: A classifier $C$, a test dataset $D_{test}$ and an interval $\theta$ where $\theta$ is a multiple of 100 and between 0 and 10
Output: An ROC curve
1 Let $X$ be a set of $x$ coordinates;
2 Let $Y$ be a set of $y$ coordinates;
3 Let $t$ be the threshold used to classify a record as positive using $C$;
4 $t \leftarrow 0$
5 while $t \leq 100$ do
6     // classify is a function which classifies a test dataset using a given classifier and threshold, then returns (true negative rate, true positive rate).
7     Append classify($D_{test}, C, t$) to $X$, $Y$;
8     $t \leftarrow t + \theta$;
9 end
10 return plot($X$, $Y$);
```

After an ROC curve is generated, the AUC is measured by calculating the area underneath it. Thus, the AUC is a measure of how well a classifier can compromise between correctly classifying positive records and correctly classifying negative records.

### 2.2 Clustering

Within a dataset, there are typically groups of records with similar attribute values. These groups are called clusters. The process of discovering clusters is best explained visually (see Figure 2.4). Clustering algorithms such as k-means (MacQueen, 1967) are used to automatically discover these clusters. Note that in Figure 2.4, not all records were found to belong to a cluster. Such records are called outliers. Clustering can be used to discover patterns and knowledge from a dataset. For example, one can observe visually that there is a cluster of mostly rainy days when temperature is low and humidity is high. Similarly, there is a cluster of mostly rain free days when temperature is high and humidity is low. Additionally, clustering can also be
used to detect outliers, then remove them so that they do not perturb the knowledge discovery process.

There are many different metric for evaluating the fit of a clustering solution. This section briefly introduces the silhouette coefficient. Silhouette coefficient is a metric calculated for a single record after a clustering solution has been found. It considers two main values. $a$ is the average distance from the record to all other records in the same cluster. $b$ is the average distance from the record to all records which aren’t in the same cluster. The silhouette coefficient for the record is then calculated using Equation 2.8.

$$s = \frac{b - a}{\max(a, b)}$$  \hspace{1cm} (2.8)

The silhouette coefficient can be in the range $-1$ to $1$. The higher the value, the more appropriately clustered the record is. When evaluating an entire clustering solution, the average silhouette coefficient of all records is taken.

### 2.3 Data Resampling

Data resampling is a strategy which aims to change the number of certain types of records in a dataset. For example, by duplicating some records which describe rainy days, there are more rainy days in the training dataset. The obvious question is: Why would a data miner do this? The answer is that the class distribution affects
2.4. Supervised, unsupervised, and active learning

two issues in classification, namely class imbalance and cost-sensitivity. These two concepts are explained in detail in Chapter 3. Therefore, for detailed information on resampling, readers are referred to that Chapter.

Resampling is an example of data preprocessing. That is, it occurs before the data is used for something like classification (see Section 2.1). The advantage that preprocessing techniques have is that they are independent of future steps like classification or clustering. In the case of some resampling techniques, the training dataset is significantly reduced which decreases the computation time required to build a classifier or perform clustering (Liu et al., 2009).

2.4 Supervised, unsupervised, and active learning

There are three major types of machine learning. Supervised, unsupervised, and active learning (Han et al., 2011a).

In supervised learning, the learning approach uses the class labels of all records. This means that class labels must be provided for all the training records. A supervised learning approach such as classification uses the records’ class labels to learn patterns within the data.

Unsupervised learning is the opposite of supervised learning in the sense that such an approach uses none of the records’ class labels. This means that class labels are not required. Additionally, this means that unsupervised learning is appropriate when class labels cannot be provided. Clustering is an example of unsupervised learning.

Active learning approaches typically halt execution at several points during the learning process. During these halts, a domain expert is asked to manually label subsets of the training records. This way, both the machine and the human expert work together to produce a learned model. A drawback of active learning is that it’s less automated compared to supervised and unsupervised learning.

There is also a combination between supervised and unsupervised learning where some of the training records’ class labels are used. This is referred to as semi-supervised learning. The advantage of semi-supervised learning approaches is that training datasets which have some missing class labels can be fully used.
Chapter 3

Literature Review

This chapter serves two purposes for the thesis. The first is to perform a thorough exploration of the related research areas of this thesis: Class imbalance and Cost-sensitive learning. The second is to explore Research Question 1. The literature review considers techniques from both fields under the unified context of bias adjustment. This unified context facilitates direct comparison between techniques from both fields, which in turn, highlights lessons which may be learned from one field and applied to the other. By taking this approach, the relationship between both fields is explored in detail, thereby exploring Research Question 1.

3.1 Introduction

A major task in data mining and machine learning is classification (Han et al., 2011a). Given a dataset D, a classification algorithm builds a predictive model called a classifier. This model can make predictions about new data such as whether a Java function contains bugs (Wang and Yao, 2013) or whether customers will churn (Zhu et al., 2017). As input, classifiers require a training dataset denoted as DT. This dataset is comprised of a set of records R = {R1, R2, ..., R|R|} and a set of attributes A = {A1, A2, ..., A|A|}. Each record is comprised of a set of values R = {V1, V2, ..., V|A|} where each value corresponds to an attribute in A. One A j ∈ A is designated the class attribute. One attribute is designated the class attribute. The classifier built from D can predict the value of the class attribute for new unseen records.

The work presented in this chapter is currently submitted for review in ACM Transactions on Knowledge Discovery from Data, H-Index: 40, Scimago Q1 2017 under the title: "Class Imbalance and Cost-Sensitive Classification in Decision Trees: A Unified Survey Based on a Fundamental Similarity".
A dataset provides the measurements of many records. We refer to each measurement as an attribute. These attributes are analogous to column headings in a table or spreadsheet. To continue this analogy, each row is analogous to a record. For each cross section of row and column (record and attribute) there is a value. This value is the result of measuring the corresponding record using the corresponding attribute. For example, consider an example where each row represents a person. The attributes might include: "Name", "Age", "Height" and "Favourite Colour". One record in the dataset could have the values: "Michael", "26", "182", "Blue".

Classification is applied in many areas such as financial credit risk assessment (Zhang et al., 2017), social media sentiment analysis (Giachanou and Crestani, 2016), and cancer relapse prediction (Abreu et al., 2016).

Using evaluation metrics, the performance of a classifier can be quantified. Classifier performance can be negatively affected by many issues such as: missing values (Rahman and Islam, 2013), class overlap (Alejo et al., 2013) and class imbalance (Japkowicz and Stephen, 2002). Each of these issues constitutes a large research area and many algorithms have been proposed which mitigate these issues. This survey focuses on the class imbalance issue. Traditional classification algorithms are generally considered unsuitable for applications which are affected by class imbalance (Zhu et al., 2017; Wang and Yao, 2013; Chen et al., 2011). Furthermore, the commonly used evaluation metric prediction accuracy is also unsuitable. For class imbalanced applications, data miners commonly evaluate classifiers using AUC or $F_{value}$ in addition to other metrics (Guo et al., 2008).

This chapter identifies a fundamental similarity between the field of class imbalance research and another field of research: cost-sensitive classification (Elkan, 2001) which is a special type of classification. The similarity is that algorithms from both fields of research aim to adjust the prediction biases in resulting classifiers. Figure 3.1 illustrates the proposed taxonomy which categorises the properties of algorithms from both fields. A definition is provided of the major categories of the taxonomy in Section 3.2.4.

Readers should note that although this chapter surveys many techniques, it does not exhaustively cover all techniques from class imbalance or cost-sensitive learning in decision trees. For exhaustive surveys on these topics, readers are referred to...
3.1. Introduction

The proposed taxonomy for the properties of bias adjustment techniques in decision trees. Properties of class imbalance treatment algorithms and cost-sensitive classification algorithms can be categorised using this taxonomy.

The following class imbalance survey (Branco et al., 2016) and cost-sensitive decision trees survey (Lomax and Vadera, 2013). Instead, this survey discusses approaches from both fields within the unified context of bias adjustment. This context considers techniques for approaching class imbalance and techniques for approaching cost-sensitivity as techniques which adjust prediction bias. In the case of class imbalance, bias towards the majority class is aimed to be reduced whereas in the case of cost-sensitivity, bias is aimed to be induced. We explain this context further in the following section. Techniques which do not aim to adjust bias in classification algorithms do not fall under the proposed definition of bias adjustment. This would include traditional classification algorithms like C4.5 (Quinlan, 2014) and CART (Breiman, 2017). Sampling algorithms which select samples from all classes at random are not bias adjustment.

In Section 3.2 the proposed bias adjustment taxonomy is described which categorises approaches from both fields of research. Sections 3.3, 3.4 and 3.5 discuss how each taxonomy subcategory has been approached from both fields. Section 3.6 experimentally demonstrates that sharing lessons between the two fields is beneficial. Finally, in Section 3.9 a summary is provided of the lessons that can be learned between the two fields and the potential future research directions that they imply.
Chapter 3. Survey on Class Imbalance and Cost-Sensitive Classification

3.2 The Fundamental Similarity: Bias Adjustment

This section outlines the main research goals of class imbalance and cost-sensitive classification (Sections 3.2.1 and 3.2.2 respectively). The common goal of these fields, bias adjustment, is described in Section 3.2.3. Finally, the major categories of the proposed taxonomy are defined in Section 3.2.4.

3.2.1 Main Goal of Class Imbalance Research

Let $D_i \subseteq D$ denote the set of all records which belong to the $i^{th}$ class. Class imbalance is commonly described as when one $|D_i|$ is significantly greater than all other $|D_j|$ where $D_j \subseteq D$. Class imbalance research is typically focused on two-class datasets. As a result, the class with the least records is called the minority class, and the class with the most records is called the majority class. Classifiers trained on $D$ are likely to be biased towards predicting new records as belonging to the majority class.

The level of class imbalance in $D$ is measured by its imbalance ratio (IR). This is calculated using Equation 3.1 where $D_{maj}$ and $D_{min}$ are the set of majority class and minority class records in $D$ respectively.

$$\text{IR} = \frac{|D_{maj}|}{|D_{min}|} \quad (3.1)$$

This chapter presents an experiment which illustrates class imbalance. Within this experiment, classifiers are trained on 99 datasets which are known to be class imbalanced (available from KEEL (Fernández et al., 2007; Fernández et al., 2017)) (See Table 3.1 for specific details of this experiment). The IR of each dataset and the corresponding classifier’s performance (measured using AUC) is plotted in Figure 3.2a. As illustrated by the linear regression line added to this scatter plot, lower classifier performance is associated with higher imbalance ratios. 25 out of 99 classifiers achieved an AUC higher than 0.95 which is generally considered great performance. These 25 datasets are evidence that great classification performance can sometimes be attained even when the dataset is imbalanced. This is further evidenced by the calculated Pearson correlation coefficient between IR and AUC: $-0.242$.

Figure 3.2b is the same as Figure 3.2a except a class imbalance treatment algorithm was applied to each dataset prior to training the classifier. Within this figure,
3.2. The Fundamental Similarity: Bias Adjustment

(Figure 3.2: The AUC of classifiers before and after treating class imbalance. AUC is a measure of classification performance (the higher the better). A linear regression line is included in both figures.)

(a) AUC of classifiers (using C4.5 algorithm) trained on 99 datasets of varying imbalance ratios.

(b) Same as plot a except a class imbalance treatment algorithm (SMOTE) was applied to each dataset prior to classifier training.
### Table 3.1: The experiment settings used to generate the results shown in Figure 3.2

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification algorithm</td>
<td>C4.5</td>
<td>The J48 implementation in WEKA with default parameters.</td>
</tr>
<tr>
<td>Balancing algorithm</td>
<td>SMOTE</td>
<td>The SMOTE implementation in WEKA with default parameters.</td>
</tr>
<tr>
<td>Evaluation process</td>
<td>k Cross-Fold Validation</td>
<td>The k cross-fold validation implementation in WEKA with $k = 10$.</td>
</tr>
<tr>
<td>Evaluation metric</td>
<td>AUC</td>
<td>The WEKA implementation.</td>
</tr>
</tbody>
</table>

one can observe that before a class imbalance treatment is performed, 10 classifiers were achieving less than 0.5 AUC. Classifiers with less than 0.5 AUC are useless since random guessing achieves 0.5 AUC (Chen and Wasikowski, 2008). Therefore, this thesis refers to them as junk classifiers. After performing the class imbalance treatment (right plot), there are only 2 junk classifiers. One can also observe that the worst performing classifier on the original datasets increased in AUC by approximately 0.45 after treatment. Finally, to summarise the increase in performance through treatment of class imbalance, the original datasets average AUC was 0.78 whereas after the treatment, the average AUC was 0.827.

These observations demonstrate that when class imbalanced datasets are treated, classifier performance can be increased, thereby mitigating the class imbalance problem. In other words, figure 3.2 illustrates the main goal of class imbalance research: to reduce the prediction bias caused when IR > 1.

#### 3.2.2 Main Goal of Cost-Sensitive Classification Research

Traditionally, classifiers are designed to minimise prediction errors (Quinlan, 2014). Cost-sensitive classifiers are designed to minimise the total cost incurred by their predictions. These costs are user-defined. They are typically defined per class (Sheng et al., 2014; Siers and Islam, 2015a; Elkan, 2001). For a dataset with two possible class values, positive and negative, $C_{TP}, C_{TN}, C_{FP}$, and $C_{FN}$ are the costs incurred by true positive (TP), true negative (TN), false positive (FP) and false negative (FN).
predictions respectively. Thus, Equation 6.1 can be used to calculate the total cost of a classifier where $N_{TP}, N_{TN}, N_{FP}$, and $N_{FN}$ are the number of TP, TN, FP and FN predictions made by the classifier during testing.

$$\text{Total Cost} = N_{TP} \times C_{TP} + N_{TN} \times C_{TN} + N_{FP} \times C_{FP} + N_{FN} \times C_{FN}$$ (3.2)

Application areas where there is a large difference between costs are considered cost-sensitive (Elkan, 2001). Decision trees are a popular type of classifier for cost-sensitive classification (Ling et al., 2006), (Sheng et al., 2014), (Bahnsen et al., 2015), (Siers and Islam, 2014), (Siers and Islam, 2015a). Figure 3.3 shows a cost-sensitive decision tree which predicts C functions as either contains bugs or contains no bugs. This decision tree was trained using data obtained from flight software for an earth orbiting satellite built by NASA. Consider a C function `foo_bar()` which has 10 blank lines, and 53 commented lines. Using this tree, `foo_bar()` is predicted to contain bugs because it is cheaper than predicting it as bug free ($100 < $318.18). The path through the tree which was used to make a prediction for `foo_bar()` is illustrated as an orange-broken line.

![Figure 3.3: A cost-sensitive decision tree which predicts whether or not a C function contains bugs. Trained on data collected from source code data from a NASA satellite’s flight software. Algorithm: CSTree (Sheng et al., 2014).](image)

Many decision tree classifiers are trained using a greedy divide-and-conquer approach. That is, they recursively split the training data into partitions based on some heuristic. A common approach to produce a cost-sensitive decision tree is to use a cost-minimising heuristic (Sheng et al., 2014; Siers and Islam, 2015a). For example,
the algorithm CSTree (Sheng et al., 2014) was used to build the tree in Figure 3.3 such that its total cost is minimised. This is the main goal of cost-sensitive classification research: to induce classifier prediction biases such that the total cost of the predictions is minimised.

3.2.3 Common Goal of Class Imbalance and Cost-Sensitive Techniques: Bias Adjustment

Sections 3.2.1 and 3.2.2 describe the main research goal for class imbalance and cost-sensitivity. For class imbalance research, algorithms are desired which reduce prediction bias whereas in cost-sensitive research, algorithms are desired which induce prediction bias. The fundamental research goal that is common to both fields is to adjust prediction bias. Naturally, this has led to some overlap between both fields. For example, cost-sensitive classification is often cited in class imbalance studies as a general approach for imbalance treatment (Liu et al., 2009; Bunkhumpornpat and Sinapiromsaran, 2017; Ofek et al., 2017). Similarly, a family of techniques which is heavily used in class imbalance research called undersampling (Seiffert et al., 2010; Tahir et al., 2012; Lin et al., 2017), has also been used in cost sensitive studies to induce cost-sensitive classifiers (Domingos, 1999; Ling et al., 2006). This survey discusses approaches from both fields within the unified context of bias adjustment. These approaches are categorised using the proposed taxonomy (shown in Figure 3.1). The major categories of the proposed taxonomy are described in the following section.

3.2.4 The Proposed Bias Adjustment Taxonomy

The proposed bias adjustment taxonomy is shown in Figure 3.1. It does not directly aim to categorise techniques from both fields. Instead, it categorises properties which are common to techniques in both fields. The major categories include: dataset modification, algorithm modification and direct adjustment. Each of these categories are subcategorised. However, to provide an effective overview, the major categories are described first.
3.2. The Fundamental Similarity: Bias Adjustment

Dataset Modification properties involve changing the dataset in a way which adjusts prediction bias. This survey further categorises dataset modification properties as either resampling (Ofek et al., 2017; Tahir et al., 2012) or feature selection. Details of these subcategories can be found in Section 3.3.1 and 3.3.2 respectively.

Algorithm Modification properties involve the modification of base classification algorithms such as C4.5 (Quinlan, 2014) such that bias is adjusted in the resulting classifiers. This chapter further categorises algorithm modification properties as either splitting criteria modification or wrapper methods. Details of these subcategories can be found in Section 3.4.1 and 3.4.2 respectively.

Direct Bias Adjustment properties involve explicitly adjusting bias in classifiers. This chapter identifies four different levels at which this can occur: class, example, model, and cluster. Intuitively, a data miner may expect that class level bias is naturally involved in all class imbalance and cost-sensitive classification algorithms. However, some techniques are designed to be insensitive to bias without considering the class distribution (Dietterich et al., 1996; Cieslak and Chawla, 2008). Class, example, model, and cluster level properties are further discussed in Sections 3.5.1, 3.5.2, 3.5.3, and 3.5.4 respectively.

Table 3.2 illustrates which properties are present in existing techniques.

Benefit of the Unified Bias Adjustment Category: After recognising the shared characteristic between cost-sensitive learning and class imbalance treatment, a data scientist can start to recognise opportunities for cross-field learning. For example, in Section 3.6, an empirically demonstrated benefit is shown. In that section, a well-known element of class imbalance treatment is used to enhance a cost-sensitive learning technique. By doing so, a lower total cost is achieved. This demonstrates that there is merit to the unified view. Throughout this survey chapter, several more opportunities are identified (See Section 3.9 for a summary).

Furthermore, this thesis is heavily influenced by the idea of treating class imbalance and cost-sensitivity as a single bias adjustment problem. In Chapter 6, the algorithm BCF is proposed which addresses both problems using a single solution rather than treating both problems individually.

The aim of this survey and of combining both the class imbalance and cost-sensitive literature is to bridge the gap between the two fields; fostering inter-field
learning such as that presented in 3.6.

### 3.2.5 Cost-Sensitive Techniques Which Also Address Class Imbalance

Some studies have investigated addressing both issues. The aim of these studies is to produce cost-sensitive classifiers which achieve greater performance. For example, BCSForest (Siers and Islam, 2015a) first creates several samples of the dataset using a non-determinate class balancing sampling technique such as SafeLevelSMOTE (Bunkhumpornpat et al., 2009). On each of the samples, the CSForest algorithm (Siers and Islam, 2014) is applied to produce a cost-sensitive forest. By combining all the forests together into a single forest, a cost-sensitive forest which is trained on balanced samples is created. The experiments used to evaluate BCSForest concluded that it could produce better performance than CSForest sometimes, but not consistently. Sampling techniques can also be used prior to performing cost-sensitive learning such as in S-CSL (Thai-Nghe et al., 2010).

### 3.3 Dataset Modification

Before applying a classification algorithm to a dataset, it can be modified such that the bias of the resulting classifier is adjusted (Chawla et al., 2002; Han et al., 2005; Chen and Wasikowski, 2008). This form of bias adjustment can be advantageous due to its independence from the classifier building process. Some dataset modification techniques also significantly reduce the size of the training data. Consequently, classification algorithms which are applied to the modified dataset execute faster (Liu et al., 2009; Tahir et al., 2012). However, some other techniques increase the size of the training data, which has the opposite effect. A disadvantage of dataset modification is that the data no longer accurately reflects the domain it was collected from which is not desired for knowledge discovery and analysis.

This chapter subcategorises dataset modification into resampling and feature selection. These subcategories are discussed in Sections 3.3.1 and 3.3.2 respectively.
3.3. Dataset Modification

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Citation</th>
<th>Sampling</th>
<th>Feature Selection</th>
<th>Splitting Criteria Modification</th>
<th>Wrapper Method</th>
<th>Class Level Bias</th>
<th>Record Level Bias</th>
<th>Model Level Bias</th>
<th>Class Specific Cluster Bias</th>
<th>Ensemble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition-Deletion</td>
<td>(Zhao et al., 2013b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACOSampling</td>
<td>(Yu et al., 2013)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adasyn</td>
<td>(He et al., 2008)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtracking Feature Selection</td>
<td>(Zhao et al., 2013a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BalanceCascade</td>
<td>(Liu et al., 2009)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Borderline-SMOTE</td>
<td>(Han et al., 2005)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustering-Based Undersampling</td>
<td>(Lin et al., 2017)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Costing</td>
<td>(Zadrozny et al., 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSForest</td>
<td>(Siers and Islam, 2014)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSRoulette</td>
<td>(Sheng and Ling, 2007)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSTree</td>
<td>(Sheng et al., 2014)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSVoting</td>
<td>(Siers and Islam, 2015a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DKM</td>
<td>(Dietterich et al., 1996)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EasyEnsemble</td>
<td>(Liu et al., 2009)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExampleCSDT</td>
<td>(Bahnson et al., 2015)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAST</td>
<td>(Chen and Wasikowski, 2008)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FastCBUS</td>
<td>(Ofek et al., 2017)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDDT</td>
<td>(Cieslak and Chawla, 2008)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRUS</td>
<td>(Tahir et al., 2012)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MetaCost</td>
<td>(Domingos, 1999)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUTE</td>
<td>(Bunkhumpornpat et al., 2011)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NearMiss</td>
<td>(Zhang and Mani, 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tomek-Links Undersampling</td>
<td>(Tomek, 1976)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partitioning</td>
<td>(Chang et al., 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-Bagging</td>
<td>(Chang et al., 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB Bagging</td>
<td>(Hido et al., 2009)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUS</td>
<td>(Drummond and Holte, 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUSBoost</td>
<td>(Seiffert et al., 2010)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROS</td>
<td>(Drummond and Holte, 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safe-Level-SMOTE</td>
<td>(Bunkhumpornpat et al., 2009)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td>(Yen and Lee, 2006)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMOTE</td>
<td>(Chawla et al., 2002)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standoff</td>
<td>Chapter 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Underbagging</td>
<td>(Barandela et al., 2003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: The presence of algorithm properties for both class imbalance and cost-sensitive classification algorithms. ✓ indicates that the corresponding algorithm has the corresponding algorithm property.
3.3.1 Resampling

This chapter further categorises resampling as either class specific or all class resampling. All class resampling algorithms choose a set of records from the original training dataset $D$. This chosen sample $D'$ is then used as the training dataset. For class specific resampling, records are chosen from the set of $i$ class records $D_i$ to produce a sampled set of $i$ class records $D'_i$. Then, $D'_i$ is combined with all $D_j \in D$, $j \neq i$ to form a sampled training dataset $D'$.

Resampling methods are overwhelmingly designed for the class imbalance problem. More specifically, the papers they are published in are almost always proposed and evaluated within the class imbalance context. In a study which presented the fundamentals of cost-sensitive learning, the "correct" method for resampling to achieve cost-sensitivity was also presented (Elkan, 2001). The method is simply a formula for calculating the optimal sample’s class distribution. It does not describe how to perform the sampling. Therefore, it can be used in conjunction with resampling methods that were originally proposed for class imbalance. The only condition is that the user must be able to control what the sample’s class distribution will be.

Elkan first describes how the traditional threshold for classification in a binary problem is 0.5. That is, a record predicted as having a probability greater than 0.5 for belonging to the positive class results in that record being classified as positive and negative otherwise. In a cost-sensitive problem, this threshold should be altered in some way in order to induce bias. For example, if the threshold was 0.2 then records are far more likely to be classified as positive. This is fundamentally cost-sensitive. Elkan formulated and proved a formula for calculating what the optimal threshold is given a cost-matrix. This formula is shown in Equation 3.3 where the optimal threshold is denoted as $p^*$.

\[
\begin{equation}
  p^* = \frac{C_{FP} - C_{TN}}{C_{FP} - C_{TN} + C_{FN} - C_{TP}}
\end{equation}
\]

Elkan then mathematically proved a new formula which provides a value which this chapter denotes as $\phi$. The formula is given in Equation 3.4 where $p_0$ denotes the original probability that any given record in the training dataset belongs to the negative class. This value is used to multiply the number of negative records in the
dataset using whatever method a data miner chooses. When used as the training dataset in an error-based classifier, a cost-sensitive classifier will be produced. The resulting ratio between positive and negative records causes a bias which mathematically equates to the optimal threshold calculated in Equation 3.3.

\[
\phi = \frac{p^*}{1 - p^*} \frac{1 - p_0}{p_0}
\]  

(3.4)

Resampling: Class Specific

Class specific resampling is performed by replacing \( D_i \) with a sampled version: \( D'_i \). If \( D_i \) is sampled such that \( |D'_i| < |D_i| \), it is referred to as undersampling. Similarly, if \( |D'_i| > |D_i| \), it is referred to as oversampling. It has been widely observed that the lower the value of \( \frac{|D_i|}{|D|} \), the less biased a resulting classifier will be towards classifying a given record as class \( i \) (Tahir et al., 2012; Seiffert et al., 2010; Liu et al., 2009). Conversely, the higher the value of \( \frac{|D_i|}{|D|} \), the more biased a resulting classifier will be towards predicting unseen records as class \( i \) (Han et al., 2005; Bunkhumpornpat et al., 2009). For this reason, undersampling and oversampling have been used for treating class imbalance and inducing cost-sensitive classifiers.

The most basic form of undersampling and oversampling is performed at random (Drummond and Holte, 2003). These are referred to as random undersampling (RUS) and random oversampling (ROS). RUS selects a completely random subsample of majority class records from the training dataset and combines this set with the minority class records to create a new training dataset. Similarly, ROS selects a completely random supersample of minority class records from the training dataset and combines this set with the majority class records to create a new training dataset. To sample a superset, ROS can sample the same record more than once. Both suffer from serious drawbacks which result from their randomness. When proposing novel undersampling methods, data miners focus on mitigating the loss of information problem which is described below (Yu et al., 2013; Seiffert et al., 2010; Tahir et al., 2012).

The loss of information problem: Consider a set of \( i \) class records \( I \subset D_i \) which are important records for a classification algorithm to learn from so that the resulting classifier can correctly differentiate the \( i^{th} \) class. Undersampling at random is likely
Chapter 3. Survey on Class Imbalance and Cost-Sensitive Classification

to cause a low value for $|D'_i \cap I|$. This means any resulting classifiers are likely to have a reduced prediction accuracy for $i$ class records.

**Bagging/Boosting-based approaches to mitigate the loss of information problem:**

The bagging/boosting-based approach is to sample the dataset multiple times, then train a classifier on each (see citations in Table 3.3). The major advantage of this approach is that more $r_x \in D_{maj}$ are used for training. These approaches are heavily influenced by Bagging and Boosting (Breiman, 1996). Although not designed for it, Boosting has been demonstrated to mitigate the loss of information (Seiffert et al., 2008). RUSBoost (Seiffert et al., 2010) performs RUS during each Boosting iteration. It was shown to outperform both RUS and Boosting for class imbalanced classification.

<table>
<thead>
<tr>
<th>Method</th>
<th>Citation</th>
<th>Method</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>(Chan and Stolfo, 1998)</td>
<td>BalanceCascade</td>
<td>(Liu et al., 2009)</td>
</tr>
<tr>
<td>Underbagging</td>
<td>(Barandela et al., 2003)</td>
<td>RB Bagging</td>
<td>(Hido et al., 2009)</td>
</tr>
<tr>
<td>Quasi-Bagging</td>
<td>(Chang et al., 2003)</td>
<td>RUSBoost</td>
<td>(Seiffert et al., 2010)</td>
</tr>
<tr>
<td>EasyEnsemble</td>
<td>(Liu et al., 2009)</td>
<td>IRUS</td>
<td>(Tahir et al., 2012)</td>
</tr>
</tbody>
</table>

**Table 3.3:** Bagging/Boosting-inspired approaches to mitigate the loss of information problem. Sorted by date.

RUSBoost is similar to another technique called EasyEnsemble (Liu et al., 2009). The difference between these two techniques is that EasyEnsemble combines RUS and bagging by performing RUS during each bagging iteration. It creates multiple samples $D' = D_{min} \cup D'_{maj}$ where $D'_{maj}$ is a random sample of $D_{maj}$ and $|D'_{maj}| = |D_{min}|$. A classifier is trained on each sampled dataset and predictions are made using the resulting ensemble of classifiers. As described by its name, EasyEnsemble’s main advantage is its simplicity. In the same publication, the authors of EasyEnsemble also proposed an extension called BalanceCascade. In BalanceCascade, after each sample is taken, a classifier is trained on the sample. Majority records correctly classified by the classifier cannot be sampled in subsequent samples. Therefore, BalanceCascade is very similar to RUSBoost.

EasyEnsemble is very similar to the algorithm RB Bagging (Hido et al., 2009). In fact, these algorithms are almost identical except for two main differences. The first is that RB Bagging does not use RUS to sample the majority records. Instead, it
randomly samples majority records according to a negative binomial distribution (a distribution based on the concept of Bernoulli trials). The second is that there is an option to sample the minority records with replacement, meaning that each classifier is trained on a different set of minority records and majority records.

EasyEnsemble is almost identical to a technique referred to as Underbagging (Barandela et al., 2003). The difference between these techniques is the number of samples they generate. EasyEnsemble allows the user to specify how many samples to create. Underbagging creates \( \frac{|D_{maj}|}{|D_{min}|} \) samples (rounded to the nearest integer). Underbagging is the same as a technique referred to as Partitioning (Chan and Stolfo, 1998). Additionally, EasyEnsemble is exactly the same as a technique called Quasi-Bagging (Chang et al., 2003).

The core concept of IRUS (Tahir et al., 2012) is counter-intuitive. IRUS creates multiple samples for which the imbalance is inverted. For example, if \( |D_{min}| = 100, |D_{maj}| = 1000 \), then \( |D'_{min}| = 100, |D'_{maj}| = 10 \). Unlike EasyEnsemble and BalancedCascade, IRUS samples without replacement. This guarantees that a majority record cannot appear in more than one sample. Therefore, the samples are much more diverse which leads to greater performance in ensembles (Freund and Schapire, 1995).

Distance-based approaches to mitigate the loss of information problem:

These approaches measure the distance between a given record and its neighbouring records. This distance is then used to determine whether or not to exclude the record from a sample. Several of these approaches involve undersampling based on the clusters in \( D \). Distance based calculations are typically computationally expensive, especially in high dimensional data. Therefore, even though undersampling produces smaller datasets, these approaches can increase the required computation time. Another disadvantage of these approaches is that they are typically deterministic. This means they cannot be used in conjunction with bagging similar to EasyEnsemble or Quasi-Bagging. Table 3.4 shows the existing approaches which are distance-based.

MUTE (Majority Undersampling TEchnique) (Bunkhumpornpat et al., 2011) removes records from \( D_{maj} \) which it considers to be noisy. To determine whether or not a given majority record \( n \) is noisy, it counts how many minority records are
within its k-nearest neighbours. This count is referred to as the record’s *safe-level*. The safe-level of the $x^{th}$ record $r_x$ can be calculated using equation 3.5. If the safe-level of $r_x$ (written as $sl(r_x)$) is greater than or equal to a user-defined threshold $\tau$, then it is considered a noisy record.

$$sl(r_x) = \text{the number of minority nearest neighbours of } r_x \quad (3.5)$$

MUTE is similar to another technique called Tomek-Links undersampling ([Tomek, 1976](#)). A Tomek link between a majority record $x$ and minority record $y$ exists if no record $z$ exists such that either of the following conditions are satisfied (Equations 3.6 and 3.7). In these equations, $\delta$ is a function which calculates the distance between two given records.

$$\delta(x, z) < \delta(x, y) \quad (3.6)$$

$$\delta(y, z) < \delta(y, x) \quad (3.7)$$

Tomek-Links undersampling involves finding all Tomek-Links within the dataset. The sampled dataset $D'$ includes all records except the majority records which are part of a Tomek-Link. The principle idea of this approach the same as MUTE: to remove majority records which are close to minority records, thereby making the decision boundary more obvious to a classification algorithm.

MUTE was extended in a follow-up publication ([Bunkhumpornpat et al., 2014](#)). Four subsets of $D_{maj}$ were defined: *noise*, *borderline*, *safe*, and *core*. These subsets are ordered by safe-level. Based on a parameter, a data miner can decide which subsets

<table>
<thead>
<tr>
<th>Method</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomek-Links</td>
<td>(<a href="#">Tomek, 1976</a>)</td>
</tr>
<tr>
<td>NearMiss</td>
<td>(<a href="#">Zhang and Mani, 2003</a>)</td>
</tr>
<tr>
<td>AdaBoost-B</td>
<td>(<a href="#">Altıncay and Ergün, 2004</a>)</td>
</tr>
<tr>
<td>SBC</td>
<td>(<a href="#">Yen and Lee, 2006</a>)</td>
</tr>
<tr>
<td>MUTE</td>
<td>(<a href="#">Bunkhumpornpat et al., 2011</a>)</td>
</tr>
<tr>
<td>MUTE (Extended)</td>
<td>(<a href="#">Bunkhumpornpat et al., 2014</a>)</td>
</tr>
<tr>
<td>Fast-CBUS</td>
<td>(<a href="#">Ölek et al., 2017</a>)</td>
</tr>
<tr>
<td>Cluster-Based Undersampling</td>
<td>(<a href="#">Lin et al., 2017</a>)</td>
</tr>
</tbody>
</table>

**Table 3.4**: Distance-based approaches to mitigate the loss of information problem. Sorted by date.
to include in the sample $D'_{maj}$. Core records are always included in $D'_{maj}$.

There are three variations of a technique called NearMiss (Zhang and Mani, 2003). NearMiss1 first finds the 3 closest minority records for each majority record. From each of these 3 minority records, the distance is measured between it and the majority record. The average of these 3 distances is then calculated. Majority records with the lowest average distance are included in the sample. NearMiss2 is the same as NearMiss1 except instead of finding the 3 closest minority records, it finds the 3 farthest minority records. NearMiss3 is different to both. For each minority record, NearMiss3 samples its $k$ closest majority records. A variant of NearMiss1 called Distant was also proposed. Distant is the same as NearMiss1 except the majority records with the greatest average distance are sampled. NearMiss1, NearMiss3 and Distant did not outperform RUS and NearMiss2 was approximately as good as RUS.

An important concept with NearMiss1 and NearMiss2 is that finding the average distance from a given majority record to all minority records can be incredibly computationally expensive. If there are 1,000 minority records, 9,000 majority records and 50 attributes, then such a calculation would involve 9,000,000 Euclidean distance calculations in 50-dimensional space. NearMiss1 and NearMiss2 deal with this problem in two different ways:

The goal behind NearMiss1 is that the majority records which are close to some of the minority records are chosen. For each majority record, it finds the 3 closest minority records, calculates the distance from the majority record to each of the 3 minority records, and calculates the average of the 3 distances. Therefore, for each majority record, it is reducing the scope of the problem to 4 records regardless of the dataset size.

The goal behind NearMiss2 is that the majority records which are close to all of the minority records are chosen. For each majority record, it finds the 3 farthest minority records, calculates the distance from the majority record to each of the 3 minority records, and calculates the average of the 3 distances. Like NearMiss1, NearMiss2 is reducing the scope of the problem to 4 records, but using the 3 farthest records.

SBC (Yen and Lee, 2006) (underSampling Based on Clustering), finds clusters in the training dataset which it uses to inform its undersampling procedure. First, it
finds $K$ clusters in $D$ using any existing clustering algorithm. Then using Equation 3.8 it calculates how many majority records to sample from each cluster. The clusters are denoted as $G_1, G_2, \ldots, G_K$. For readability the function $\text{ratio}(\cdot)$ is defined which, for a given cluster, computes its number of majority records divided by its number of minority records. The number of majority records to sample from the $i$th cluster is denoted by $S_i$.

$$S_i = (m \times |D_{\text{min}}|) \times \frac{\text{ratio}(G_i)}{\sum_{j=1}^{K} \text{ratio}(G_j)}$$

After determining how many majority records to sample from each cluster, SBC samples them using RUS. Extensions of SBC were also proposed which simply involved using the variations of NearMiss instead of RUS. The author’s experiments showed that the highest F-Measure was achieved when sampling was performing by simply using RUS. There are several other undersampling techniques which also utilise clustering. For example Fast-CBUS (Ofek et al., 2017) was proposed as an extension of SBC.

Rather than cluster the whole dataset, Fast-CBUS only clusters $D_{\text{min}}$. For each cluster, it samples majority records which are inside the cluster’s boundaries. The number of sampled majority records per cluster is equal to the number of minority records in the corresponding cluster. Thus, each cluster has an associated set of records which have an equal class distribution. A classifier is trained on each of these associated subdatasets. When classifying a new record, if it does not fall within any of the boundaries of any cluster, it is classified as majority. However, if it falls within a cluster, a weighted ensemble voting equation is used to determine which classifier to use for prediction. The voting equation favours the classifier whose corresponding cluster centroid is closer to the record.

Compared to SBC and Fast-CBUS, the technique clustering-based undersampling (Lin et al., 2017) is much simpler. This technique finds $|D_{\text{min}}|$ clusters within $D_{\text{maj}}$. The set of cluster centroids are then used as $D'_{\text{maj}}$. This ensures that the sampled dataset has an equal class distribution. Cluster centroids are not real records. Therefore, a variant was also proposed which instead samples the closest majority neighbour of the centroid. This technique is identical to a sampling technique published
13 years prior (Altınçay and Ergün, 2004).

**Evolution-based approaches to mitigate the loss of information problem:**

Evolution-based approaches typically involve weights which are assigned to each majority record. Initially, these weights are all equal to each other. These approaches involve many iterations. During each iteration, a weighted random sample is taken from the original training dataset. After each sample is taken, a classifier is trained on it and evaluated using a performance measure. Well performing samples have their weights increased which results in a higher probability that they are selected in subsequent samples.

The advantage of these approaches is that many potential sample sets are evaluated, typically using the data miners desired performance measure. Therefore, a data miner can directly choose the measure which they want the resulting classifier to be optimised for. For example, if a classifier is desired which achieves a high AUC score, then AUC can be used to evaluate each iteration’s sample.

Many evolution-based approaches are inspired by naturally occurring processes. For example, ACOSampling is based on the behaviour of food-searching ants (Yu et al., 2013). In nature, ants must work together to find pathways which lead from their nest to food. If an ant finds a pathway which leads to food, it will leave behind a pheromone which attracts its fellow ants. The ACOSampling algorithm considers the journey of an ant from nest to food to represent one sample. Along this journey, the ant visits each majority record. Connecting every majority record are two pathways. The 1 pathway and the 0 pathway which represent that the record is chosen in the sample or not respectively. An ant will visit every majority record, but may choose either pathway. Therefore, once an ant has visited every majority record, a sample is formed. Each pathway has a pheromone score which is updated after each iteration based on the sample’s performance.

EUS (García and Herrera, 2009) is another evolutionary approach to the loss of information problem in undersampling. EUS uses the same chromosome representation as ACOSampling. It creates many generations of chromosomes where the fitness function is the performance of the union of that chromosome’s majority records and the full set of minority records. EUS has been combined with the underbagging approach to create the algorithm EUS-Bag (Sun et al., 2018). Similar to the purpose
Chapter 3. Survey on Class Imbalance and Cost-Sensitive Classification

of underbagging, EUS-Bag aims to use all of the majority records for learning. In EUS, only the chosen subset of the majority records are used. However, EUS-Bag builds an ensemble where each iteration uses EUS to find the optimal set of majority records. This results in more of the majority records being used for learning. EUS-Bag was found to out-perform EUS.

**The oversampling-caused overfitting problem:** To create a sample $D'$ of dataset $D$ such that $|D'_i| > |D_i|$, random oversampling must result in duplicate records within $D'_i$. This is harmful to a fundamental goal of classification algorithms: to build classifiers which generalise to unseen data. If a classification algorithm produces classifiers which fail to meet this goal, it is referred to as overfitting. More specifically, the resulting classifiers are too specific to the training data and thus predict poorly on unseen data. By passing duplicates of $r_x \in D$ as training data to a classification algorithm, overfitting is likely to occur. The fundamental goal of oversampling research is to achieve $|D'_i| > |D_i|$ but mitigate overfitting.

This overfitting drawback of ROS was elegantly mitigated by the algorithm SMOTE (Chawla et al., 2002). In the class imbalance literature, the overwhelming majority of proposed methods for achieving $|D'_i| > |D_i|$ are extensions of SMOTE. The paper which it was published in has been heavily influential on the class imbalance literature, accruing approximately 5000 citations. SMOTE, or Synthetic Minority Oversampling TTechnique achieves $|D'_i| > |D_i|$ without sampling any $r_x \in D_i$ more than once. SMOTE creates a sampled $D_i$ such that $D'_i = D_i \cup S_i$ where $S_i$ is a set of new synthetic class $i$ records. Each synthetic record in $S_i$ is generated from a $r_x \in D_i$ and a random neighbouring record of $r_x$. Its attribute values are set to a random point between the corresponding attribute values of the two real records. Many of the extensions to the SMOTE algorithm involve generating synthetic records differently based on the characteristics of the corresponding real record. These extensions are discussed in Section 3.5 because they introduce a different level of bias for each record.

In this section, many sampling techniques were discussed. When a data scientist aims to build a performant classifier using class imbalanced data, a variety of approaches should be trialed before deploying the model to production. For example,
using a cross validation approach, several methods could be evaluated on the available dataset. The parameters in resampling approaches typically control the size of the resulting sampled dataset and it’s imbalance ratio. Therefore, it is important to use a hyper-parameter optimisation approach such as grid search. In the grid search approach, all possible combinations of parameters are used and evaluated to find the best performing combination. The possible combinations can be limited.

**Resampling: All Classes**

Resampling to achieve cost-sensitivity has a strong theoretical foundation. Zadrozny et al (Zadrozny et al., 2003) first described, then proved a *folk theorem*. They used the term folk theorem because it was a theorem that was informally accepted as true within the cost-sensitive classification community (Zadrozny et al., 2003). The folk theorem stated that if a subsample $D'$ was taken from a dataset $D$ using a method which was proportionate to the classification costs, then a cost-insensitive classification algorithm applied to $D'$ would produce a cost sensitive classifier. Resampling-based cost-sensitive algorithms have been proposed which are based on this theorem (Zadrozny et al., 2003; Sheng and Ling, 2007).

The authors then formalised this theorem and proved it to be true. They then designed a sampling algorithm designed to create an appropriate $D'$ called cost-proportionate rejection sampling (Zadrozny et al., 2003). This sampling algorithm is combined with a classification algorithm called bagging (Breiman, 1996) to produce a cost-sensitive classifier. This entire process is called Costing.

Costing was criticised for producing samples which are significantly smaller than the original dataset. Thereby causing a significant loss of information. In response to this, *Cost Proportionate Roulette Sampling* (CPRS) (Sheng and Ling, 2007) was proposed. This sampling approach can generate samples of any desired size. Roulette sampling is an approach inspired by roulette wheels which are found in casinos. A casino roulette wheel is spun such that a ball will land in a random segment of the wheel. Each wheel segment corresponds to a number. The core idea of CPRS is to represent each $r_x \in D$ as a segment, then adjust the probability of landing on each segment proportionate to the cost of misclassifying the corresponding $r_x$. To add a sample from $D$ to $D'$, the roulette wheel is spun, and the corresponding $r_x$ of
the resulting segment is added to $D'$. This process can be repeated as many times as desired thereby solving the small sample problem of Costing. Similar to Costing, CPRS is combined with bagging to produce a cost-sensitive classifier which the authors call a *CSRoulette* classifier.

As mentioned above, CSRoulette was proposed as a response to Costing causing a loss of information. The common problem of losing information due to undersampling has been approached differently in class imbalance literature. For example, it has been demonstrated that subagging (bagging without replacement) can decrease false positive rate (fpr) whilst maintaining the true positive rate. This suggests that the loss of information is mitigated because the positive class is predicted more accurately.

### 3.3.2 Feature Selection

Traditionally, feature selection (also called feature subset selection or feature subsetting) is a technique used to reduce the number of attributes in $D$ with the goal of increasing prediction accuracy (Kohavi and John, 1997; Gheyas and Smith, 2010). It is commonly applied to datasets with a large number of attributes called high dimensional datasets. Some of these datasets can also be class imbalanced. In the datasets where both these issues are present (class imbalance and high dimensionality), algorithm modification and dataset modification are not effective approaches to treating class imbalance (Forman, 2003; Putten and Someren, 2004). Instead, choosing a subset of attributes from the training dataset which are the least affected by class imbalance gives desirable results (Wasikowski and Chen, 2010).

One such method which does this is FAST, or Feature Assessment by Sliding Thresholds. FAST is a measure for evaluating how useful a given attribute is for prediction of the class attribute in class imbalanced datasets. The FAST measure is an approximation of the AUC of a non-class attribute. The authors performed a empirical comparison and found that FAST could produce a close approximation of AUC ten times more quickly. FAST was demonstrated to outperform existing feature selection algorithms which were not designed for class imbalanced datasets. It was not compared against existing class imbalance treatment algorithms.
3.3. Dataset Modification

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Citation</th>
<th>Sampling</th>
<th>Feature Selection</th>
<th>Splitting Criteria Modification</th>
<th>Wrapper Method</th>
<th>Class Level Bias</th>
<th>Record Level Bias</th>
<th>Model Level Bias</th>
<th>Class-Specific Cluster Bias</th>
<th>Ensemble</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOSampling</td>
<td>(Yu et al., 2013)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adasyn</td>
<td>(He et al., 2008)</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BalanceCascade</td>
<td>(Liu et al., 2009)</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Borderline-SMOTE</td>
<td>(Han et al., 2005)</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustering-Based Undersampling</td>
<td>(Lin et al., 2017)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSRoulette</td>
<td>(Sheng and Ling, 2007)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EasyEnsemble</td>
<td>(Liu et al., 2009)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FastCBUS</td>
<td>(Ofek et al., 2017)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRUS</td>
<td>(Tahir et al., 2012)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUTE</td>
<td>(Bunkhumpornpat et al., 2011)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NearMiss</td>
<td>(Zhang and Mani, 2003)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tomek-Links Undersampling</td>
<td>(Tomek, 1976)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partitioning</td>
<td>(Chang et al., 2003)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quasi-Bagging</td>
<td>(Chang et al., 2003)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB Bagging</td>
<td>(Hido et al., 2009)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUS</td>
<td>(Drummond and Holte, 2003)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUSBoost</td>
<td>(Seiffert et al., 2010)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROS</td>
<td>(Drummond and Holte, 2003)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safe-Level-SMOTE</td>
<td>(Bunkhumpornpat et al., 2009)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBC</td>
<td>(Yen and Lee, 2006)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMOTE</td>
<td>(Chawla et al., 2002)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Underbagging</td>
<td>(Barandela et al., 2003)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Summary of sampling methods - The presence of algorithm properties for both class imbalance and cost-sensitive classification algorithms. ✓ indicates that the corresponding algorithm has the corresponding algorithm property.
This survey identified an opportunity to modify FAST such that it can be used as a cost-sensitive feature selection method. This would be applicable to high-dimensional cost-sensitive datasets. Currently, FAST ranks a given subset by its area under the ROC curve. Another type of curve, known as the cost curve (Drummond and Holte, 2006), was proposed as an alternative to ROC curves which accounts for misclassification costs. A full proposal and evaluation of such a modification out of the scope of this thesis. However, if researchers were to explore this concept, it would be a welcome addition to the literature. In this exploration, researchers should investigate how this compares to techniques such as Backtracking Feature Selection (Zhao et al., 2013a) and Addition-Deletion Cost-Sensitive Feature Selection (Zhao et al., 2013b). Both of these techniques take a more direct approach of cost-sensitive feature selection by considering the misclassification cost of each subset.

3.4 Algorithm Modification Properties

As mentioned at the beginning of this chapter, this survey focuses on decision tree based classification. However, other classification algorithms such as logistic regression and neural networks are used extensively today. It is important to note that these techniques also have strategies for introducing cost-sensitivity. Example Dependent Cost-Sensitive Logistic Regression (Bahnsen et al., 2014) is a modification of logistic regression which aims to minimize a cost-based function when calculating the posterior probability of a given record belonging to the positive class. Kukar and Kononenko describe an approach of introducing cost-sensitivity to neural networks by minimizing a cost-function instead of minimizing an error based metric such as squared error (Kukar and Kononenko, 1998).

3.4.1 Splitting Criteria Modification

Two decision tree algorithms have been designed to be insensitive to class imbalance. DKM (Dietterich et al., 1996) was shown to only be weakly insensitive to class imbalance (Cieslak and Chawla, 2008) whereas HDDT (Cieslak and Chawla, 2008) completely ignores the class priors. In this way, HDDT is truly insensitive to class
imbalance. HDDT can be easily summarised by the two algorithms presented in Algorithms 2 and 3. In these algorithms, it is assumed that continuous attributes have been discretized. The notation $D_{a=\beta}$ is used to represent the subset of the dataset $D$ where the value of attribute $\alpha$ is equal to value $\beta$. HDDT was shown to outperform DKM by a small margin but consistently achieve a higher AUC than C4.5 (Quinlan, 2014).

**Input:** Dataset $D$ and threshold $\theta$

1. if $|D| \geq \theta$ then
2.  for each attribute $a \in D$ do
3.    $H_a = \text{Calc}_\text{Hellinger}(D, a)$
4.  end
5.  $b = \max(H)$
6.  for each value $v$ of $b$ do
7.    HDDT($D_{b=v}$, $\theta$)
8.  end
9. end

**Algorithm 2: HDDT**

**Input:** Dataset $D$ and attribute $a$

1. for each value $v \in a$ do
2.    $\text{Hellinger} += (\sqrt{|D_{\text{min},a=v}|/|D_{\text{min}}|} - \sqrt{|D_{\text{maj},a=v}|/|D_{\text{maj}}|)^2}$
3. end
4. return $\sqrt{\text{Hellinger}}$

**Algorithm 3: Calc_Hellinger**

CSTree (Sheng et al., 2014; Ling et al., 2006) is similar to the well known C4.5 decision tree algorithm (Quinlan, 2014). Whereas C4.5 uses gain ratio as its splitting criteria, CSTree uses expected cost. To understand how expected cost is calculated for a given set of records, Equations 3.9 and 3.10 must first be understood. These equations calculate the cost of calculating all records in a given dataset as negative, and positive respectively. These two values are denoted as $C_N$ and $C_P$.

$$C_N = |N| \times C_{TN} + |P| \times C_{FP}$$

(3.9)

$$C_P = |P| \times C_{TP} + |N| \times C_{FN}$$

(3.10)
Chapter 3. Survey on Class Imbalance and Cost-Sensitive Classification

After calculating $C_P$ and $C_N$, Equation 3.11 is used to calculate the dataset’s expected cost which is denoted by $E$. Equation 3.11 is equivalent to the harmonic mean of $C_P$ and $C_N$. When deciding whether or not to split a node, CSTree first calculates the node’s $E$. Then, for a potential split, the cumulative $E$ of its children (denoted by $E_A$) is calculated using Equation 3.12 where $C_{P_i}$ is the $C_P$ for the $i^{th}$ child. Once the $E_A$ of all potential splits has been calculated, the potential split with the lowest $E_A$ is found. If the split’s $E_A$ is less than the parent’s $E$, then it is used to split the parent. The process recursively repeats until no more splits can be found where $E_A < E$.

$$E = \frac{2 \times C_P \times C_N}{C_P + C_N}$$ \hspace{1cm} (3.11)

$$E_A = 2 \times \sum_{i=1}^{k} \frac{C_{P_i} \times C_{N_i}}{C_{P_i} + C_{N_i}}$$ \hspace{1cm} (3.12)

CSForest (Siers and Islam, 2015a; Siers and Islam, 2014) is an extension of CSTree. Instead of producing a single tree, CSForest trains a forest.

3.4.2 Wrapper Methods

Wrapper methods wrap existing classification algorithms in a framework such that their output is modified (Liu et al., 2009). They are also referred to as meta-classification methods (Domingos, 1999). The existing algorithm which is wrapped can be considered a parameter to the wrapper method and is referred to as the underlying algorithm. A major advantage of wrapper methods is that a data miner is free to intelligently select an underlying algorithm which suits the characteristics of the dataset.

Such methods have been used to train cost-sensitive classifiers (induce bias) (Domingos, 1999; Zadrozny et al., 2003; Sheng and Ling, 2007) and to train unbiased classifiers from class imbalanced datasets (reduce bias) (Liu et al., 2009), (Tahir et al., 2012).

Table 3.2 shows that 12 techniques have the wrapper property. Out of these 12 techniques, 3 are cost-sensitive. Two have already been discussed in Section 3.3.1, namely CSRoulette (Sheng and Ling, 2007) and Costing (Zadrozny et al., 2003).
The most well-known wrapper method for cost-sensitive classification is MetaCost (Domingos, 1999).

MetaCost has also been referred to as a relabelling technique because it changes the class values in the original dataset such that a classifier trained on the dataset will be cost-sensitive. MetaCost can be considered as being comprised of three processes: Private-Ensemble-Building, Relabelling and Classification. The private-ensemble-building process of MetaCost requires \( m \) number of samples to be created from the original training dataset with \( n \) records each. The base classifier is trained on each sample. The result is an ensemble classifier. This is not the final resulting classifier.

The relabelling process of MetaCost changes the class labels of the records in the training dataset. This is performed by using a cost-minimisation equation which requires classifier predictions as input. These predictions are provided by the private-ensemble. For each record in each sample, MetaCost uses Equation 3.13 to calculate the probability that record \( x \) belongs to class \( j \), denoted by \( P(j|x) \). In this Equation, \( M_i \) denotes the \( i^{th} \) classifier. The class of record \( x \) is then changed to the class that minimises Equation 3.14. In this Equation, \( C(k,j) \) is the cost of classifying a class \( j \) record as class \( k \) (this value is given in the cost-matrix).

\[
P(j|x) = \frac{1}{\sum_{i=1}^{1} \sum_{i} P(j|x, M_i)} \quad (3.13) \quad \sum_{j} P(j|x)C(k,j) \quad (3.14)
\]

Finally, in the classification process, the base classification algorithm is used to produce the final classifier trained on the relabelled dataset.

Out of the 12 wrapper methods surveyed in this chapter, 9 are designed to handle class imbalance. It is very common to use sampling with wrapper methods, particularly with undersampling (Liu et al., 2009), (Seiffert et al., 2010), (Tahir et al., 2012). As such, 8 of these methods were already discussed in Section 3.3.1. The remaining method Standoff (proposed in Chapter 4) uses a cost-sensitive approach to produce a cost-insensitive classifier from class imbalanced data. This cluster based approach is discussed in Section 3.5.4.
When reviewing all 12 wrapper based methods, it was common to see the same base classifiers recommended or used in their original papers. Table 3.6 presents the number of times each base classifier was used in one of the 12 papers.

<table>
<thead>
<tr>
<th>Base Classifier</th>
<th>Times Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5</td>
<td>7</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>2</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>2</td>
</tr>
<tr>
<td>SVM</td>
<td>1</td>
</tr>
<tr>
<td>Ripper</td>
<td>1</td>
</tr>
<tr>
<td>MetaCost</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.6: The number of times each base classifier was used when implementing a proposed wrapper method. (The implementation in the original proposing paper.)

From Table 3.6 one can see that C4.5 is overwhelmingly the most common base classifier used when implementing a wrapper method. This is likely due to its unstable nature. More specifically, a small change in training data can result in a great change in the resulting model. This is the justification given in the IRUS paper’s evaluation (Tahir et al., 2012). 11 of the 12 wrapper methods are ensemble based approaches where unstable classifiers are desired since they result in high variance (Breiman, 1996). EasyEnsemble and BalanceCascade (Liu et al., 2009) were evaluated using AdaBoost as the base classifier which is an interesting choice since AdaBoost itself typically uses C4.5 as its own base classifier (Liu et al., 2009).

### 3.5 Direct Bias Adjustment

#### 3.5.1 Class Level

The overwhelming majority of techniques for bias adjustment do so at the class level. For example, most cost-sensitive classification techniques adjust bias using a class-dependent cost-matrix (Sheng et al., 2014; Siers and Islam, 2015a; Ling et al., 2006; Domingos, 1999). Furthermore, most class imbalance treatment methods adjust for the difference in distribution between classes. However, there are exceptions in both class imbalance and cost-sensitive learning. For example, ExampleCSDT (Bahnsen et al., 2015) requires a cost-matrix for each record in the training dataset; this results in the bias adjustment occurring at the record level (see Section 3.5.2. Some class
imbalance techniques are used to correct class imbalance between clusters within the training dataset (see Section 3.5.4).

The clear advantage of adjusting bias at the class level is that typically only a single adjustment factor is required prior to classifier learning. For example, in cost-sensitive learning, this adjustment factor is a single cost-matrix (Elkan, 2001). In class imbalance treatment methods, it is usually a desired class distribution such as randomly undersampling until an equal distribution is reached. This simplicity is not achieved through techniques such as ExampleCSDT (Bahnsen et al., 2015) which requires a cost-matrix for each record in the training dataset.

As can be seen in Table 3.2, all but three methods adjust bias at the class level. The three methods which do not adjust bias at the class level are ExampleCSDT (Bahnsen et al., 2015), DKM (Dietterich et al., 1996) and HDDT (Cieslak and Chawla, 2008). The former two of which aim to eliminate the opportunity for bias to occur regardless of the class distribution.

Any class balancing technique or cost-sensitive learning technique which discriminates based on the class distribution and class costs is considered a class level bias adjusting technique. As a result the class level category overlaps heavily with the rest of this survey.

### 3.5.2 Record Level

Most class imbalance and cost-sensitive classification algorithms adjust bias at the class level. However, some researchers have achieved better performance by adjusting the bias for each record differently. These techniques commonly consider neighbouring records when calculating these biases. In the case of cost-sensitive classification, these biases are typically only used when building the classifier. The class level costs are then used when predicting new records.

Recall from Section 3.3.1 the oversampling algorithm SMOTE (Chawla et al., 2002) which creates new synthetic minority records based on existing records. Through the algorithm Adasyn (He et al., 2008), SMOTE was extended by taking into consideration the difficulty of classifying each specific minority record. If the neighbouring records of a minority record are predominantly minority class, then its difficulty is considered to be low. Conversely, if the neighbouring records are predominantly
majority class, then its difficulty is considered to be high. The authors of Adasyn argue that the number of synthetic records created from a specific minority record should be proportional to its difficulty. Adasyn was found to outperform SMOTE. By creating a different amount of synthetic records around each existing minority record, Adasyn induces record level bias.

Similar to Adasyn, there have been several extensions to SMOTE which use the concept of difficulty to apply SMOTE differently to each record. For example, Borderline-SMOTE (Han et al., 2005) defines the concept of dangerous which is fundamentally the same as difficulty. If at least half of the records neighbouring a minority record are also minority records SMOTE is not applied to it. However, if less than half of the records neighbouring are majority records, SMOTE is applied.

A similar approach was taken in Safe-Level-SMOTE (SLS) (Bunkhumpornpat et al., 2009). The authors of SLS first define the safe-level and safe-level-ratio which is similar to difficulty. The safe level of a minority record is the number of minority records within its $k$-nearest neighbours. The safe-level-ratio between a minority record and one of its nearest neighbours, is equal to the safe-level of the minority record divided by the safe-level of the nearest neighbour. Similar to SMOTE, for each minority record, SLS randomly chooses a nearest neighbour. Then a new record is synthetically generated with attribute values between the corresponding attribute values of the minority record and its nearest neighbour. However, SLS is more likely to generate synthetic records closest to the record with the highest safe level. SLS was shown to outperform Borderline-SMOTE and SMOTE on several real datasets. Since SLS oversamples differently for each record, it adjusts the bias at a record level.

Compared to the techniques discussed above, the approach Example-Dependent Cost Sensitive Decision Tree (Bahnsen et al., 2015) (ExampleCSDT for short) requires a large amount of information to be gathered beforehand. This information is a cost-matrix for each individual record in the training dataset. Remember that traditionally, cost-sensitive techniques require a single cost-matrix, however for a dataset with 20000 records, 20000 cost-matrices are required. Each example dependent cost-sensitive cost-matrix corresponds to a particular record. It is represented in the same way as a traditional cost-matrix. These cost-matrices can be generalised into a single cost-matrix. For example, in the original ExampleCSDT paper, the authors use their
algorithm to approach the credit card fraud detection problem. In their dataset, each record represents a credit card transaction. The aim is to predict if a transaction is fraudulent (positive) or not fraudulent (negative). The authors defined a generalised cost-matrix for that problem which had the following costs (note that the $i$ subscript denotes which record the cost corresponds to):

- $C_{TP_i} =$ The administrative cost of investigating a fraud alert.
- $C_{TN_i} = 0$
- $C_{FP_i} =$ The administrative cost of investigating a fraud alert (same as $C_{TP_i}$)
- $C_{FN_i} =$ The amount of the transaction represented by the $i$’th record. This amount is an attribute in the dataset.

Since Example CSDT uses multiple cost-matrices, it requires an alternative definition to Equations 3.10, and 3.9. In other words, it must define equations for calculating $C_P$ and $C_N$, the cost of labelling all records in a dataset as positive or negative respectively. The calculations for $C_N$ and $C_P$ are easiest to understand when represented using pseudocode. This pseudocode is presented in Algorithms 5 and 4 respectively.

**Input:** A dataset $D$, A cost-matrix for each record in $D$

```
1 total = 0
2 for each record $r_i$ $\in$ $D$ do
3    if $r_i$ is positive then
4       total = total + $C_{TP_i}$
5    end
6    else if $r_i$ is negative then
7       total = total + $C_{FP_i}$
8    end
9 end
10 return total
```

**Algorithm 4:** Calculate_C_P

**Input:** A dataset $D$, A cost-matrix for each record in $D$

```
1 total = 0
2 for each record $r_i$ $\in$ $D$ do
3    if $r_i$ is negative then
4       total = total + $C_{TN_i}$
5    end
6    else if $r_i$ is positive then
7       total = total + $C_{FN_i}$
8    end
9 end
10 return total
```

**Algorithm 5:** Calculate_C_N
Example CSDT uses a cost-based impurity measure $I_C(S)$ as defined in Equation 3.15. To calculate the best possible split for a dataset $D$, it calculates the cost-based gain of each possible split using Equation 3.16. (Where $D_0$ and $D_1$ are the left and right splits respectively.)

$$I_C(D) = \min\{C_N, C_P\}$$  \hspace{1cm} (3.15)

$$Gain = I_C(D) - \frac{D_0}{D} I_C(D_0) - \frac{D_1}{D} I_C(D_1)$$  \hspace{1cm} (3.16)

The splits with the lowest possible impurity are used to build the decision tree until no reduction in impurity can be achieved through splitting. This approach is suitable for scenarios where the costs for each record are very different from each other, and the record-specific costs are readily available or calculable from the dataset. It would be interesting to see this approach adapted for approaching class imbalance classification.

Strictly speaking, in reality, there are only two possible costs that can result from a given record. A record which is actually positive can only incur a true positive or false negative prediction. Similarly an actually negative record can only incur a true negative or false positive prediction. However, when dealing with predicting class labels, the class label is not known and the costs must be defined for all four possible predictions.

### 3.5.3 Model Level

An ensemble classifier is a classifier which is comprised of several independent classifiers. The predictions of these classifiers may conflict with each other. Therefore, to predict using an ensemble classifier, a strategy is required to combine multiple predictions into a single prediction. These strategies are referred to as voting (Islam and Giggins, 2011).

Researchers have proposed voting strategies which produce cost-sensitive predictions from an ensemble classifier (Siers and Islam, 2014; Siers and Islam, 2015a).
Similarly, voting strategies have also been proposed which produce unbiased predictions from biased ensemble classifiers (Ofek et al., 2017). Ensembles classifiers which are comprised of decision tree models are referred to as decision forests.

To classify a given record \( x \), the method CSVoting (Siers and Islam, 2015a) finds the leaf which classifies \( x \) for each tree in the decision forest. Then for each leaf, \( C_N \) and \( C_P \) are calculated using Equations 3.9 and 3.10 respectively. The total \( C_N \) and \( C_P \) are found over all leaves and the cheapest classification is used to classify \( x \). An advantage of CSVoting is that it can be used on cost-insensitive decision forests in order to make cost-sensitive predictions. It has been found to decrease the total cost when used with the cost-insensitive decision forest algorithm SysFor (Islam and Giggins, 2011). SysFor uses Gain Ratio (as used in C4.5 (Quinlan, 2014)) to determine the best splitting points to use at the root node of a decision tree using the training data. Then, using each of the best splitting points as the initial split, trains a C4.5 tree on the resulting branches. This results in a forest of trees which all have high accuracy.

An interesting approach to the class imbalance problem is Fast-CBUS (Ofek et al., 2017). This approach first clusters the minority class records in the training dataset. After finding the cluster centroids, each cluster’s bounds are calculated. To do this for a given cluster, the minority record with the farthest distance from the centroid is found. All records (minority and majority) which are within this distance from the centroid are considered part of the cluster. This results in all minority records belonging to a cluster but only some majority records belonging to a cluster. For each cluster, a classifier is trained using the majority and minority records which are within the maximum distance. When classifying a new record, if the record does not fall within a cluster, it is automatically classified as majority. However, if it falls within a cluster, the classifiers on all clusters form an ensemble. To perform voting using this ensemble for a new record \( r \), the probabilities of all classifiers are considered. However, a weight is applied to each classifiers vote. This weight is proportional to the distance between \( r \) and the corresponding classifier’s cluster centroid.
3.5.4 Cluster Level

Consider the subsets of $D$ which contain only records of the same class. Within each of these subsets, clusters can be found which are called class specific clusters (Defined in Chapter 5). If there exists a class specific cluster which has many more records than the other class specific clusters of the same class, then it is a case of within-class imbalance (Japkowicz, 2001). Another name for this is the problem of small disjuncts (Holte et al., 1989).

The method Cluster-Based Undersampling (Lin et al., 2017) first finds the clusters in the majority class records. The number of clusters it finds is set to the number of minority records. Then, it replaces the majority records by the majority cluster centroids. This results in an exactly balanced new dataset $D'$. A strong advantage of this technique is that it only needs to cluster the majority records and not the minority records. This is in contrast to the Standoff technique (Proposed in Chapter 4), which requires both classes to be clustered, thus requiring two clustering steps. The Standoff technique uses the concept of Lanchester’s laws to balance the influence of each class-specific cluster. Lanchester’s laws are a set of formulae for calculating the rate of attrition in military warfare. Standoff uses these formulae to calculate a cost-matrix for each class-specific cluster such that when using these costs to build a classifier, the classifier is balanced. The reason for this technique’s name is that since each cluster has the same influence, a standoff scenario is created.

Two techniques were also discussed, SBC (Yen and Lee, 2006) and Fast-CBUS (Ofek et al., 2017) in Section 3.3.1. These techniques also adjust bias at the cluster level.

3.6 Example of Inter-Field Learning

This section provides an example of how lessons learned in one field can be applied to the other in a beneficial way. It first describes a lesson learned in the class imbalance literature. Then, it describes how this lesson has not been applied to an existing cost-sensitive learning algorithm. Finally, it proposes and evaluates an extension of the existing cost-sensitive learning algorithm by applying the aforementioned lesson.
Lesson Learned in Class Imbalance Literature: Sampling a training dataset $D_T$ such that the resulting sample contains more than one instance of any single record from $D_T$ can result in overfitting when used for training a classifier. This damages the resulting classifier’s ability to generalise to future data. This problem was addressed through the use of synthetic data point generation (Chawla et al., 2002). More specifically, it was used to avoid the record duplication problem in Random Oversampling of the minority class. The proposed remedy to this problem was the SMOTE algorithm.

How CSRoulette Introduces Overfitting: The algorithm CSRoulette (discussed in Section 3.3.1) (Sheng and Ling, 2007) creates samples of a training dataset such that a classifier trained on the sample will be cost-sensitive. The classification algorithm used for training is not cost-sensitive. A disadvantage of CSRoulette is that it can sample the same record multiple times. However, the above-mentioned lesson teaches that this can damage the resulting classifier’s ability to generalise to future data. The inner-process that CSRoulette uses to sample the training dataset is called CPRS or Cost Proportionate Roulette Sampling. CSRoulette produces several samples using CPRS, then trains an ensemble classifier by training a single classifier on each sample. Majority voting is used make predictions using the ensemble.

The Proposed Solution - SCSRoulette and SCPRS: An extension to CSRoulette is proposed named SCSRoulette. The overall process is the same as CSRoulette, however it modifies CPRS such that records from $D_T$ may only be sampled once. If they are chosen to be sampled again, a synthetic record which the same class value is instead created and used as the sample instead. This eliminates the possibility for a record to be sampled more than once. This chapter refers to this sampling algorithm as SCPRS which is short for Synthetic Cost-Proportionate Roulette Sampling. The process for synthesizing a record is inspired by SMOTE (Chawla et al., 2002). The only difference is that the proposed algorithm SynthesizeRecord can synthesize records for any given class. SynthesizeRecord is presented in Algorithm 6.

3.6.1 Experimental Evaluation of the Proposed SCSRoulette
This chapter compares the proposed SCSRoulette against the existing CSRoulette over four datasets available from the UCI machine learning repository (Lichman,
Input: A training dataset $D_T$. 
An existing record to base the synthesized record on $r_x \in D_T$. 
Output: A synthesized record $r$

1. $\text{nearestNeighbours} = \text{getFiveNearestNeighbours}(r_x, D_T)$
2. $\text{randomIndex} = \text{randomInt}(0, 5)$
3. $\text{chosenNeighbour} = \text{nearestNeighbour}[$randomIndex$]$
4. $\text{syntheticValues} = []$; // will contain the synthetic record’s values.
5. foreach attribute $a \in D_T$ do
6. \hspace{1cm} $v_1 = r_x.\text{valueOf}(a)$
7. \hspace{1cm} $v_2 = \text{chosenNeighbour}.\text{valueOf}(a)$
8. \hspace{1cm} $\text{syntheticValues}.append(\text{randomFloat}(v_1, v_2))$
9. end
10. return $\text{Record(\text{syntheticValues}, r_x.\text{classValue()})}$

/* getFiveNearestNeighbours returns a list of five records in the passed dataset which are closest to the passed record based on Euclidean distance. randomInt returns a random integer greater than or equal to the first parameter and less than the second parameter. record.valueOf returns the value of the passed attribute for the record the function was called from. randomFloat returns a random float greater than or equal to the first parameter and less than the second parameter. record.classValue returns the class value of the record it was called from. Record returns a record with the passed values and passed class value. */

Algorithm 6: SynthesizeRecord

2013):

- Dataset 1: Breast-Wisconsin (Diagnostic)
  

- Dataset 2: Breast-Wisconsin (Original)
  

- Dataset 3: Recurrence of Breast-Cancer
  
  https://archive.ics.uci.edu/ml/datasets/Breast+Cancer

- Dataset 4: SpectF - Heart Disease Prediction
  
  https://archive.ics.uci.edu/ml/datasets/SPECTF+Heart

Metadata about the above-mentioned datasets is given in Table 3.7.

The evaluation metric used is the same that is used in the original CSRoulette paper - average cost per record. This is calculated using Equation 3.17. The evaluation is performed using 3 cross-fold validation.
3.6. Example of Inter-Field Learning

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Records</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast-Wisconsin (Diagnostic)</td>
<td>569</td>
<td>32</td>
</tr>
<tr>
<td>Breast-Wisconsin (Original)</td>
<td>699</td>
<td>10</td>
</tr>
<tr>
<td>Recurrence of Breast-Cancer</td>
<td>286</td>
<td>9</td>
</tr>
<tr>
<td>SpectF - Heart Disease Prediction</td>
<td>267</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 3.7: Metadata on the datasets used in this section.

AverageCost = \[
\frac{N_{TP} \times C_{TP} + N_{TN} \times C_{TN} + N_{FP} \times C_{FP} + N_{FN} \times C_{FN}}{N_{TP} + N_{TN} + N_{FP} + N_{FN}}
\] (3.17)

In the experiments, the implementation of CART (Breiman, 2017) in SciKit-Learn (Pedregosa, 2011) is used as the base classifier. More specifically, the DecisionTreeClassifier function is used with the criterion parameter set to “gini” and the max_depth parameter set to 2. Both CSRoulette and SCSRoulette use an ensemble size of 10. The experiment is performed using sample sizes of both 100% and 200% of the original training dataset. The results of these experiments are presented in Figures 3.4, 3.5, 3.6 and 3.7. Figure 3.8 summarises these results by displaying the average results over the four datasets.

![Figure 3.4: SCSRoulette vs CSRoulette for Dataset 1.](image1)

![Figure 3.5: SCSRoulette vs CSRoulette for Dataset 2.](image2)

To be consistent with the original CSRoulette paper, the cost matrices are set as follows: \( C_{TP} = 0 \), \( C_{FP} = 0 \), \( C_{FP} \) is set to the number of positive records in \( D_T \) and \( C_{FN} \) is set to the number of negative records in \( D_T \).

Since there are 4 datasets with two different parameter settings, there are eight different comparisons between SCSRoulette and CSRoulette. In 6 of these comparisons, SCSRoulette achieves a lower average cost than CSRoulette. Out of the two remaining comparisons, one is very close (see Figure 3.7 for sample percentage = 100).
Chapter 3. Survey on Class Imbalance and Cost-Sensitive Classification

Figure 3.6: SCSRoulette vs CSRoulette for Dataset 3.

Figure 3.7: SCSRoulette vs CSRoulette for Dataset 4.

Figure 3.8: Average Results for SCSRoulette vs CSRoulette.

The average results over all four datasets is presented in Figure 3.8. The average results suggest that the proposed SCSRoulette is better at producing low cost classifiers. Therefore, the experiments in this section demonstrate that by lessons learned in one bias adjustment field can benefit techniques proposed in the other bias adjustment field.

3.7 Summary of Experimental Results in the Surveyed Studies

It would be useful to be able to compare all the studied techniques in this survey. However, this would take an extremely long time. As an alternative, this chapter provides Table 3.8. In this table, each row and each column is a technique studied in this chapter. Within a cell, for example, where the row is ACOSampling and the column is RUS, it denotes how ACOSampling compared to RUS in the original ACOSampling publication. Since there is a + in this cell, ACOSampling achieved better performance than RUS according to the ACOSampling publication. Since there is a + in this cell, ACOSampling achieved better performance than RUS according to the ACOSampling publication. The symbols = and − indicate that the technique corresponding to the row achieved equal and lesser performance than the technique corresponding to the column. To limit
the scope of this table, only the techniques compared against in the original papers are included if they are already discussed in this paper.

It’s important to remember that this is simply a summary of the existing studies. Table 3.8 is a useful summary of all the experiments conducted in these papers.

If a fair comparison was to be made between all of these techniques, it would be on a massive scale since all techniques in all papers would be required to be implemented and tested over all the datasets used in each paper. Since this was part of a literature review, the focus was on the discoveries of other studies.

### 3.8 Identified gaps which motivate this thesis

During the survey of the literature, the following gaps were identified which provide the motivation for the rest of this thesis.

#### 3.8.1 Lack of Delineation and Definition of Cost Types

This survey identified and delineated the concepts of balancing costs and domain costs. These two types of costs are useful for class imbalance and cost-sensitivity respectively. However there is no definition of how they can both be used to approach one problem at the same time.

Costs are used as input to many classification algorithms in the form of a cost-matrix (Ling et al., 2006; Sheng et al., 2014; Bahnsen et al., 2015). However, there is no definition of the difference in how they are used. For example, the costs provided by the cost-matrix may be designed to mirror the degree of class imbalance or they may be designed to accurately portray the business cost resulting from the classifiers predictions.

Consider a dataset which is both class imbalance and cost-sensitive such as those which occur in software defect prediction (Sheng et al., 2014). To build a classifier on this dataset and combat both problems, a data scientist is faced with a choice. Shall they ignore the class imbalance and build a cost-sensitive classifier? Shall they ignore the cost-sensitivity and build a class balanced classifier? Shall they ignore both problems and build a non cost-sensitive and class imbalanced classifier? If they
Table 3.8: For any given cell, + indicates that the method corresponding to the row outperforms the method corresponding to the column. The converse is true for -. = denotes that the methods are considered equal in performance. This data is collected from the papers that propose the method corresponding to each row. Cells which are empty indicate that no comparison was made in the paper between the two corresponding methods.
choose to address both issues, shall they address the class imbalance or the cost-
sensitivity first? This thesis proposes an alternative which removes the need to make
the choice. This is done by proposing a classification approach which considers both
the balancing and domain costs as one problem, then addresses it in a sensible way.

**How the thesis addresses this literature gap:**

One of the aims of this thesis is to produce an algorithm which handles both
issues simultaneously. Using our definition of bias adjustment, we now define the
following machine learning task:

**DEFINITION: Balanced Costs Classification** - Classification performed using
an algorithm which combines balancing costs and domain costs into a single cost-
matrix as input. This can also apply to example dependent cost-sensitive classifica-
tion algorithms; in that case, multiple cost-matrices are generated and used as input.

In Chapter 6, a framework for performing balanced costs classification is pro-
posed. Within that chapter, methods for combining balancing costs and domain
costs are explored.

### 3.8.2 Existing Class Imbalance Methods Are Not Designed for Knowl-
edge Discovery

The overwhelming majority of techniques for combating class imbalance are sam-
pling based (Chawla et al., 2002; Tahir et al., 2012; Drummond and Holte, 2003). This results in the training dataset being perturbed. As a result, the trained mod-
els are less suitable for knowledge discovery since the patterns discovered by the
predictive models do not describe the actual training dataset.

One of the goals of this thesis is to produce a classifier which does not perturb
the training dataset, even when the dataset is cost-sensitive and class imbalanced.
This means that the patterns which can be extracted by the classifier describe the all
the historical data, not a subset (Drummond and Holte, 2003) or synthetic superset
(Chawla et al., 2002) of it. Since the approach produced in this thesis is capable of
achieving this, the approach is applied to software defect prediction; a classification
problem which is well-known to be both cost-sensitive (Sheng et al., 2014) and class
imbalanced (Siers and Islam, 2015a).

**How the thesis addresses this literature gap:**
In Chapter 4 of this thesis, a method for producing a balancing cost matrix for each record is proposed. Using this method, a non-cost-sensitive but class balanced classifier is created without sampling or perturbing the original dataset. Later, in Chapter 6, Standoff is used in a proposed framework which combines the balancing costs produced by Standoff with a suitable domain cost matrix. By doing so, a cost-sensitive and class balanced classifier is created which doesn’t sample or perturb the original dataset. Therefore, the knowledge which can be directly and easily extracted from the resulting classifier directly represents the whole available dataset.

The knowledge discovery capability of the proposed method is also demonstrated in Chapter 7 by applying it to the software defect datasets made available by NASA. These datasets describe the source code used in several systems.

3.8.3 Unexplored ability of producing low cost classifiers by taking a unified approach

There are many techniques which have been proposed to address class imbalance (Chawla et al., 2002; Tahir et al., 2012; Bunkhumpornpat et al., 2012). Similarly, there have been many techniques proposed to produce cost-sensitive classifiers Zadrozny et al., 2003; Domingos, 1999. However, there is a shortage of existing literature for the effect on total classification cost when dealing with both issues. This has been addressed in the method BCSForest (Siers and Islam, 2015a) but it was not capable of consistently producing the lowest total cost classifier when compared to other techniques.

How the thesis addresses this literature gap:

The approach produced in this thesis is capable of consistently training the lowest cost classifiers when compared to the studied techniques. This provides evidence that there is merit to the idea of addressing both issues through a unified algorithm. Furthermore, the technique proposed in this thesis is capable of combining multiple balancing cost-matrices together with the domain matrix. A future work item of this thesis may be to further extend the proposed approach to combine multiple balancing cost matrices with multiple domain cost matrices such as those used in example dependent cost-sensitive decision trees (Bahnsen et al., 2015).
3.9 Concluding Remarks

Class imbalance and cost-sensitive learning are both active research fields. Section 3.2.3 described how these two fields share a common goal: adjusting bias in classifiers. This survey aimed to discover areas in each field where the other field could learn from. That is, this chapter aimed to bridge the gap between these two fields to facilitate inter-field learning. For example, the method FAST (see Section 3.3.2) is used to address class imbalance in high dimensional datasets. This is done by using AUC to select a subset of attributes (Chen and Wasikowski, 2008). When viewing both fields from the unified perspective of bias adjustment, it becomes clear that FAST could easily be modified to address cost-sensitivity by replacing area under the ROC curve with area under the cost curve. This was explained in Section 3.3.2.

Another example would be the method CSRoulette (Sheng and Ling, 2007) (see Section 3.3.1). In this technique, the whole dataset is sampled with replacement with the aim of producing a cost-sensitive classifier. Even the authors mentioned in the CSRoulette paper that this can result in overfitting (Sheng and Ling, 2007). Overfitting by replacement sampling is a problem which was addressed in class imbalanced literature by generating synthetic records (Chawla et al., 2002). Therefore, rather than sample by replacement in CSRoulette, overfitting could be avoided by using this synthetic sampling approach. Section 3.6 proved this concept. This chapter defined a generic synthetic record generation algorithm and used it to enhance the existing algorithm CSRoulette into S-CSRoulette. This chapter’s experiments demonstrate that by using this lesson from the class imbalance field in the cost-sensitive field, performance was increased. These examples demonstrate how inter-field lessons can be learned by studying both fields together.

Researchers used to apply undersampling and oversampling to achieve cost sensitivity (Domingos, 1999), (Elkan, 2001). However, this stopped as there became a wealth of cost-sensitive algorithms to choose from. Meanwhile, many advancements were made in both oversampling (Bunkhumpornpat et al., 2009), (He et al., 2008) and undersampling (Tahir et al., 2012), (Liu et al., 2009) which have not been explored for use in building cost-sensitive classifiers. A welcome contribution to the cost sensitive learning literature would investigate and draw insight into the use of advanced
sampling techniques such as Adasyn (He et al., 2008) or IRUS (Tahir et al., 2012) with Elkan’s general approach (Elkan, 2001) to generate cost-sensitive classifiers.

This review of the literature uncovered many undersampling algorithms which were identical (excluding trivial differences). For example, Section 3.3.1 described how Quasi-Bagging (Chang et al., 2003), Underbagging (Barandela et al., 2003), Partitioning (Chang et al., 2003) and EasyEnsemble (Liu et al., 2009) are extremely similar algorithms. Each of these algorithm’s corresponding publications have added to the literature in their own way. Section 3.3.1 discussed this problem by highlighting the many close similarities between the existing algorithms.

The remaining Chapters of this thesis focus on both class imbalance and cost-sensitive learning. Chapter 6 is an integral part of this thesis which converts the proposed approach to class imbalance proposed in Chapter 4 into a cost-sensitive classification method. The approach proposed in Chapter 6 is heavily influenced by the survey completed in this Chapter. This is because it manipulates the cost-matrix to deal with both cost-sensitivity and class imbalance, thereby using the cost-matrix as a generalised bias-adjustment technique. More details can be found in Chapter 6.
Part II

Classification in Imbalanced Data:
Cost-Insensitive
Chapter 4

Standoff Balancing: Classification
in Class Imbalanced Datasets

Resampling can be applied to a dataset to combat the class imbalance problem. However, this approach results in a classifier which is trained on a dataset with either missing (Drummond and Holte, 2003), duplicate (Drummond and Holte, 2003), or synthetic records (Chawla et al., 2002). This thesis focuses on decision forests due to their knowledge discovery capabilities. Therefore, it is useful if the knowledge extracted from each decision tree accurately describes the full dataset that has been provided. This chapter proposes a technique which achieves competitive performance when compared to other techniques. This is done without changing the original dataset, thus increasing its suitability for knowledge discovery.

The work performed in this chapter addresses Research Question 1. The proposed algorithm produces a classifier which treats class imbalance without affecting the original training dataset. Chapter 6 utilises the algorithm proposed in this chapter to produce class balanced, cost-sensitive classifications without affecting the original training dataset.

4.1 Introduction

During the learning process, a classifier observes the patterns in the training dataset $D_T$. Based on these patterns, the classifier may then classify records that do not have
class values. The performance of a classifier is dependent on multiple factors. One factor is the quality of $D_T$. Various issues may negatively affect the quality of $D_T$. This chapter addresses the class imbalance issue (He et al., 2008).

Consider the task of training a classifier which classifies patients as either cancerous or non-cancerous. That is, the class attribute’s value is either cancerous or non-cancerous. Assume that $D_T$ contains records of 90 non-cancerous patients but only 10 records of cancerous patients. $D_T$ is considered imbalanced because there are many more records of one class than the other class. The classifier trained on $D_T$ will likely be biased towards classifying patients as non-cancerous. This is unacceptable since one or more cancerous patients might forgo life-saving treatment. Class imbalance treatment methods can be used to balance $D_T$. For example, each record of a cancerous patient could be duplicated, and two thirds of the non-cancerous patients are removed from $D_T$. This would result in a dataset of 30 cancerous patients and 30 non-cancerous patients. Thus, the resulting classifier will not be biased. Another approach could involve attaching weights to each patient. The weight of a cancerous patient would be higher (9 times) than the weight of a non-cancerous patient. This chapter creates an analogy between the class imbalance problem and war. By creating this analogy, it is made possible for military strategies to be used as class imbalance treatment methods.

Imbalance may also occur in military warfare. That is, consider two forces that are at war with one another. If one is considerably larger than the other, then it is an imbalanced battle. This study presents an analogy between class imbalance and war. By creating this analogy, it is possible to apply military strategies to the class imbalance problem. The usefulness of this analogy is demonstrated by proposing a novel class imbalance treatment method Standoff-Balancing which is based on a well known military strategy. By doing so, a technique from a highly developed literature is applied to a comparatively immature problem area. This chapter’s experiments demonstrate that the proposed treatment method results in higher performing classifiers than existing treatment methods. The demonstrated superiority of the proposed method might encourage more research into the incorporation of military strategies in data mining and machine learning.
4.2 Main Contributions

Figure 4.1 illustrates the major steps in the proposed Standoff Balancing algorithm. This figure provides an overview of the algorithm.

The rest of this chapter is structured as follows. Section 4.3 introduces the related military strategy. Section 4.4 first introduces the analogy between class imbalance and war. The proposed analogy between class imbalance and war is presented in Section 4.4.1. The steps in the proposed algorithm Standoff-Balancing are presented in Section 4.4.2. Then the proposed method Standoff-Balancing is presented. The proposed technique is compared against four existing methods over five datasets in Section 4.5. The concluding remarks are provided in Section 4.6.

4.2 Main Contributions

- By presenting an analogy between class imbalance and war, the use of military strategy in class imbalance is made possible.

- A class imbalance treatment method is proposed based on the extension of a well known military strategy.

- The proposed method’s AUC is compared against 4 existing methods on 5 datasets.
Chapter 4. Standoff Balancing: Classification in Class Imbalanced Datasets

4.3 Lanchester’s Laws

Lanchester’s laws (Lanchester, 1956) help military strategists to understand the outcome of a battle between two forces. For example, consider two armies. Army B has 6 soldiers and army A has 23 soldiers. By using Lanchester’s linear law, assume that each soldier can only fight one other soldier at a time. This simply means that the size of both armies decreases one at a time until one army is completely wiped out. Thus, Lanchester’s linear law would suggest that for army B to have an equal force to army A there are three solutions. The first is increasing its size to 23. The second is increasing its firepower to be \( \frac{23}{6} \) times stronger than army A’s firepower. The third is a combination of both. Lanchester’s linear law is used for ancient combat, that is, using simple weapons such as swords and spears.

However, in modern combat soldiers do not always fight one-on-one. Thus, Lanchester also proposed a variant for modern combat called Lanchester’s squared law. In the squared variant, army B will need to increase its firepower to be the square of the division of the army A’s size by army B’s size. Thus, army B will need to increase its firepower by a factor of \( \left(\frac{23}{6}\right)^2 \). In practice, rather than use the linear or squared law, 1.5 is used as the exponent (Simpkin, 1988). Therefore, in order to fully emulate what is done in military practice, 1.5 is also used as the exponent in this chapter.

4.4 The Proposed Method

4.4.1 Analogy Between Class Imbalance and War

War and class imbalance share many similarities. In war, two factions engage in an ongoing fight. Within a training dataset \( D_T \), the majority class records \( M \) and minority class records \( N \) are “fighting” for adequate representation. In this fight, each record represents a soldier. In war, each faction may have multiple armies stationed at different locations. Within \( M \) and \( N \), groups of records which have similar values may exist. These groups are commonly referred to as clusters. In this analogy, clusters are considered to be armies. In an army, each soldier is armed with weaponry. Some soldiers may have superior weaponry to others. This is similar
to how some records may have higher weights than other records in cost-sensitive classification. These four key similarities are the core of the analogy.

Consider a situation in which two opposing armies are of equal strength. If these armies fight each other, almost all soldiers from both parties are likely to die. Therefore, neither party wishes to provoke the other party since it will result in mutually assured destruction. This situation is known as a standoff. If $M$ and $N$ are considered to be factions at war with each other, then creating a standoff situation will prevent $M$ from fighting $N$ for adequate representation.

4.4.2 Standoff-Balancing

The proposed method Standoff-Balancing aims to balance the impact of each cluster within the dataset. To do this, Lanchester’s laws are used to calculate the required increase in firepowers for minority class records. The pseudocode for Standoff-Balancing is presented in Algorithm 24. Standoff Balancing is split into three steps. They are explained as follows:

**Step 1 - Preprocessing:** For every attribute in the dataset, the values are normalized to be between 0.0 and 1.0. The reason for this is to make sure that each attribute has equal impact when calculating distance. The dataset is then split into two datasets, one comprising of minority records, and the other of majority records.

**Step 2 - Firepower Calculation and Cost Matrices Generation:** The majority and minority datasets $M$ and $N$ are then clustered independently. The discovered clusters can now be considered as armies within the minority faction and majority faction. By considering the clusters in this way, Lanchester’s laws can be applied to calculate the required increase in firepower for every record within that cluster. Clusters that are far apart from each other are easier to differentiate by a classifier. Therefore, when calculating the strength of the majority class compared to a minority cluster, rather than just take the count of records in the opposing majority clusters, the count is multiplied by the distance. This way, clusters which are farther away have less impact on each other.

Equation 4.1 is used to calculate the required firepower $F_i$ for each minority cluster $N^G_i$ in the set of minority clusters $N^G$. Each minority cluster is balanced against the sum of the number of records in the majority clusters. However, the euclidean
**Input:** A training dataset $D_T$.
A cost-sensitive classification method which accepts cluster-specific cost-matrices $L$.
A clustering method $G$.

**Output:** A balanced classifier.

1. **Step 1: Preprocessing**
   - foreach $A_k \in D_T$ do
     - Normalize($D_T$, $A_k$, 0, 1) /* Normalize is a function which normalizes all values of $A_k$ in $D_T$ to be between 0 and 1. */
   - $M \leftarrow \{D_T : \text{IsMajority}(D_T)\}$ /* IsMajority returns true if the record is majority. */
   - $N \leftarrow \{D_T : \text{IsMinority}(D_T)\}$ /* IsMinority returns true if the record is minority. */

2. **Step 2: Firepower Calculation and Cost Matrices Generation**
   - /* $M^C$ and $N^G$ will hold the cost-matrices for each majority and minority cluster. */
   - $M^C \leftarrow []$
   - $N^G \leftarrow []$
   - $M^G \leftarrow G(M)$
   - $N^G \leftarrow G(N)$
   - foreach $M^G_j \in M^G$ do
     - $M^F_j \leftarrow (\sum_{i=1}^{[M^G_j]} |(M^G_j| \times (1-\text{dist}(S^M_j, S^N_i))) |N^C_i|^{-1.5})$
     - $M^C_j \leftarrow \{C_{TP} : 0, C_{TN} : 0, C_{FP} : 1, C_{FN} : 1\}$
   - end
   - foreach $N^G_i \in N^G$ do
     - $N^F_i \leftarrow (\sum_{j=1}^{[N^G_i]} |(M^G_j| \times (1-\text{dist}(S^M_j, S^N_i))) |N^C_i|^{-1.5})$
     - $N^C_i \leftarrow \{C_{TP} : 0, C_{TN} : 0, C_{FP} : 1, C_{FN} : M^F_j\}$
   - end

3. **Step 3: Classifier Training**
   - /* Depending on the classification method used, the cluster specific costs may need to be unpacked to record specific costs. */
   - return $L(D_T, M^C, N^G)$

**Algorithm 7: Standoff Balancing**
distance from the minority cluster to the majority cluster is used as a weight for each majority cluster’s size. The distances from the minority cluster’s centroid $S_N^i$ to each majority cluster’s centroid $S_M^j$ is normalized such that the farthest centroid is 1.0 and the closest is 0.0. Thus, the farther a majority cluster is to a minority cluster, the less firepower is required.

These firepowers can then be used to create cost-matrices for the minority clusters by using the minority cost-matrix in Table 4.1. Each majority record uses the standard majority cluster cost-matrix shown in Table 4.1. Finally when training the classifier, the corresponding cost-matrix is used for each record.

$$F_i = \left( \sum_{j=1}^{|M^G_j|} (|M^G_j| \times (1 - dist(S_M^j, S_N^i))) \right)^{1.5}$$ (4.1)

**Step 3 - Classifier Training:** Finally when training the classifier, the corresponding cost-matrix is used for each record. The core component of Standoff-Balancing is to create cluster-specific weights to use in cost-sensitive classification. Thus, standoff-balancing may be used with most cost-sensitive methods given a small adjustment from a fixed cost-matrix to cluster-specific cost-matrices as input. Threshold methods handle costs a posteriori (Hernández-Orallo et al., 2012), and therefore could handle Standoff-Balancing easily.

Another option for the classification algorithm is a cost-sensitive example-dependent algorithm (Bahnsen et al., 2015). These algorithms allow the data miner to specify a cost-matrix for each record in the training dataset. Therefore, to use such an algorithm, the cluster specific costs can be unpacked such that each record has a cost-matrix which is the same as the cost-matrix for its assigned cluster.

A cluster’s centroid is a record whose values are the average of the records within that cluster.

<table>
<thead>
<tr>
<th>Minority Cluster</th>
<th>Majority Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{TP} = 0$</td>
<td>$C_{TP} = 0$</td>
</tr>
<tr>
<td>$C_{FP} = 1$</td>
<td>$C_{TP} = 1$</td>
</tr>
<tr>
<td>$C_{FN} = F_i$</td>
<td>$C_{FN} = 1$</td>
</tr>
<tr>
<td>$C_{TN} = 0$</td>
<td>$C_{FP} = 0$</td>
</tr>
</tbody>
</table>

Table 4.1: The Proposed Minority and Majority Cluster Cost-Matrices
Table 4.2: Number of Majority and Minority records in each dataset.

<table>
<thead>
<tr>
<th>Pima</th>
<th>glass0</th>
<th>haberman</th>
<th>ecoli1</th>
<th>BreastCancer</th>
</tr>
</thead>
<tbody>
<tr>
<td># Minority</td>
<td># Majority</td>
<td># Minority</td>
<td># Majority</td>
<td># Minority</td>
</tr>
<tr>
<td>268</td>
<td>500</td>
<td>70</td>
<td>144</td>
<td>81</td>
</tr>
</tbody>
</table>

4.5 Experimental Results

The proposed method is compared against four existing methods over five real world datasets. Overall accuracy or error rate are not useful metrics for evaluating classifiers built from class imbalanced datasets. Therefore the area under the receiver operating characteristic curve (AUC) is used due to its popularity in the class imbalance literature. C4.5 (Quinlan, 2014), MetaCost (Domingos, 1999) and k-means (Lloyd, 1982) are used as the classifier, cost-sensitive method, and clustering method respectively. These methods were chosen for their popularity within their respective area of literature. Since k-means is highly dependent on correct parameter settings, EM (Dempster et al., 1977) with k-means is used.

Information on the four existing methods that are compare against are now provided. Namely, CSTree, SMOTE, Adasyn and MetaCost. 10 fold stratified cross validation is used and the folds used for each method are the same. For SMOTE and Adasyn uses $k = 5$ since it is highly popular within the literature. SMOTE sets the percentage to 200% as suggested by the experiments by Chawla and Bowyer (Chawla et al., 2002). Due to the randomness in SMOTE and Adasyn, the average of 3 independent runs is presented. The WEKA (Hall et al., 2009) implementations of SMOTE, MetaCost and EM with k-means are used. MetaCost uses the default parameters specified in its WEKA implementation. Pruning is turned off in CSTree to build deeper trees which achieve better performance as suggested in the literature (Siers and Islam, 2015a).

Table 4.2 provides the number of minority and majority records in each dataset.

The proposed technique Standoff-Balancing is referred to as Standoff in Table 4.3 in order to conserve space. Five datasets available from KEEL (Fernández et al., 2007; Fernández et al., 2017) and the UCI (Lichman, 2013) machine learning repository are used. The first four are from KEEL whereas the BreastCancer dataset is from the UCI machine learning repository.

The results of the comparison are presented in Table 4.3. They are also presented
4.5. Experimental Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Original</th>
<th>CSTree</th>
<th>SMOTE</th>
<th>Adasyn</th>
<th>MetaCost</th>
<th>Standoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pima</td>
<td>0.735</td>
<td>0.72</td>
<td>0.740</td>
<td>0.700</td>
<td>0.725</td>
<td>0.757</td>
</tr>
<tr>
<td>glass0</td>
<td>0.77</td>
<td>0.82</td>
<td>0.817</td>
<td>0.764</td>
<td>0.784</td>
<td>0.836</td>
</tr>
<tr>
<td>haberman</td>
<td>0.604</td>
<td>0.572</td>
<td>0.627</td>
<td>0.641</td>
<td>0.646</td>
<td>0.647</td>
</tr>
<tr>
<td>ecoli1</td>
<td>0.924</td>
<td>0.899</td>
<td>0.925</td>
<td>0.877</td>
<td>0.866</td>
<td>0.929</td>
</tr>
<tr>
<td>BreastCancer</td>
<td>0.939</td>
<td>0.943</td>
<td>0.935</td>
<td>0.901</td>
<td>0.945</td>
<td>0.952</td>
</tr>
</tbody>
</table>

FIGURE 4.2: AUC Comparison of the Methods (Higher the Better) visualized. Those who are reading this in black-and-white are referred to Table 4.3 which presents the same information but in further detail.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pima</td>
<td>0.735</td>
</tr>
<tr>
<td>glass0</td>
<td>0.77</td>
</tr>
<tr>
<td>haberman</td>
<td>0.604</td>
</tr>
<tr>
<td>ecoli1</td>
<td>0.924</td>
</tr>
<tr>
<td>BreastCancer</td>
<td>0.939</td>
</tr>
</tbody>
</table>

In the results, one can observe that some techniques on some datasets did not perform better than the original. In this case, original is the original dataset with C4.5 applied to it. Recall from Chapter 3 that the performed experiments demonstrated that performance is not always improved when using a class imbalance treatment method. When comparing a set of methods for classifying a class imbalance dataset, some methods will be better than others. Naturally, some of these methods will not perform as well as C4.5. When deploying a class imbalance treatment method, it is important to choose a method which will consistently provide better results than using a non-class imbalance treatment method such as C4.5 on the original dataset.
Table 4.4 compares the execution time of Standoff and the other compared methods. The datasets used in this experiment are used in the following Chapter. For details on these datasets, the reader should refer to Chapter 6. This experiment highlights how Standoff has a much higher execution time than the compared methods. The following chapter of this thesis addresses this weakness.

**Table 4.4: Execution Time Comparison of the Methods - Seconds to complete a 10x10 cross fold validation**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Original</th>
<th>CSTree</th>
<th>SMOTE</th>
<th>Adasyn</th>
<th>MetaCost</th>
<th>Standoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1'</td>
<td>0.309</td>
<td>1.005</td>
<td>0.458</td>
<td>1.403</td>
<td>0.913</td>
<td>27.620</td>
</tr>
<tr>
<td>MC2'</td>
<td>0.258</td>
<td>0.341</td>
<td>0.267</td>
<td>1.086</td>
<td>0.573</td>
<td>89.03</td>
</tr>
<tr>
<td>KC3'</td>
<td>0.262</td>
<td>0.542</td>
<td>0.252</td>
<td>0.883</td>
<td>0.683</td>
<td>19.778</td>
</tr>
<tr>
<td>MW1'</td>
<td>0.298</td>
<td>0.635</td>
<td>0.253</td>
<td>0.628</td>
<td>0.742</td>
<td>29.075</td>
</tr>
</tbody>
</table>

Figure 4.3 compares the improvement of Standoff over Original against the percent of records within the dataset that are minority. The figure shows that the best improvement was found in the datasets with 26.57% and 32.71%.

**Figure 4.3: Standoff Improvement over Original vs Percent Minority**

### 4.5.1 Sensitivity to Outliers

Since Standoff uses clustering, it is important to address how Standoff deals with outliers. To investigate this, the Pima dataset was altered three times to include various levels of outliers. This was achieved by duplicating a random sample of
records within the Pima dataset but with all values multiplied by 100. This was performed three times, with 5%, 10%, and 15% added outliers.

Figure 4.4 shows the AUC of Standoff on each of these created datasets. The comparison illustrates how the more outliers there are, the lower the AUC of Standoff. This provides evidence that Standoff is indeed negatively affected by outliers.

![Figure 4.4: Standoff's AUC on the Pima datasets that have 0%, 5%, 10%, and 15% added outliers.](image)

### 4.5.2 Sensitivity to High Dimensions

Similar to the previous subsection, Standoff is potentially susceptible to high dimensions since it uses clustering. This section investigates the effect that high dimensions have on Standoff. This experiment uses four versions of the Arcene dataset (Lichman, 2013). All version of the Arcene dataset removed 68 of the minority records to create a class imbalanced dataset. The datasets use 50, 100, 150, and 200 of the non-class attributes respectively.

Figure 4.5 shows the AUC of Standoff on each of these created datasets. The comparison indicates that higher dimensions do not have a negative effect on Standoff.
Chapter 4. Standoff Balancing: Classification in Class Imbalanced Datasets

0.8132 0.8462 0.8392 0.8459
AUC
0.65 0.7 0.75 0.8 0.85
Arcene-50 Arcene-100 Arcene-150 Arcene-200

4.6 Conclusion

This chapter presented an analogy between war and class imbalance. By doing it is made possible to apply military strategies to the class imbalance problem. The work presented in this Chapter provides an answer to Research Question 1. This chapter proposed a novel class imbalance treatment method Standoff-Balancing which uses Lanchester’s laws to create a balance in strength between the clusters within a dataset. An experimental comparison on five publicly available real world datasets suggests the efficacy of the proposed technique. The proposed method’s efficacy was demonstrated by two different factors. First, it achieved higher AUC than the existing techniques on all five datasets. Secondly, it was the only method to consistently achieve a higher AUC than the original dataset. This chapter’s experiments not only demonstrate the efficacy of the proposed method, but also the usefulness of the presented analogy. The usefulness of this analogy suggests that further research into the application of military strategy to the class imbalance problem has strong potential.

The proposed algorithm Standoff Balancing fulfils an important milestone towards answering Research Questions 2 and 3. For the former, Standoff balancing
acts as a starting point because class imbalance is addressed. In Chapter 6, Standoff is utilised within a framework to produce cost-sensitive classifications whilst retaining its ability to avoid the negative affects of class imbalance. For the former, Standoff does not affect the original training dataset. Therefore, the knowledge extracted by the Standoff classifier directly reports the original data. This advantage is utilised for knowledge discovery in Chapters 6, 7 and 8.

A weakness of Standoff Balancing is that it requires a clustering algorithm to be run twice on the dataset; once on the positive class and once on the negative class. High performance clustering algorithms such as Affinity Propagation (Frey and Dueck, 2007) and GenClust (Rahman and Islam, 2014) do not scale well. For example, both of these algorithms have quadratic complexity. This poses an issue for Standoff Balancing because extensive experimentation will require a large time investment. This issue is addressed in the following chapter.
Chapter 5

RBClust: Fast Discovery of Class Specific Clusters

The previous chapter proposed an algorithm Standoff which provides an answer to Research Question 1. It requires the use of a clustering algorithm which must be performed twice, once for each class in the dataset. Typically, clustering is a computationally expensive activity. Consequently, clustering algorithms can take a long time to execute. This problem is doubled in Standoff because clustering must be performed twice. To account for this weakness of Standoff, this chapter designs an algorithm with the following aims in mind. First, it should perform quickly. This allows us to experiment with Standoff more efficiently. Second, it should not need to partition the dataset per class and run on each partition. The technique proposed in this chapter achieves both of these aims.

To explain this visually, the overview of the major steps in Standoff are illustrated in Figure 5.1. Figure 5.2 illustrates how these steps change when RBClust is used to discover the clusters instead. When compared together, these figures highlight how RBClust is used to simplify Standoff. This simplification results in execution time savings for the Standoff algorithm.

The work presented in this chapter allows us to experiment with the Standoff Balancing algorithm without requiring such a large time investment. This allowed the research of this thesis to progress and allowed further modification the proposed work to produce cost-sensitive classifications (see Chapter 6). Therefore, the work

---

The work presented in this chapter was published under the title: "RBClust: High quality class-specific clustering using rule-based classification" in the proceedings of the 2016 European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, published online by i6doc, pp 593-598. ERA 2010 Rank B
Chapter 5. RBClust: Fast Discovery of Class Specific Clusters

This chapter indirectly benefits this thesis’ exploration into Research Questions 2 and 3 and enhances the answer to Research Question 1.

5.1 Introduction

A class-specific cluster (CSC) is a cluster in which each record has the same class label. Some data mining algorithms (Japkowicz, 2001) (Nickerson et al., 2001) require the discovery of CSCs. This includes Standoff which was proposed in the previous chapter. However, these algorithms use traditional clustering methods which are not designed for finding CSCs. Traditional clustering methods can often be slow due to high numbers of distance calculations. This chapter proposes RBClust, a fast rule-based method for finding high quality CSCs. RBClust uses a rule-based method such as C4.5 (Quinlan, 2014) to find the patterns in the dataset. Then the clusters are
extracted from the rules. Then to avoid potential issues such as class imbalance, noise, and overfitting, some clusters are merged. Due to the design of RBClust, the number of distance calculations needed is much lower, so computation time is kept low. By leveraging the information of the class-labels, RBClust finds high quality CSCs. The rest of this study is organised as follows. Section 5.3 gives a briefing of the basic concepts and related work. Section 5.4 presents RBClust. Section 5.5 provides a discussion of this chapter’s empirical comparisons between RBClust and the chosen existing methods. Finally, Section 5.6 presents the concluding remarks.

5.2 Main Contributions

- This chapter demonstrates empirically that the proposed method can provide clusters with similar or higher quality than the existing methods.

- Computation time of the proposed method is empirically shown to be much lower than some existing methods.

5.3 Related Work

Classification is the task of determining the class label \( L_i \) of a record \( x \) from a predefined set of classes \( L = (L_1, L_2, \ldots, L_3) \). For example, in medical diagnosis, \( x \) can represent a patient and \( L \) could be a set of possible diagnoses. That is, \( L = (\text{“Asthma”, “Diabetes”, “Cancer”, “Healthy”}) \).

Classifiers are models which can intelligently assign a \( L_i \) to \( x \). Building a classifier requires two things: a dataset \( D \), and a classification method \( m \). One type of classifier model is the rule-based classifier. A rule-based classifier is comprised of a set of rules \( R \). Each \( R_j \in R \) is a sequence of conditions and also has an associated \( L_i \). The \( L_i \) for \( x \) is chosen by checking which \( R_j \in R \) is satisfied by \( x \). \( L_i \) is then taken as the class label of \( R_j \). To build a rule-based classifier, \( D \) must contain records similar to \( x \) which have known class labels, and \( m \) must be a rule-based classification method such as C4.5 (Quinlan, 2014) or CSForest (Siers and Islam, 2014)(Siers and Islam, 2015a).
Clustering is the task of finding groups (known as clusters) of similar records within a dataset $D$. The set of found clusters $C$ is often called the clustering solution. Unlike classification, clustering typically does not require $D$ to have known class labels. Continuing the previous example, if all $x \in D$ where $L_i = “Asthma”$ were taken, a new dataset $D'$ would result which does not have class labels. A clustering method may find two clusters within $D'$. One where each patient was diagnosed with asthma since birth and one where the diagnosis occurred after pneumonia. These two clusters would be class-specific to “Asthma”. The centroid of a cluster is the mean of all records within a cluster.

Whereas classifiers may be evaluated by testing the accuracy on a separate testing dataset, a clustering solution is harder to evaluate. There exist many metrics for evaluating clustering solutions. These metrics often require the calculation of distances between records resulting in a high computational cost. For example, the silhouette coefficient (Rousseeuw, 1987) metric requires the distance to be calculated between every possible $\{(x_i, x_j) | x_i \in D, x_j \in D\}$. Many clustering methods require potential clustering solutions to be evaluated multiple times. This evaluation step contributes heavily to the computation time of clustering methods. State-of-the-art clustering methods may even take several hours to run (Rahman and Islam, 2014).

K-means (MacQueen, 1967) is a clustering method which first randomly chooses $k$ number of records where $k$ is user-defined. These chosen records are called seeds. Each record in $D$ is then assigned to its nearest seed such that $k$ number of clusters are formed. The centroids of these clusters are chosen as the new seeds, and the process is repeated until some termination condition or a maximum number of iterations is reached. The main disadvantage of k-means is that the user needs to guess the number of clusters $k$ in the dataset. If the version of k-means uses a termination condition, then the condition must be checked every iteration. This can become time consuming. Several algorithms such as Affinity Propagation (Frey and Dueck, 2007) (AP) and GenClust (Rahman and Islam, 2014) do not require the user to estimate the number of clusters in $D$. However, these algorithms typically have high-complexity (GenClust and AP both have quadratic complexity).
5.4 The Proposed Method: RBClust

Section 5.3 highlighted two main disadvantages with the reviewed existing methods: long computation time and the reviewed existing methods are not designed specifically for finding CSC. Therefore, they do not make use of the overall dataset for finding the CSCs. This chapter proposes a rule-based clustering method RBClust which aims to overcome these disadvantages. The pseudocode for RBClust is shown in Algorithm 8. The steps of RBClust can be described as follows:

- **Step 1: Rule Discovery** Build a rule based classifier over the whole training dataset $D$. In Algorithm 8 the classifier choice is the parameter $m$. The rules need to be extracted from the classifier, that is, the results of applying $m$ to $D$. Later, in Section ?? the C4.5 (Quinlan, 2014) algorithm is chosen for $m$. C4.5 uses a pruning step to avoid a classifier that is too specific to the training data (known as overfitting). However, since a classifier with high accuracy on $D$ is desired, pruning was turned off.

- **Step 2: Record Sorting** A tri-dimensional array $X$ is used to store the records where $X_{ijk}$ is the $k^{th}$ record which follows the $j^{th}$ rule and belongs to the $i^{th}$ class. The simplest way of performing this step is to classify all records from $D$ using the classifier.

- **Step 3: Cluster Formation** Using each $X_{ij} \in X$, a data structure required for a cluster is built and stored as $C_{ij} \in C$. During this step, empty sets of $X_{ij}$ are ignored (where no $i^{th}$ class records followed the corresponding rule).

- **Step 4: Cluster Merging** There are three problems that could be present after using Steps 1-3. Firstly, $D$ could have an imbalanced ratio of records in each class. For example, there could be 200 records with class label $L_1$ and 40 records with $L_2$. This is a well-known problem in classification called class-imbalance. If $D$ is class imbalanced, RBCClust has many more records to find clusters specific to $L_1$ than clusters specific to $L_2$. Secondly, the clusters in $C$ could be too specific to $D$ which can cause the clustering solution to have poor

---

The classifier must produce rules which include all training records and are mutually exclusive. C4.5 is an example.
generalization on future unseen records. This is known as cluster overfitting. Thirdly, sets of $X_{ij}$ could have a very small number of records. Many other algorithms would not consider this a cluster, and instead treat it as a noisy record. To avoid these problems, a parameter $\theta$ is introduced. During this step, RBClust checks each $C_{ij} \in C$ to see if it has a minimum number of records. If it does not, all records from $C_{ij}$ are moved to the nearest cluster (Lines 25-29). The minimum number of records $S_{i}^{\text{min}}$ for an $i^{th}$ class cluster is calculated in Line 23 by multiplying the total number of $i^{th}$ class records in $D$ by $\theta$. This combats each of the previously mentioned problems.

Due to the merging step in RBClust, it avoids outliers. If there are few outliers in a group, then they will be merged into the nearest cluster. If there are many outliers
in a group, then they will be rightfully considered as a cluster.

**Input:** A dataset \( D \), a rule-based classification method \( m \) (default: C4.5), minimum cluster percentage \( \theta \) (default: 0.02)

**Output:** A set of clusters \( C \) where \( C_{ij} \) is the \( j^{th} \) cluster specific to the class with index \( i \)

**Step 1: Rule Discovery**

1. Let \( R \) be a set of rules;
2. \( R \leftarrow \text{RetrievalRules}(D, m) \);
3. end

**Step 2: Record Sorting**

1. Let \( X \) be a set of records where \( X_{ijk} \) is the \( k^{th} \) record which follows the \( j^{th} \) rule and belongs to the \( i^{th} \) class;
2. Similarly, let \( X_{ij} \) be the set of records which follow the \( j^{th} \) rule and belong to the \( i^{th} \) class;
3. \( X \leftarrow \text{ClassifyRecords}(D, R) \);
4. end

**Step 3: Cluster Formation**

1. foreach \( X_{ij} \in X \) do
2. if \( \text{numberOfRecords}(X_{ij}) > 0 \) then
3. \( C_{ij}.\text{add}(X_{ij}) \);
4. end
5. end

**Step 4: Cluster Merging**

1. Let \( S_i \in S \) be the number of records in \( D \) with the class that has the \( i^{th} \) index;
2. \( S_i \leftarrow \text{CountClassRecords}(D) \);
3. Let \( S_i^{\text{min}} \) be the minimum number of records allowed for a class-specific cluster with class index \( i \);
4. foreach \( S_i \in S \) do
5. \( S_i^{\text{min}} \leftarrow S_i \times \theta \);
6. end
7. foreach \( C_{ij} \in C \) do
8. if \( \text{numberOfRecords}(C_{ij}) < S_i^{\text{min}} \) then
9. Find the nearest cluster (distance between centroids) to \( C_{ij} \). Let it be \( C_{ij}^{\text{neighbor}} \);
10. Combine \( C_{ij} \) and \( C_{ij}^{\text{neighbor}} \) into one cluster;
11. Update \( C \) accordingly;
12. end
13. end
14. return \( C \);
15. end

**Algorithm 8:** RBClust
The classifier must be rule-based. This is because in Step 3 of RBClust, the last condition in the formed rules is used to create clusters.

### 5.5 Experiments

#### 5.5.1 Setup

This chapter empirically compares the proposed method RBClust against a basic version of k-means SimpleKMeans, and Affinity Propagation (AP). The basic k-means is chosen since it is what was used in an early work which required CSCs (Japkowicz, 2001). AP was chosen due to its popularity within traditional clustering literature. The methods are compared over 4 class-specific problems which come from 2 separate datasets. The transfusion dataset is available from the UCI Machine Learning Repository (Lichman, 2013). Since class-specific clustering is a major component in several class imbalance related methods (Japkowicz, 2001)(Nickerson et al., 2001), this chapter uses a dataset (ecoli1) from the imbalanced dataset repository in KEEL (Fernández et al., 2007; Fernández et al., 2017). The implementation of SimpleKMeans from WEKA (Hall et al., 2009) and the implementation of AP from ELKI (Schubert et al., 2015) (version 0.7) are used. For k-means, k was set to 4 to be consistent with (Japkowicz, 2001). For RBClust, $\theta$ was set to 0.02 based on this chapter’s experiments. All other parameters are set to default values. For the cluster evaluation metric, the well-known silhouette coefficient (Rousseeuw, 1987) was used.

<table>
<thead>
<tr>
<th>&lt;Dataset-ClassValue&gt;</th>
<th>#Records</th>
<th>#Attributes</th>
<th>kMeans</th>
<th>AP</th>
<th>RBClust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfusion-1</td>
<td>178</td>
<td>5</td>
<td>0.307</td>
<td>0.179</td>
<td>0.408</td>
</tr>
<tr>
<td>Transfusion-0</td>
<td>570</td>
<td>5</td>
<td>-0.028</td>
<td>0.074</td>
<td>0.227</td>
</tr>
<tr>
<td>ecoli1-P</td>
<td>77</td>
<td>8</td>
<td>0.26</td>
<td>0.194</td>
<td>0.345</td>
</tr>
<tr>
<td>ecoli1-N</td>
<td>259</td>
<td>8</td>
<td>-0.006</td>
<td>0.184</td>
<td>0.235</td>
</tr>
</tbody>
</table>

**Table 5.1:** Methods Comparison - Silhouette Coefficient

#### 5.5.2 Discussion of Results

The results for the cluster silhouette comparison are shown in Table 5.1. They are also visualised in Figure 5.3. RBClust achieves the highest silhouette coefficient
5.5. Experiments

Those who are reading this in black-and-white are referred to Table 5.1 which presents the same information but in further detail.

(shown in bold) in all four problems compared to the existing methods. kMeans can achieve poor results due to the set number of clusters \((k)\). However, this is not an issue for RBClust since the number of clusters is not pre-defined (also true for AP).

The computation time needed for finding all CSCs within each dataset was also recorded. This information is shown in Table 5.2 and visualised in Figure 5.4. The computation time was recorded using a simple machine with a single 1.6 GHz core, 1 GB of RAM, 533 MHz bus speed and a 512 KB processor cache.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>kMeans</th>
<th>AP</th>
<th>RBClust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfusion</td>
<td>116</td>
<td>5496</td>
<td>197</td>
</tr>
<tr>
<td><code>ecoli1</code></td>
<td>128</td>
<td>192</td>
<td>199</td>
</tr>
</tbody>
</table>

Table 5.2: Time to Find all CSCs (ms)

It is apparent that for the larger dataset (Transfusion), AP scales poorly in computation time. However, RBClust is able to perform the clustering process in a small fraction of the time. Even though kMeans is slightly faster in both datasets, one can see in Table 5.1 that it does not perform as well as RBClust. Based on these results, RBClust has promise for delivering high quality clusters with low computation time.
Future work relating to this chapter involves extending RBClust for higher quality without significantly increasing the computation time.

### 5.5.3 Time improvement to Standoff

To investigate the effect that using RBClust with Standoff has on execution time, 10 cross fold validation for Standoff was performed on four datasets. On each dataset, the average time to finish building a classifier from a fold was recorded. This process was repeated twice; once when using Standoff with EM clustering (as proposed in Chapter 4), and once when using Standoff with RBClust. Therefore, the comparison is between the original Standoff and the enhanced Standoff (through the use of RBClust.) This corresponds to the execution time of the processes shown in Figures 5.1 and 5.2. The results of this time comparison are summarised in Figure 5.5 and Table 5.3. The datasets used for this comparison are discussed in detail in Chapter 6.

Figure 5.6 illustrates the scalability of Standoff with EM vs Standoff with the proposed RBClust. In the figure, one can observe that Standoff vs RBClust appears to scale better than Standoff with EM. That is, for each dataset, Standoff with RBClust...
5.5. Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Standoff + RBClust Standoff + EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC2</td>
<td>MC2 0.445±0.129 2.762±0.971</td>
</tr>
<tr>
<td>KC3</td>
<td>KC3 2.813±0.663 8.903±1.747</td>
</tr>
<tr>
<td>MW1</td>
<td>MW1 9.851±2.568 19.778±3.804</td>
</tr>
<tr>
<td>CM1</td>
<td>CM1 13.012±4.216 29.075±7.405</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>6.530</strong> <strong>15.129</strong></td>
</tr>
</tbody>
</table>

Table 5.3: The average execution time (seconds) of a fold in 10 cross-fold validation. Results shown are for Standoff with EM (proposed in Chapter 4) and Standoff with RBClust (proposed in this Chapter).

Figure 5.5: The average execution time (seconds) of a fold in 10 cross-fold validation. Results shown are for Standoff with EM (proposed in Chapter 4) and Standoff with RBClust (proposed in this Chapter). visualised. Those who are reading this in black-and-white are referred to Table 5.3 which presents the same information but in further detail.

is the fastest and the gradient of Standoff with RBClust’s line is less steep than the line of Standoff with EM.

5.5.4 Discussion about setting $\theta$

Recall from Section 5.4 that the parameter $\theta$ is used to avoid having many small clusters. Therefore, for datasets that contain disparate, sparse and very small groups of records will require a higher value for $\theta$. A sensible recommendation would be to understand the data before applying RBClust such that an understanding of the sparseness is achieved. For smaller datasets such as those used in this chapter, a value of 0.02 will likely be the best performing value. However, for larger datasets
Chapter 5. RBClust: Fast Discovery of Class Specific Clusters

![Graph showing time taken to execute on each dataset. Each point along the x-axis is the number of records within that dataset.](image)

**Figure 5.6:** The time taken to execute on each dataset. Each point along the x-axis is the number of records within that dataset.

It would be worthwhile to perform a grid-search to determine which value of $\theta$ is most appropriate.

## 5.6 Conclusion

The previous chapter provided an answer to Research Question 1 through the proposed algorithm Standoff. A drawback of Standoff is its execution speed. As a result, the research of this thesis could not proceed efficiently until the execution speed of Standoff was improved. A time-consuming step of Standoff is that a clustering algorithm must be run twice, once on the positive class and once on the negative class.

This chapter defined the class-specific clustering problem. The algorithm proposed in this Chapter (RBClust) allows us to find the class-specific clusters quickly with sufficient accuracy (measured using silhouette coefficient). This allowed us to further experiment using Standoff in Chapters 6 and 7.

Both Standoff and RBClust were proposed within the context of binary classification. RBClust can easily be used with multiple class datasets since each class is clustered individually following a decision tree classification. Since decision trees can handle multiple classes, RBClust can too. To apply multiple class classification to
5.6. Conclusion

Standoff, it would require further modification. To achieve it without further modifying Standoff, a data scientist could instead split the multiple class dataset up into several datasets. For example, if one dataset had 3 class values, Standoff could be applied with three versions of the dataset. Class 1 vs non-Class1, Class 2 vs non-Class2 and Class 3 vs non-Class3.

The following chapters of this thesis use Standoff Balancing and RBClust (proposed in the previous chapter and this chapter respectively) as components in a larger framework which addresses Research Questions 2 and 3. This larger framework introduces cost-sensitivity as a major component which allows for high performance cost-sensitive classification (Chapter 6) and cost-sensitive knowledge discovery (7) from class imbalanced data.
Part III

Classification in Imbalanced Data:
Cost-Sensitive
Chapter 6

BCF: Balanced Costs Framework

Research Question 1 of this thesis was addressed by proposing Standoff and RB-Clust in Chapters 4 and 5 respectively. Using these algorithms, one can produce a cost-insensitive classifier. However, Research Questions 2 and 3 are focused on cost-sensitive classification. This chapter uses the work performed in previous chapters as components in a larger framework. This framework is able to perform cost-sensitive classification on class imbalanced datasets. Therefore, this chapter presents an answer to Research Question 2 and a tool for exploring Research Question 3.

Although Standoff uses costs to combat the class imbalance problem, it does not produce a cost-sensitive classifier. This is because the specified costs reflect the class imbalance, not the real life costs of TN, TP, FN, and FP predictions. For this reason, this chapter defines the difference between these two types of costs. This thesis calls the former balancing costs and the latter domain costs. The framework proposed in this chapter can use both balancing costs and domain costs together with Standoff and RBClust. The resulting framework BCF produces a cost-sensitive and balanced decision forest without changing the original training dataset.

6.1 Introduction

Predicting which sections of code contain bugs is a process called Software Defect Prediction (SDP) (Öztürk et al., 2015; Ricky et al., 2016; Jiang et al., 2013; Shepperd et al., 2013; Ling et al., 2006; Sheng et al., 2014; Siers and Islam, 2014; Siers and Islam, The work presented in this chapter was published under the title: “Addressing Class Imbalance and Cost Sensitivity in Software Defect Prediction by Combining Domain Costs and Balancing Costs” in the proceedings of the 2016 International Conference on Advanced Data Mining and Applications, LNCS volume 10086, Springer Cham, pp 156-171. ERA 2010 Rank B, for which it won the Spotlight Paper award.
These sections of code are referred to as modules. If a module contains at least one bug, it is considered defective. In many SDP studies, each C/Java function is considered as one module (Shepperd et al., 2013). Therefore, these studies separated a software project’s source code into each function, then predicted which function contained bugs.

SDP studies either take a non cost-sensitive approach (Öztürk et al., 2015; Ricky et al., 2016; Jiang et al., 2013) or a cost-sensitive approach (Ling et al., 2006; Sheng et al., 2014; Siers and Islam, 2014; Siers and Islam, 2015a). When performed non-cost-sensitively, the aim is to make as many correct predictions as possible. Certain types of predictions incur different real life costs. When SDP is performed cost-sensitively, the aim is to minimise costs which are incurred by the predictions. This study focuses on cost-sensitive SDP.

Recall from Chapter 3 that typically, four cost-parameters need to be defined prior to performing cost-sensitive SDP. These four parameters are the monetary costs incurred by each of the following prediction types: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). These types and their associated costs are explained in Table 6.1. This table also includes the values typically used in SDP studies (Ling et al., 2006; Sheng et al., 2014; Siers and Islam, 2014; Siers and Islam, 2015a).

<table>
<thead>
<tr>
<th>Prediction Type</th>
<th>Acronym</th>
<th>Business Scenario</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive</td>
<td>TP</td>
<td>A defective module is correctly predicted as defective. The software development business must now assign resources to fixing the defective module.</td>
<td>$1000</td>
</tr>
<tr>
<td>True Negative</td>
<td>TN</td>
<td>A defect-free module is correctly predicted as containing no defects. No action is required.</td>
<td>$0</td>
</tr>
<tr>
<td>False Positive</td>
<td>FP</td>
<td>A defect-free module is incorrectly predicted as containing defects. After assigning resources to fix the module, the software development business learns that the module is actually defect-free. The cost associated with this prediction type is the typically considered equal to the cost associated with a TP.</td>
<td>$1000</td>
</tr>
<tr>
<td>False Negative</td>
<td>FN</td>
<td>A defective module is incorrectly predicted as defect-free. Therefore, the defect remains undetected within the program. This can cause huge damage to the program and hinder the software development process at a later stage. The cost associated with this prediction type is typically 5 to 8 times more than the cost associated with a TP.</td>
<td>$5000 to $8000</td>
</tr>
</tbody>
</table>

TABLE 6.1: The different types of predictions in SDP and their associated costs.
This thesis refers to a program which performs SDP as an SDP system. Such a system consists of a classifier which is generated by a supervised algorithm such as a decision tree (Quinlan, 2014). A training dataset $D_T$ which is used for SDP has the following characteristics: each record represents one module; each attribute represents a module measure such as number of lines; the possible classes are Defective and Non-Defective.

Cost-sensitive classification algorithms are traditionally used to produce cost-sensitive classifiers. They can alternatively be used to mitigate class imbalance such as the proposed Standoff in Chapter 4. However, when used for this purpose, a non-cost-sensitive classifier is produced. This can be achieved by setting the TP, FP, TN, and FN costs to mirror the degree of imbalance. These costs are different to the costs in Table 6.1. To distinguish between these two types of costs, this thesis refers to the costs in Table 6.1 as domain costs and the costs used to negate class imbalance as balancing costs. This thesis defines these costs as follows:

**Balancing Costs:** This thesis denotes the balancing costs by $C_{TP}^\beta$, $C_{TN}^\beta$, $C_{FP}^\beta$, and $C_{FN}^\beta$. Where $C_{TP}^\beta$ denotes the balancing cost of a TP prediction. They are used as input to a cost sensitive classification algorithm to negate class imbalance and produce a non-cost-sensitive classifier. To achieve this, $C_{TP}^\beta$ and $C_{TN}^\beta$ are set to 0 since there should not be a penalty for producing correct predictions. $C_{FN}^\beta$ is set to $\frac{|M|}{|N|} \times C_{FP}^\beta$, where $|M|$ and $|N|$ are the number of majority and minority class records respectively. By doing so, the ratio between the majority and minority class is mirrored by the ratio between $C_{FN}^\beta$ and $C_{FP}^\beta$. For simplicity, $C_{FP}^\beta$ is typically set to 1.

**Domain Costs:** Similar to this thesis’s notation for balancing costs, domain costs are denoted by $C_{TP}^\delta$, $C_{TN}^\delta$, $C_{FP}^\delta$, and $C_{FN}^\delta$. They describe the real life cost incurred by making the corresponding prediction. For example, $C_{FN}^\delta$ is the real life cost that is incurred when a defective module is assumed to be defect free. In this case, the cost is high since an undetected defect exists within the code base. Once $C_{TP}^\delta$, $C_{TN}^\delta$, $C_{FP}^\delta$, $C_{FN}^\delta$ have been defined, they can be used as input for a cost-sensitive classification algorithm to produce a cost-sensitive classifier.
6.2 BCF: A cost-sensitive classification technique

Balanced Cost Framework (BCF), is a cost-sensitive classification framework for class imbalanced datasets. It consists of 4 steps. An overview of these steps is illustrated in Figure 6.1. In this figure, blue rectangles represent processes, and orange ellipses represent inputs or outputs. This chapter fully provides an explanation of BCF. Its details are presented in three forms. BCF is represented through pseudocode in Algorithm 9 which is further explained in the following paragraphs. An overview of BCF is also represented visually in Figure 6.1.

BCF - Step 1: Discovery of Class-Specific Clusters: The proposed framework uses RBClust (proposed in Chapter 5) to discover the class-specific clusters in the training dataset. The same settings are used as was used in Chapter 5. These clusters are required as input to Step 2. RBClust does not need to separate the data into the majority and minority subsets before clustering. By using RBClust in this framework, the need to run a clustering algorithm twice is eliminated.

BCF - Step 2: Calculation of Cluster Specific Balancing Costs: The class specific clusters found in Step 1 are used as input to the Standoff algorithm (proposed in Chapter 4). Standoff calculates the balancing costs for each cluster. The output of Standoff is a set balancing cost-matrices, one for each record.

BCF - Step 3: Combination of Domain Costs and Balancing Costs

Different methods for combining balancing cost matrices with a domain cost matrix are compared here. Section 6.3 uses the most reliable combination method in the proposed framework.

Two Simple Combination Methods:
Input: A training dataset $D_T$.
A domain cost-matrix $C^\delta = \{C^\delta_{TP}, C^\delta_{TN}, C^\delta_{FP}, C^\delta_{FN}\}$

Output: A balanced cost-sensitive classifier.

1 **Step 1: Discovery of Class-Specific Clusters**
   
   /* $G$ holds all the class-specific clusters in $D_T$ such that $G_{ij}$ is the $j^{th}$ cluster for the $i^{th}$ class. */
   
   $G \leftarrow \text{RBClust}(D_T)$; // See Chapter 5 for details on RBClust

2 **Step 2: Calculation of Cluster Specific Balancing Costs**
   
   /* The following function StandoffV2 is a modified version of Standoff Balancing proposed in Chapter 4. StandoffV2 accepts the class-specific clusters as input, thereby bypassing lines 1 to 10 inclusive of the original algorithm (see Chapter 4). The second modification is that the return value is instead the calculated balancing cost-matrices $C^\beta$ for each record. */

   $C^\beta \leftarrow \text{StandoffV2}(D_T, G)$

3 **Step 3: Combination of Domain Costs and Balancing Costs**
   
   /* $D_{Tx}$ denotes the record in $D_T$ with index $x$ */

   foreach $D_{Tx} \in D_T$ do
      
      /* $C^\gamma$ is the set of combined cost-matrices such that $C^\gamma_{TPx}$ is the combined true positive cost for the $x^{th}$ record in $D_T$ */

      $C^\gamma_{TPx} \leftarrow C^\delta_{TPx}$
      $C^\gamma_{FPx} \leftarrow C^\delta_{FPx}$
      $C^\gamma_{TNx} \leftarrow C^\delta_{TNx}$
      $C^\gamma_{FNx} \leftarrow \frac{(C^\delta_{FNx} + C^\delta_{FPx})}{2}$

   end

4 **Step 4: Cost-Sensitive Classification using Modified CSForest**
   
   /* For details on the modified version of CSForest called below, see the accompanying text for Step 4 in Section 6.2. */

   return $\text{ModifiedCSForest}(D_T, C^\gamma)$

end

Algorithm 9: BCF - Balanced Costs Framework
These techniques involve either adding or multiplying the domain costs and balancing costs. The addition technique computes the combined costs as the addition of the corresponding domain and balancing costs. For example $C_{TP} = \delta_{TP} + \beta_{TP}$. The difference between the addition technique and the multiplication technique is that the corresponding costs are multiplied instead of added.

These two simple techniques are compared against a CSForest classifier. Since CSForest is a recent cost-sensitive method proposed for SDP, it acts as a reasonable baseline for performance. The data set used for evaluation is MC2', a publicly available software defect prediction data set (Shepperd et al., 2013). We measure the effectiveness of the resulting classifier in total cost. The equation for total cost is given in Equation 6.1. Note that total cost is calculated using the domain costs only. 10-fold stratified cross validation is used in order to reduce variance in this analysis.

$$TotalCost = (N_{TP} \times C_{TP}^d) + (N_{FP} \times C_{FP}^d) + (N_{TN} \times C_{TN}^d) + (N_{FN} \times C_{FN}^d)$$ (6.1)

One can observe from Table 6.2 that addition and multiplication methods are not lowering total cost from the CSForest classifier. Thus, the results for addition and multiplication are not promising.

<table>
<thead>
<tr>
<th></th>
<th>CSForest</th>
<th>Addition</th>
<th>Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>135</td>
<td>136</td>
<td>134</td>
</tr>
</tbody>
</table>

Using Balancing Costs in the Prediction Phase:

Typically, cost-sensitive methods use domain-costs in both the classifier training process and the classifier prediction process. The proposed framework uses CSForest which first uses the costs in training the classifier. The costs are then used in CSVoting when classifying a new record.

<table>
<thead>
<tr>
<th></th>
<th>CSForest</th>
<th>Addition</th>
<th>Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>135</td>
<td>124</td>
<td>132</td>
</tr>
</tbody>
</table>
In Table 6.2, the combined cost-matrices were used in both CSForest and CSVoting. However, it is possible that a better result could be achieved by only using the combined matrices once. Thus, an analysis is now performed for the performance of using the combined cost-matrices when building the CSForest classifier, but only the domain costs in CSVoting. This way, the classifications are optimized for total cost in the CSVoting step. This process is illustrated in Figure 6.2.

The above approach is now compared against using the combined cost-matrices for both CSForest and CSVoting. Note that in Table 6.2, the latter approach is used, but in Table 6.3 the former is used. Thus, a comparison can be made between the two tables. The total cost of a CSForest classifier is also included for comparison.

In comparison to Table 6.2, the multiplication technique performs slightly better, and the addition technique is performing much better. Therefore, only domain costs are chosen to be used in CSVoting. Before deciding whether to use the addition or multiplication technique, a comparison is first made against a slightly more intelligent approach.

**Only Changing the FP and FN Costs:** We now analyse an alternative to the two simple techniques of addition and multiplication. When generating balancing costs, it is the ratio between the FP and FN costs that is important (Siers and Islam, 2015b). Therefore, only the values of FP and FN in the domain costs are changed to generate the combined cost-matrices. This allows the domain costs of TP and TN classifications to be maintained.

To represent both the balancing cost-matrices and the domain cost-matrix for the FP and FN costs, two weights $a$ and $b$ are introduced for the balancing and domain cost matrix respectively. Therefore, the combined cost matrix can be written as shown in Table 6.4. When setting balancing cost-matrices $C_{FP}^\beta$ can be set arbitrarily since $C_{FN}^\beta$ is calculated as a factor of $C_{FP}^\beta$. This experiment sets $C_{FP}^\beta = 1$. When using cost-sensitive classification to deal with class imbalance, $C_{FP}^\beta$ is typically set
Table 6.4: Combining Only FP and FN Costs

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>$C_{TP}^\alpha$</td>
<td>$C_{FP}^\alpha$</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>$a \times (C_{FN}^\beta) + b \times (C_{FN}^\delta)$</td>
<td>$C_{TN}^\gamma$</td>
<td></td>
</tr>
</tbody>
</table>

...to 1 (Siers and Islam, 2015b). Therefore, in order to simplify our calculations, $C_{FP}^\gamma$ is set to $C_{FP}^\delta$, that is, both of these values are one.

Further experiments are performed with applying different sets of values for weights $a$ and $b$ where $a + b = 1.0$. They range from 0.0 to 1.0 and with pitch width 0.1. For example, when $a = 0.5$ and $b = 0.5$, this is the same as taking the average between $C_{FN}^\beta$ and $C_{FN}^\delta$ for computing $C_{FP}^\gamma$. Figure 6.3 shows a clear trend for the best combination to be at $a = 0.5$ and $b = 0.5$.

When using this approach for combining the cost-matrices, the experiments illustrate that it can further reduce the total cost in the MC2’ data set to 122. Therefore, the combined cost-matrix in Table 6.4 is chosen for $a = 0.5$ and $b = 0.5$ and only the domain costs in CSVoting as shown in Figure 6.2. This figure also shows a second degree trend curve. This curve helps to further illustrate the trend of performance for the different weight settings.

**Figure 6.3: Total Cost Comparison for Different Weight Combinations**

BCF - Step 4: Cost-Sensitive Classification using Modified CSForest: CSForest (Siers and Islam, 2015a) requires a single cost-matrix as input and thus does not handle multiple cost-matrices. The proposed approach modifies CSForest to use record
specific cost-matrices rather than a single cost matrix. This is done so that the combined cost-matrices produced in Step 3 can be used. Expected Misclassification Cost (EMC) is calculated using $C_P$ and $C_N$. By replacing these equations with Equations 6.2 and 6.3, CSForest is modified to use record specific cost-matrices. In the proposed modified equations, $C_{TP_i}^{\beta}$ denotes $C_{TP}^{\beta}$ from the $i^{th}$ record’s balancing cost-matrix.

\[
C_P = \sum_{i=1}^{\lvert R \rvert} \begin{cases} 
C_{TP_i}^{\beta} & \text{if } R_i \in P \\
C_{FP_i}^{\beta} & \text{if } R_i \in N
\end{cases} \quad \text{(6.2)} 
\]

\[
C_N = \sum_{i=1}^{\lvert R \rvert} \begin{cases} 
C_{PN_i}^{\beta} & \text{if } R_i \in P \\
C_{TN_i}^{\beta} & \text{if } R_i \in N
\end{cases} \quad \text{(6.3)} 
\]

6.3 Experiments

This section compares the effectiveness of the proposed framework BCF against existing methods. These experiments are described in Section 6.3.1. Their results are discussed in Section 6.3.3.

6.3.1 Experimental Setup

The experiments are conducted using a 10-fold cross validation approach. When using this approach to evaluate total cost there is a small difference compared to how cross fold validation is typically used. This detail can be seen in Figure 6.4 which describes the 10-fold cross validation approach for evaluating total cost on a given dataset. The small difference is that for total cost, the total cost of each fold is summed to produce the final total cost. This differs to other measures which calculate the average of each fold. The reason for this difference is that total cost is not a fraction like most other evaluation metrics such as accuracy, AUC, and precision. Note that although a data miner could instead find the average total cost by dividing the summed total cost by 10, there is no need to do so. Some implementations of cross validation include a component which finds optimal parameter settings. The implementation used in this study does not include a parameter optimisation component.

The domain costs for cost sensitive methods are set to: $C_{TP}^{\delta} = 1$, $C_{TN}^{\delta} = 0$, $C_{FP}^{\delta} = 1$ and $C_{FN}^{\delta} = 5$. This is consistent with existing SDP studies (Siers and Islam,
To evaluate the proposed classification framework, a comparison was performed against eight existing classification methods. In the upper portion of Table 6.6, the compared methods and their properties are presented. The 2 properties which a method can have are cost-sensitive and treats class imbalance.

This experiment measured the total cost achieved by each method on nine publicly available SDP datasets. In the table, each cell shows the result of adding all total costs of each cross-fold. Four of these datasets were collected from software projects undertaken by NASA. This experiment uses the cleaned versions of the datasets (Shepperd et al., 2013) (KC3’, MC2’, MW1’, CM1’). The remaining 5 SDP datasets were collected from software projects undertaken by a white-goods manufacturer (Softlab, 2009). Further information on these datasets are given in Table 6.5.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Lines of Code</th>
<th>Language</th>
<th>Number of Defective Modules</th>
<th>Number of Defect-Free Modules</th>
<th>Percentage of Modules Defective</th>
<th>Total Number of Modules</th>
<th>Number of Software Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR6</td>
<td>2078</td>
<td>C</td>
<td>15</td>
<td>86</td>
<td>14.85%</td>
<td>101</td>
<td>29</td>
</tr>
<tr>
<td>AR1</td>
<td>2467</td>
<td>C</td>
<td>9</td>
<td>112</td>
<td>7.44%</td>
<td>121</td>
<td>29</td>
</tr>
<tr>
<td>AR5</td>
<td>2732</td>
<td>C</td>
<td>8</td>
<td>28</td>
<td>22.22%</td>
<td>36</td>
<td>29</td>
</tr>
<tr>
<td>MC2’</td>
<td>5503</td>
<td>C</td>
<td>44</td>
<td>83</td>
<td>34.65%</td>
<td>127</td>
<td>39</td>
</tr>
<tr>
<td>AR3</td>
<td>5624</td>
<td>C</td>
<td>8</td>
<td>55</td>
<td>12.70%</td>
<td>63</td>
<td>29</td>
</tr>
<tr>
<td>KC3’</td>
<td>6399</td>
<td>Java</td>
<td>36</td>
<td>164</td>
<td>18%</td>
<td>200</td>
<td>39</td>
</tr>
<tr>
<td>MW1’</td>
<td>6905</td>
<td>C</td>
<td>27</td>
<td>237</td>
<td>10.23%</td>
<td>264</td>
<td>37</td>
</tr>
<tr>
<td>AR4</td>
<td>9196</td>
<td>C</td>
<td>20</td>
<td>87</td>
<td>18.69%</td>
<td>107</td>
<td>29</td>
</tr>
<tr>
<td>CM1’</td>
<td>15486</td>
<td>C</td>
<td>42</td>
<td>302</td>
<td>12.21%</td>
<td>344</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 6.5: Metadata on the SDP datasets used in this study. Sorted by the number of lines of code used in the project’s source code.
6.3. Experiments

<table>
<thead>
<tr>
<th>Treats Class Imbalance</th>
<th>✓</th>
<th>X</th>
<th>X</th>
<th>✓</th>
<th>✓</th>
<th>X</th>
<th>X</th>
<th>✓</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost Sensitive</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BCSForest</th>
<th>CSForest</th>
<th>CSTree</th>
<th>IRUS</th>
<th>Adasyn + C4.5</th>
<th>SysFor + Voting 1</th>
<th>SysFor + Voting 2</th>
<th>ForestPA</th>
<th>BCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>KC3'</td>
<td>166</td>
<td>174</td>
<td>170</td>
<td>157</td>
<td>155</td>
<td>156</td>
<td>160</td>
<td>178</td>
<td>153</td>
</tr>
<tr>
<td>MC2'</td>
<td>128</td>
<td>135</td>
<td>128</td>
<td>125</td>
<td>132</td>
<td>127</td>
<td>167</td>
<td>160</td>
<td>175</td>
</tr>
<tr>
<td>MW1'</td>
<td>128</td>
<td>136</td>
<td>173</td>
<td>179</td>
<td>147</td>
<td>122</td>
<td>130</td>
<td>136</td>
<td>118</td>
</tr>
<tr>
<td>CM1'</td>
<td>215</td>
<td>206</td>
<td>218</td>
<td>228</td>
<td>208</td>
<td>212</td>
<td>214</td>
<td>210</td>
<td>203</td>
</tr>
<tr>
<td>AR1</td>
<td>193</td>
<td>125</td>
<td>188</td>
<td>185</td>
<td>154</td>
<td>133</td>
<td>133</td>
<td>121</td>
<td>129</td>
</tr>
<tr>
<td>AR3</td>
<td>81</td>
<td>72</td>
<td>135</td>
<td>78</td>
<td>101</td>
<td>73</td>
<td>69</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>AR4</td>
<td>188</td>
<td>113</td>
<td>220</td>
<td>215</td>
<td>201</td>
<td>126</td>
<td>131</td>
<td>115</td>
<td>107</td>
</tr>
<tr>
<td>AR5</td>
<td>53</td>
<td>36</td>
<td>44</td>
<td>51</td>
<td>53</td>
<td>41</td>
<td>44</td>
<td>39</td>
<td>36</td>
</tr>
<tr>
<td>AR6</td>
<td>214</td>
<td>111</td>
<td>228</td>
<td>187</td>
<td>197</td>
<td>215</td>
<td>218</td>
<td>105</td>
<td>99</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>151.78</td>
<td>123.11</td>
<td>167.11</td>
<td>156.11</td>
<td>149.78</td>
<td>139.11</td>
<td>139.89</td>
<td>127.00</td>
<td>114.67</td>
</tr>
</tbody>
</table>

**Table 6.6**: Total cost comparison (lower is better). Highlighted cells indicate the best performing methods for the corresponding dataset. The upper portion of this table indicates the properties of the corresponding method. ✓ indicates the presence of a property and X denotes the opposite.

6.3.2 Note About Previously Used Datasets

A requirement for selecting datasets in this chapter’s experimentation is that they are cost-sensitive. This means that the datasets used in previous chapters’ experiments are not suitable. More specifically, there are no domain cost-matrices available for those datasets. For this reason, software defect prediction datasets are a good fit for evaluating BCF.

6.3.3 Results and Discussion

The results presented in Table 6.6 illustrate that in $\frac{2}{3}$ datasets, BCF achieves the lowest total cost. These results are also visualised in 6.5. On average, BCF achieves 6.86\% lower total cost than the next best method CSForest (Siers and Islam, 2015a). This result suggests that BCF is a useful technique for cost-sensitive classification in SDP.

Methods which only treat class imbalance achieved a total cost of $152.95 on average. Similarly, methods which are cost-sensitive but do not treat class imbalance achieved a total cost of $145.11 on average. Methods which are cost-sensitive and treat class imbalance achieved $133.23 total cost on average. This suggests that
having a strategy to address class imbalance can increase the performance of cost-sensitive classification methods.

6.3.4 Cost Spread Analysis

The previous subsection presented the application of the 10 cross-fold validation technique described by Figure 6.4 and its results in Table 6.6. This subsection provides a deeper analysis of the studied algorithms’ performance. Specifically, the aim of this subsection is to investigate the variance in performance of each technique. To reach this end, this analysis extends the number of experiments of each algorithm
6.3. Experiments

Figure 6.6: The minimum, average, and maximum results of the cost spread analysis for KC3. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.

on each dataset, then records the minimum, maximum, and average performances.

The procedure for this experiment is explained as follows. The approach described in Figure 6.4 is carried out 10 times for each dataset. Since the approach in Figure 6.4 is non-determinate, 10 different performance scores are produced. Using these scores, the average, minimum, and maximum performances are calculated for each algorithm on each dataset. This experiment uses the NASA MDP datasets (Shepperd et al., 2013) which are described in Table 6.5. Additionally, cost sensitive knowledge discovery is performed on these datasets in Section 7.2.

These experiments are designed to answer the following questions:

- Which algorithms have the least/most consistent performances on average?

- Which algorithm most commonly achieves the lowest mean total cost on average?

These questions are investigated in this section. Before these investigations, the results of the experiments are presented in Table 6.7. Since these results can be quickly understood when visualised, they are also illustrated in Figures 6.6, 6.7, 6.8, 6.9, and the cost spread is visualised in 6.10.
Chapter 6. BCF: Balanced Costs Framework

**Figure 6.7:** The minimum, average, and maximum results of the cost spread analysis for MC2. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.

**Figure 6.8:** The minimum, average, and maximum results of the cost spread analysis for MW1. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.
6.3. Experiments

Figure 6.9: The minimum, average, and maximum results of the cost spread analysis for CM1. The lower the mean, the better - visualised. Those who are reading this in black-and-white are referred to Table 6.7 which presents the same information but in further detail.

Figure 6.10: The cost spread results visualised. The technique with the lowest mean is written in bold and illustrated in blue.
Which algorithm most commonly achieves the lowest total cost on average?

The first question that was investigated is a simple one. The results found reinforce the conclusions gathered from Table 6.6. BCF achieves the lowest mean cost in all of the NASA MDP datasets. This is quickly observable from Figure 6.10. On all datasets, some algorithms achieve a lower minimum and maximum than BCF. The main purpose of the utilised 10x10 cross validation approach is to reduce variance in results by finding the mean result over repeated experiments. Therefore, the mean performance is the most indicative of each algorithm’s general performance.

Which algorithms have the least/most consistent performances on average?

This question allows us to assess the variance in performance for each algorithm. To explore it, the minimum is subtracted from the maximum for each algorithm and dataset. Then the average of these differences is calculated. The higher the average,
6.3. Experiments

Figure 6.11: The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised.

the greater the spread for that algorithm. Ideally, an algorithm has a low total cost on average, and a small spread. This would result in a data miner having high confidence that the algorithm would achieve strong performance. The results for this experiment are presented in Table 6.8. They are also visualised in Figures 6.11, 6.12, 6.13 and 6.14.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Dataset</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KC3</td>
<td>MC2</td>
</tr>
<tr>
<td>BCSForest</td>
<td>39</td>
<td>22</td>
</tr>
<tr>
<td>CSForest</td>
<td>29</td>
<td>37</td>
</tr>
<tr>
<td>CSTree</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>IRUS</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Adasyn + C4.5</td>
<td>37</td>
<td>30</td>
</tr>
<tr>
<td>SysFor + Voting 1</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>SysFor + Voting 2</td>
<td>24</td>
<td>39</td>
</tr>
<tr>
<td>Forest PA</td>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>BCF</td>
<td>43</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>max - min</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BCSForest</td>
<td>39</td>
<td>22</td>
<td>28</td>
<td>33</td>
<td>30.5</td>
</tr>
<tr>
<td>CSForest</td>
<td>29</td>
<td>37</td>
<td>22</td>
<td>25</td>
<td>28.25</td>
</tr>
<tr>
<td>CSTree</td>
<td>14</td>
<td>18</td>
<td>31</td>
<td>40</td>
<td>25.75</td>
</tr>
<tr>
<td>IRUS</td>
<td>17</td>
<td>17</td>
<td>26</td>
<td>17</td>
<td>19.25</td>
</tr>
<tr>
<td>Adasyn + C4.5</td>
<td>37</td>
<td>30</td>
<td>38</td>
<td>34</td>
<td>34.75</td>
</tr>
<tr>
<td>SysFor + Voting 1</td>
<td>31</td>
<td>16</td>
<td>12</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>SysFor + Voting 2</td>
<td>24</td>
<td>39</td>
<td>14</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>Forest PA</td>
<td>11</td>
<td>28</td>
<td>8</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>BCF</td>
<td>43</td>
<td>20</td>
<td>25</td>
<td>24</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 6.8: The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance.

From Table 6.8 one can observe that the algorithm which has the largest spread is Adasyn (He et al., 2008) with C4.5 (Quinlan, 2014). A likely explanation for this is it is because Adasyn is a non-determinate sampling algorithm; this adds an element of
Figure 6.12: The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised.

Figure 6.13: The maximum cost minus the minimum cost achieved in the cost spread analysis. A lower value indicates that the algorithm achieves more consistent performance - visualised.
randomness. The algorithm with the smallest spread is Forest PA (Adnan and Islam, 2017). As can be seen in Figure 6.10, the cost spread for Forest PA on dataset CM1’ was tiny (1). Recall from Table 6.6 that Forest PA achieved the third best performance on average. If variance was important to consider for a particular problem, Forest PA would be a strong choice. The proposed technique BCF, had the 6th smallest spread on average out of the 9 compared methods.

This variance poses a weakness of BCF compared to the other techniques. Such a problem may be addressed in future work. However, for the work in this thesis, the performance of BCF meets the aim of producing lower total costs on average. In the context of lowest total cost on average, BCF is the clear winner.

Also calculated was the correlation between the cost per record and the level of class imbalance. More specifically, the total cost that BCF achieved for each dataset was divided by the number of records in that dataset. Then the pearson correlation between these values and the percent of defective modules was calculated. The result is 0.335. This indicates that BCF is more effective the more imbalanced the dataset is.
6.3.5 Variance Analysis

An experiment was also conducted to analyse the variance in total cost of each method. The settings used are identical to those used in the preceding experiment sections. The results of this experiment are shown in Figures 6.15, 6.16, 6.17 and 6.18. These results provide further evidence that BCF is not the best with variance. However, it is important to remember that the mean result is the most indicative of performance.

Interestingly, ForestPA consistently achieved very low variance in total cost. This is in agreement with the conclusion of the previous subsection’s experiment’s conclusion.

![Figure 6.15: Variance of total cost over 10x10 crossfold validation for each method on dataset KC3.](image-url)

6.4 Conclusions

This chapter proposed a framework for building cost-sensitive decision forests from class imbalanced data. This is achieved using the proposed framework BCF (see Section 6.2). Existing class imbalance techniques such as resampling mitigate prediction bias by changing the original training data (Tahir et al., 2012; Chawla et al., 2002; He et al., 2008). This negatively affects the knowledge discovery process since
6.4. Conclusions

**Figure 6.16:** Variance of total cost over 10x10 crossfold validation for each method on dataset MC2.'

**Figure 6.17:** Variance of total cost over 10x10 crossfold validation for each method on dataset MW1.'
Chapter 6. BCF: Balanced Costs Framework

Figure 6.18: Variance of total cost over 10x10 crossfold validation for each method on dataset CM1.

the data it is extracted from has been perturbed. To get the full statistics on a pattern, it must be checked against the original unperturbed dataset. Since the proposed framework does not perturb the data, the patterns it finds accurately reflect the original data. Thus, the proposed framework does not require post-checking of a pattern to find the full statistics. This chapter’s experimental evaluation over 9 software projects demonstrated that BCF outperforms the existing methods in this study. BCF achieved a lower average total cost than the next best performing method CSForest by 6.86%. This chapter therefore presents an answer to Research Question 2.

Now that Research Question 2 has been answered, this thesis moves on to Research Question 3. Recall that Research Question 3 is focused on how cost-sensitive knowledge discovery can be performed on class imbalanced datasets. BCF is designed such that the patterns it uses for classification directly describe the entire training dataset (as opposed to a sampling strategy (Drummond and Holte, 2003; Chawla et al., 2002) or popular ensembling strategies (Breiman, 1996; Freund and Schapire, 1995; Breiman, 2001)). BCF is also designed to produce a decision forest classifier which can be manually inspected to discover knowledge (Siers and Islam, 2015a). Therefore, through BCF strides have already been made towards answering
Research Question 3. A challenge in fully answering Research Question 3 is determining exactly how BCF can most effectively be used for cost-sensitive knowledge discovery. The following chapter addresses this challenge through four contributions: The first: Defining how to measure the cost-sensitive interestingness of a pattern. The second: Designing an algorithm to automatically extract and sort patterns by their cost-sensitive interestingness. The third: A process for a data scientist to follow to utilise the proposed techniques to perform cost-sensitive knowledge discovery. The fourth: A demonstration of the proposed methods thus far to perform cost-sensitive knowledge discovery on class imbalanced data.
Chapter 7

Cost-Sensitive Knowledge Discovery Using The Proposed Methods

The previous chapter presented an answer to Research Question 2, allowing us to build a cost-sensitive decision forest whilst avoiding the negative effects of class imbalance. This chapter develops several strategies which are used in conjunction with BCF to extract cost-sensitive knowledge from data. The work presented in this chapter provides an answer to Research Question 3.

In Chapter 1, it was mentioned that a data scientist could inspect a decision forest to find rules from a cost-sensitive decision forest such as one generated by BCF. However, this process is manual and unguided. It was described how a strategy with three characteristics could provide a guided process to performing cost-sensitive knowledge discovery from a cost-sensitive decision forest. The first is an equation for quantifying how useful a decision forest’s pattern is. Second, a method for extraction and sorting of a decision forest’s patterns based on their usefulness. Thirdly, a process for taking each pattern, enhancing it, then reporting on it. This Chapter proposes methods for achieving these three characteristics in conjunction with BCF. This allows us to perform cost-sensitive knowledge discovery on class imbalanced datasets through a guided process. Thus, an answer to Research Question 3 is provided in this chapter. Furthermore, this chapter demonstrates the proposed

The work presented in this chapter has been published under the title: "Novel Algorithms for Cost-Sensitive Classification and Knowledge Discovery in Class Imbalanced Datasets with an Application to NASA Software Defects". In: Information Sciences, Vol. 459. May 2018, pp. 53-70
URL: https://doi.org/10.1016/j.ins.2018.05.035.
approach by applying it to software defect datasets collected from several projects undertaken by the National Aeronautics and Space Administration (NASA).

### 7.1 Introduction

This chapter has four major contributions. Table 7.1 summarises each contribution and provides its corresponding section number.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Contribution Number</th>
<th>Description</th>
<th>Presented in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge Discovery</td>
<td>1</td>
<td>SibCost: A cost-sensitive rule set interestingness measure. Used to cost-sensitively quantify the interestingness of a rule set.</td>
<td>7.1.1</td>
</tr>
<tr>
<td>Knowledge Discovery</td>
<td>2</td>
<td>SCORE: An algorithm for automatic extraction of the most interesting rule sets.</td>
<td>7.1.2</td>
</tr>
<tr>
<td>Knowledge Discovery</td>
<td>3</td>
<td>Breadth-First-Mining: A process for cost-sensitive knowledge discovery in decision tree/forests.</td>
<td>7.1.3</td>
</tr>
<tr>
<td>Demonstration of Knowledge Discovery</td>
<td>4</td>
<td>Cost-Sensitive Knowledge: Insights found using the proposed methods on NASA’s software projects.</td>
<td>7.2</td>
</tr>
</tbody>
</table>

Table 7.1: The major contributions of this paper.

Figure 7.1 is an overview of how Contributions 1, 2, 3, and BCF (proposed in Chapter 6) can be used together to discover cost-sensitive knowledge from a dataset. In this figure, SibCost is drawn inside SCORE because SCORE uses SibCost. This is similarly true for SCORE and Breadth-First Mining.

![Figure 7.1: How Contributions 1, 2, 3, and BCF can be used together to discover cost-sensitive knowledge from a dataset. Inputs/Outputs are represented as orange parallelograms. Each blue rectangle represents one of the proposed contributions.](image)

#### 7.1.1 SibCost: A Cost-Sensitive Rule Set Interestingness Measure

This section proposes a cost-sensitive measure for quantifying the interestingness of a pattern. The proposed approach is based on domain costs (see Table 6.1). The measure represents the potential monetary gain that can be achieved by being aware
of the pattern. The proposed measure is called SibCost which is short for sibling rule set cost.

Typically, interestingness is measured for a single logic rule. However, SibCost measures the interestingness of a sibling logic rule set. This thesis defines a sibling logic rule set as a set of logic rules which have the same final condition. This thesis hereafter refers to this concept as a rule set. In Figure 7.2, there are 3 leaf nodes. The 2 logic rules corresponding to Leaf Node 2 and Leaf Node 3 form a sibling logic rule set. Each rule generated by the proposed classification framework has an associated expected cost per record. The two logic rules within a sibling logic rule set which have the highest difference in expected cost per record is found. For example, the SibCost of this rule set is $5.13.

The reasoning for considering the rule sets with the highest sibcost to be most interesting is as follows. The rule set consists of a set of conditions $K$, and a prediction. The final condition $K_{|K|}$ is the condition which differentiates one logic rule from its sibling logic rule. All other conditions in $K$ are shared by each logic rule in the rule set. If the difference in expected cost between these two logic rules is high, then that means that $K_{|K|}$ heavily influences the expected cost of the records which satisfy all other $K \in K$.

It is potentially a challenging task to accurately determine suitable costs to set in the cost-matrix. The authors of CSTree (Sheng et al., 2014) used a general set of costs which were used in conjunction with an industry partner.

However in techniques such as ExampleCSDT (Bahnsen et al., 2015), a more pragmatic approach is used to set appropriate costs. When setting record specific
costs such as in ExampleCSDT, it is often calculated using the characteristics of the dataset. For example, in a credit card fraud detection scenario, if a feature of the training dataset is the amount of the transaction, this amount can be used to set appropriate costs within the cost matrix for each record.

The term potential is used since there is only one feature of the two sibling leaves which differentiates the two siblings. If the expected cost difference between is $400 then one could reasonably deduce that based on historical data, if one could control the differentiating factor, then the expected cost could potentially be lowered.

7.1.2 SCORE: An Algorithm for Automatic Extraction of the Most Interesting Rule Sets from Decision Trees and Forests

This subsection presents the proposed algorithm which automatically finds all possible rule sets within a decision tree, measures their SibCost and outputs an ordered stack of rule sets such that the most interesting rule sets are at the top. The proposed algorithm is named SCORE, which stands for Stacked, Cost Ordered Rule Extractions. Algorithm 10 shows the pseudocode for the SCORE algorithm which is broken into three steps. These steps are explained below.

**SCORE - Step 1: Find All Sibling Groups:** In this step, all possible groups of sibling nodes are recursively found within the decision tree. Note that for trees which only contain numeric splits, all groups will contain just two sibling nodes. However, for categorical splits, a group may contain many sibling nodes.

**SCORE - Step 2: Calculate Maximum Differences:** For each group of sibling nodes, the two nodes which have the greatest difference of expected cost per record are found. This difference is the SibCost of the rule set (we propose SibCost in Section 7.1.1).

**SCORE - Step 3: Sort and Return Stack:** The sibling groups are sorted by their corresponding maximum differences. The algorithm then returns an ordered stack of rule sets such that the most interesting rule set is at the top of the stack.

The rule sets are ordered by SibCost, with greatest SibCost being at the top and lowest at the bottom. The justification for doing this is that the rule set with the great potential monetary gain is at the first accessible location with lesser potential monetary gains further down the stack. A stack produced by SCORE has an advantage
**Input:** A decision forest $F$, a cost matrix $M$

**Output:** A stack of logic rules ordered by the maximum difference between EMCs of the corresponding leaf nodes.

1. **Step 1: Find All Sibling Groups**
   - `siblings ← [[]];`  // siblings is a list of lists.
   - `numParents ← 0;`
   - **procedure siblingSearch(node)**
     - **if** node.hasChildren **then**
       - `siblings[numParents] = node.children;`
       - `numParents ← numParents + 1;`
       - **foreach** child ∈ node.children **do**
         - `siblingsSearch(child);`
     - **end**
   - **end**
   - **for** tree ∈ $F$ **do**
     - `siblingsSearch(tree.rootNode);`
   - **end**

2. **Step 2: Calculate Max Diffs**
   - `maxDiffs ← [];`
   - **for** $i ← 0; i < numParents; i ← i + 1** do**
     - `maxDiffs[i] ← 0;`
     - /* Find the maximum difference between sibling Es (expected classification cost) for all combinations of sibling nodes in siblings[i]. */
     - **for** $j ← 0; j < siblings[i].length; j ← j + 1** do**
       - **for** $k ← j + 1; k < siblings[i].length; k ← k + 1** do**
         - `diff = |E(siblings[i][j], M) − E(siblings[i][k], M)|;`
         - **if** diff > maxDiffs[i] **then**
           - `maxDiffs[i] ← diff;`
       - **end**
   - **end**

3. **Step 3: Sort and Return Stack**
   - `sortSiblingsByMaxDiff(siblings, maxDiff);`
   - `returnStack = new Stack();`
   - **for** $i ← siblings.length − 1; i ≥ 0; i ← i + 1** do**
     - `returnStack.push(new LogicRule(siblings[i]));`
   - **end**
   - `return returnStack;`

**Algorithm 10:** SCORE algorithm.
over rules produced using support and confidence. It is ordered by cost, thereby providing a method for discovering knowledge which is optimised for monetary gain as opposed to accurate information. This is not to say that the rule sets produced by SCORE are not accurate, they are simply optimised for a different goal.

### 7.1.3 Breadth-First-Mining: A process for cost-sensitive knowledge discovery

This section proposes a process for knowledge discovery. The usefulness of this process is demonstrated in Section 7.2. The process is represented visually in Figure 7.3. Each step is described in detail as follows.

**FIGURE 7.3: The proposed process for knowledge discovery which uses the proposed algorithms: Breadth-First-Mining.**

**Breadth-First-Mining - Step 1: Environment Setup:** To begin the process, a data miner first needs a flat file (such as csv, arff, or json) which contains a dataset with class values. The data contained in the flat file is described in a data dictionary. It allows the data miner to look up the meaning of each attribute when necessary. As part of Step 1, the data miner loads the flatfile into a database. For the implementation used in this study, a SQLite (SQLite) database was used but any database can be used. To finish setting up the knowledge discovery environment, a decision forest is trained on the flatfile. If there is more than one flatfile, a decision forest is trained on each.
Breadth-First-Mining - Step 2: Breadth First Knowledge Search: After setting up the environment in Step 1, the data miner can begin Step 2. This step is named Breadth First Knowledge Search because it is designed to discover shallow knowledge, but on a broad range of attributes. This is achieved by using the SCORE algorithm which was proposed in Section 7.1.2. Therefore, by the end of this Step, an ordered stack of rule sets is produced such that the most interesting rules are at the top.

Breadth-First-Mining - Step 3: Depth Addition: When writing a data analysis report, logic rules in their raw form can be unclear. They can also provide a very narrow view. Therefore, the database generated in Step 1 can be used to perform manual analyses. These manual analyses are done to add depth to each interesting logic rule. Once the relevant SQL queries have been performed, the knowledge can be recorded in prose (everyday language). Additionally, the knowledge can be represented visually where appropriate using sensible charts. The prose and charts come together to comprise the final report. This final report is now in a form which can be delivered to a client, or presented in a research paper (such as Section 7.2 of this study).

7.1.4 Experimental Evaluation of SCORE

A simple evaluation was also performed of the proposed SCORE method. First two forests were trained, each consisting of five trees. One forest was trained using CS-Forest and the other using the proposed BCF. Similar to the approach used in the CS-Forest publication (Siers and Islam, 2015a), the top 5 rules in the CSForest classifier were found using support and confidence (Han et al., 2011b). That is, the interestingness of each rule was measured as \(0.5 \times \text{confidence} + 0.5 \times \text{support}\). The top 5 rules from the forest trained using BCF using the proposed SCORE algorithm were also found. The sibling rule set cost of the top 5 rules found using each approach was then compared. Figure 7.4 presents the results of this simple evaluation. The proposed framework (denoted by BCF + SCORE in Figure 7.4) produces rule sets which are 50% more interesting. To further explore the usefulness of SCORE, it is applied in the following section to discover knowledge from NASA software.

The evaluation also includes CSForest with SCORE. This makes for a fair comparison of SCORE’s patterns between BCF and CSForest in terms of SibCost.
Chapter 7. Cost-Sensitive Knowledge Discovery Using The Proposed Methods

![Average SibCost for the top 5 most interesting rule sets](image)

**Figure 7.4:** Average SibCost for the top 5 most interesting rule sets (higher the better).

7.2 Knowledge Discovered by Applying the Proposed Methods to NASA Software

This section applies the proposed novel framework to discover knowledge from NASA’s software defects. A description is provided of the dataset which the proposed approach extracted knowledge from in Section 7.2.1. The extracted knowledge is presented in Section 7.2.2.

7.2.1 The Dataset: MDP′

In Section 6.3, four of the datasets were collected from software projects undertaken by NASA. These datasets are KC3′, MC2′, MW1′ and CM1′. Metadata on these datasets are presented in Table ??. To extract general insights about NASA’s software defects, all the datasets were combined into one combined dataset which this chapter refers to as MDP′. The datasets KC3′, and MC2′ contain all the software measures that MW1′ and CM1′ do. However, KC3′, and MC2′ have two extra measures which are not included in MW1′ and CM1′. To combine these four datasets, the new dataset MDP′ contains all records from these datasets, but not the extra software measures that KC3′, and MC2′ include. The metadata for MDP′ is shown in Table 7.2.

The software modules in MDP′ are described by 37 different software measures. These include line count measures, Halstead measures (Halstead, 1977), and cyclomatic complexity based measures (McCabe, 1976). Before presenting the extracted
## 7.2. Knowledge Discovered by Applying the Proposed Methods to NASA Software

<table>
<thead>
<tr>
<th>Dataset Language</th>
<th>Lines of Code</th>
<th>Number of Defective Modules</th>
<th>Number of Defect-Free Modules</th>
<th>Percentage of Modules Defective</th>
<th>Total Number of Modules</th>
<th>Number of Software Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDP’</td>
<td>Java, C</td>
<td>34293</td>
<td>149</td>
<td>786</td>
<td>15.94%</td>
<td>925</td>
</tr>
</tbody>
</table>

Table 7.2: Metadata for the combined dataset MDP’.

knowledge, a brief overview is presented of these measures as follows.

The four fundamental Halstead measures $N_2, N_1, n_2,$ and $n_1$ are used to calculate the remaining six main Halstead measures. For a given software module, $N_1$ and $N_2$ are the total number of operators and operands respectively. Similarly, $n_1$ and $n_2$ are the distinct number of operators and operands respectively. The Halstead *Length* of a module is the total number of operators *and* operands. It is denoted by $N$. Similarly, Halstead *Vocabulary* is the number of distinct operators and operands. It is denoted by $n$. The equations for Length and Vocabulary are given in Equations 7.1 and 7.2.

$$N = N_1 + N_2 \quad (7.1)$$
$$n = n_1 + n_2 \quad (7.2)$$

Using the above measures, four more Halstead measures can be calculated: *Volume*, *Difficulty*, *Effort*, and *Bugs*. *Volume* combines Length and Vocabulary. Thus, Volume is a measure of the size of the program’s source code. *Difficulty* is an estimate of the how hard it is to understand the code. The *Effort* measure describes how much effort is required to implement the program. Effort can be divided by 18 to estimate the number of seconds required for implementation. *Bugs* is an estimate of how many bugs will be in the code. Volume, Difficulty, Effort, and Bugs are denoted as $V$, $D$, $E$, and $B$ respectively. They are calculated using Equations 7.3, 7.4, 7.5 and 7.6.

$$V = N \times \log_2 n \quad (7.3)$$
$$D = \frac{n_1}{2} \times \frac{N_2}{n_2} \quad (7.4)$$
$$E = D \times V \quad (7.5)$$
$$B = \frac{E^2}{3000} \quad (7.6)$$
7.2.2 Extracted Knowledge

From the dataset which was constructed in the previous section, a decision forest of 10 trees was trained. This was done using the proposed classification framework BCF. To avoid trees which were too deep, the minimum records in each leaf was set to 150. To be consistent with the cost matrixes used in the SDP literature, the following costs were used: $C_{TP}^S = 100, C_{TN}^S = 0, C_{FP}^S = 100, C_{FN}^S = 500$. From these 10 trees, SCORE produced an ordered stack of 17 rule sets. The expected cost differences of rule sets found by SCORE ranged from $1.28$ per function to $64.00$ per function. This chapter summarises the knowledge found using the proposed methods into five insights. They are presented as follows:

**Insight 1 - Volume has the strongest relationship to expected cost per function:**

This insight was found by inspecting the rule set on the top of the stack produced by SCORE. Recall from Equation 7.3 that volume $= N \times \log_2 n$. Volume is a summary of the operands and operators used in a module. Therefore, it is representative of the module’s size. The corresponding rule set for this insight is illustrated in Figure 7.5. The relationship between Volume and expected function cost is further investigated using Figure 7.6. This histogram shows a steady increase in expected function cost as volume increases. The expected function cost is $0$ for the final two intervals since there were no defective modules in these intervals. On average, increasing a function’s volume by 2000 increases its expected cost by $14.78$.

**Insight 2 - Although effort is an extension of volume, it does not predict expected cost better than volume:**

The second-top-most rule set from the SCORE stack described Halstead effort’s relationship to expected function cost. The corresponding rule set for this insight is illustrated in Figure 7.7. Recall from Section 7.2.1, effort is an estimate of the number of seconds required to implement the function. Since it is calculated as (difficulty $\times$ volume), it can be considered an extension of volume. Similarly to Insight 1, the average increase between adjacent intervals was calculated from Figure 7.8: The function’s expected cost increases by $10.11$ for every 250000 effort.

---

For two adjacent intervals where at least one interval contains no defective modules, the difference was considered to be 0.
Insight 3 - A function’s number of executable lines can predict its expected cost. However, it is less computationally expensive to measure a function’s total line count. By doing so, there is no significant drop in predictive performance:

There are many line count related measures for functions. For example: the total number of lines; the number of executable lines; the percentage of comments; the number of blank lines. In the SCORE stack, two rule sets described the relationship between line count related measures and expected function cost. These rulesets are shown in Figures 7.9 and 7.10. A keen observer may notice that these two rulesets are very similar. Calculating the total number of lines in a function is a straightforward calculation. However, it is more computationally expensive to count only the
executable lines. Similar to the previous insights, the average increase in expected function cost between adjacent intervals was calculated in Figure 7.11. From this figure, one can observe that on average, every 50 lines of executable code increases a function’s expected cost by $20.56. However, it is important to note that in Figure 7.11 that although there is an upwards trend from left-to-right, it is not monotonically increasing. This pattern would have been stronger if there was a monotonic increase.

![Figure 7.10: Second rule set for Insight 3.](image1)

**Figure 7.11:** Executable lines vs function cost.

**Figure 7.12:** Rule set for Insight 4.

**Insight 4 -** Halstead’s estimated number of errors is a good predictor of expected function cost. However, contrary to its design, it poorly identifies defective modules:

Out of all 37 measures in the studied dataset, only 1 was directly designed to predict the presence of bugs in modules: Halstead’s estimated number of errors. In the SCORE stack, there was one rule set which described the relationship between this measure and a function’s expected cost. This rule set is shown in Figure 7.12. Interestingly, this measure was only the sixth most interesting pattern.

One might expect that the median for estimated number of bugs in defective functions would be at least 1.0, and close to 0 for defect free functions. However, as can be seen from Figure 7.13, the median number of estimated bugs for defective functions was approximately 0.4. On one hand, it is a good predictor for software cost. However, on the other hand, more than 75% of the defective functions were
estimated to have less than 1 defect. These are clearly inaccurate estimates since defective functions have at least 1 defect.

**Insight 5 - For functions of at least moderate size** (Volume $> 248.96$), **Level is the strongest indicator of the expected cost per function:**

Another rule set found by the proposed method SCORE is an extension of the rule set shown in Figure 7.5. This extended rule set is shown in Figure 7.14. It uses a measure which is explained below: Halstead Level.

Recall that Volume is a measure of a programs size based on its operators and operands. A module’s minimal *potential volume* is considered to occur when there are only two operators. Level (L) is defined as the ratio between a module’s potential volume and its actual volume (Halstead, 1977). It is calculated as $L = \frac{2n_2}{n_1N_2}$. This makes Level a measure of how concise the implementation of a module is. Conciseness is considered to be a desired trait of a module (Halstead, 1977). Therefore, in summary, Insight 5 can be re-stated as: For a function of at least moderate size, keeping its implementation concise will help minimise its expected cost.
7.3 Conclusions

This chapter presents an answer to Research Question 3. It allowed us to extract cost-sensitive knowledge from an application area that is well-known to be class imbalanced. Furthermore, this knowledge discovery was performed without changing the original dataset in any way.

The proposed measure SibCost (see Section 7.1.1) allows data miners to quantify the interestingness of sibling logic rule sets. This measure represents the potential financial savings that can be achieved by being aware of the rule set. This chapter also proposed SCORE, a method for extracting all logic rule sets ordered by SibCost. The top five rules found using SCORE with BCF were compared against an existing method for finding cost-sensitive rules. The proposed methods found rule sets with 50% higher SibCost on average.

This chapter proposed and demonstrated a process for cost-sensitive knowledge discovery: this process was named Breadth-First-Mining. This process was used with the proposed method BCF, however, it can be used in conjunction with any decision forest algorithm. The proposed process was followed to create Section 7.2. That section summarised the findings into 5 cost sensitive insights on NASA software defects. These insights provide an example of the knowledge that can be gained using Breadth-First-Mining.

At this point in the thesis, Research Questions 1, 2 and 3 have been addressed. Chapter 8 discusses a strategy for how this thesis’ research could be applied in practice.
Part IV

Denouement
Chapter 8

Discussion

This chapter discusses two subjects which have arisen from the work proposed in Chapters 3, 4, 5, 6, and 7. The subjects are as follows:

- **Integrating this thesis’ research into the software development process**: This subject is addressed in 8.1 through the presentation of an IDE integration concept (IDE stands for integrated development environment. Common examples include Eclipse (Eclipse-Foundation, 2004) and Microsoft Visual Studio (Microsoft, 1997)).

- **The Future**: This chapter also describes the potential research directions that are conducive from this thesis in Section 8.3 and also the implications that this work has on the domain’s literature as a whole.

8.1 IDE Integration Concept

Chapter 7 proposed an approach for finding patterns which most affect the expected real-life cost of a function. The SDP literature has typically focused on building high performance classifiers. However, there is a lack of research which explores how SDP can be integrated into the software development process. This subsection provides a discussion on that problem; the following questions are discussed:

1. Can SDP be performed in real-time to assist a developer as they are programming?

---

The work presented in this section is currently under review for publication in Data and Knowledge Engineering, H-Index: 77, Scimago Q2 under the title: "Integrating Cost-Sensitive Knowledge Discovery into the Software Development Process".
2. Can cost-sensitive knowledge be presented to a developer in a way which facilitates code improvement?

### 8.1.1 Back-End Architecture

The back-end architecture of the IDE integration concept is illustrated in Figure 8.1. It is split into two steps which are described in detail as follows.

**Figure 8.1: The back-end architecture of how the proposed work can be integrated into a software development environment.**

In Step 1, the process starts with the version control repository that is in use such as Git (Torvalds, 2005) or Subversion (Apache, 2004). From the version control, historic bug data is extracted along with software measurements for each function. This may sound like a gargantuan task, but it can be elegantly performed using the DePress framework which is written in Java (Madeyski and Majchrzak, 2014). This extracted data is combined to make an historic code SDP dataset. The final component of Step 1 is to use the BCF algorithm which was proposed in Chapter 6 to build a decision forest classifier.

Step 2 is performed each time a programmer appends code to a function (denoted as \( f \)) within some source code. For example, consider that the programmer has declared a new variable named `height` on a new line. This triggers the back-end
to re-measure \( f \). This re-measurement can be handled by using the DePress framework (Madeyski and Majchrzak, 2014). Using the re-measured metrics, the code metrics dataset is updated. This dataset holds the software metrics for the current software project’s source code. In the example, the code metrics for \( f \) will increment both the number of variable and the line of code count. Using the SCORE stack extracted from the BCF classifier, the current function is classified. This classification is then presented to the user by displaying the expected cost of \( f \). Additionally, the sibling rule set associated with the classification of \( f \) is used as a suggestion for reducing the expected cost of \( f \). For example, if \( f \) is classified by the logic rule \( \text{if } \text{lines\_of\_code} > 45 \text{ and } \text{percentage\_comments} \leq 10 \text{ then } \text{cost} = 215.16 \), and its sibling rule is \( \text{if } \text{lines\_of\_code} > 45 \text{ and } \text{percentage\_comments} > 10 \text{ then } \text{cost} = 124.55 \), the suggestion to the programmer might read: "To reduce the expected cost of \( f \), try adding more comments."

### 8.1.2 Front-End Interface

The previous subsection described the back-end architecture through Figure 8.1. The front-end interface is closely related to Step 2. As can be seen in Figure 8.2, the user will see green or red bars to the left of the line numbers. Each bar corresponds to a function in the source code. If the bar is green then its expected cost is lower than the target cost threshold (a parameter set by the software development organisation). The bar is red otherwise. When the user moves the cursor over a bar, they will see the function tip. This tip is comprised of several elements: The function’s current expected cost; Whether or not the cost is below the target cost threshold; A suggestion for reducing the expected cost. The suggestion is the best one according to the SCORE stack; A button to display a different suggestion (the next best one according to the SCORE stack); A button to explain how the suggestion is calculated.

Figure 8.3 is similar to Figure 8.2 except the function tip is green, indicating that its expected cost is lower than the target cost threshold. A suggestion is also shown here for reducing the cost.

When a user clicks on the button which says "How is this calculated?" they are shown a modal screen. This can be seen in Figure 8.4. This screen describes the leaf which was used to classify the current function by listing the conditions which were
satisfied. After reporting the expected cost of that leaf, it describes the conditions of the sibling leaf which has a lower expected cost.

Figure 8.5 shows the screen which appears from the right after the user presses the cost list icon in the top right. It is the list of functions where they are ordered by their expected cost. The user can then select a function from this list to be taken to its location in the source code. The costs are also shown for each function. These costs are also written in green or red depending on whether their expected cost is below or above the target cost threshold.

This interface provides answers to the research questions initially stated in Section 3.1. The SDP classifier which is built using the back-end architecture in Section 8.1.1 is used to report the expected cost to a developer as they are programming. This answers research question 1. Figures 8.2 and 8.3 demonstrate that research question 2 has also been answered since the cost-sensitive knowledge is reported to the developer in a way which can facilitate immediate action.
DePress.java
code_metrics.java
Code Metrics
predict.java
classify.java
tree.java
Decision Trees
console.java
about.txt
readme.md
Source
Data
SDP
New
data.csv
code_metrics.java
tree.java

public void setPositiveClassIndex(int positiveClassIndex) {
    this.positiveClassIndex = positiveClassIndex;
    if (positiveClassIndex == 0) {
        negativeClassIndex = 1;
    } else {
        negativeClassIndex = 0;
    }
}

public double classifyInstance(Instance instance) throws Exception {
    Node leaf = getLeaf(instance);
    double costOfLabelingPositive = 0.0;
    double costOfLabelingNegative = 0.0;
    Instances instances = leaf.getInstances();
    costOfLabelingPositive = costOfLabellingPositive(instances);
    costOfLabelingNegative = costOfLabellingNegative(instances);
    if (costOfLabelingPositive <= costOfLabelingNegative) {
        return (double) positiveClassIndex;
    } else {
        return (double) negativeClassIndex;
    }
}

public void buildClassifier(Instances instances) throws Exception {
    rootNode = new Node(instances, null);
    rootNode.grow();
    prune(rootNode);
}

We predict that adding 6 extra lines of comments to this function will reduce the expected cost to $120.

Next suggestion
How is this calculated?

This function exceeds the target cost threshold set by your organisation.

Expected cost: $132

FIGURE 8.2: The user has selected the function information tip for the function: ClassifyInstance.
To optimise further, we suggest adding more variables to reduce the complexity of your code. Expected cost: $58. How is this calculated?

Good job! This function is below the target cost threshold set by your organisation.
8.1. IDE Integration Concept

We predict that adding 6 extra lines of comments to this function will reduce the expected cost to $120. How is this calculated?

Next suggestion:

This function exceeds the target cost threshold set by your organisation.

The selected function falls in a cluster with the following conditions:
- Less than or equal 5 comments
- Greater than 11 variables
- Greater than 100 lines

Based on historical data, each function in this cluster typically costs approximately $132.

Our suggestion is based on a very similar cluster which has the following conditions:
- Less than or equal 5 comments
- Greater than 11 variables
- Less than or equal to 100 lines

Based on historical data, each function in this cluster typically costs approximately $120.
DePress.java
code_metrics.java
Code Metrics
predict.java
classify.java
tree.java
Decision Trees
console.java
about.txt
readme.md
Source
Data
SDP
New
data.csv
code_metrics.java
tree.java

\[
\begin{align*}
\text{public} & \quad \text{void} \quad \text{setPositiveClassIndex} \quad (\text{int} \quad \text{positiveClassIndex}) \\
& \quad \text{this}.\quad \text{positiveClassIndex} \quad = \quad \text{positiveClassIndex}; \\
& \quad \text{if} \quad (\text{positiveClassIndex} \quad == \quad 0) \quad \text{else} \quad \text{negativeClassIndex} \quad = \quad 0;
\end{align*}
\]

\[
\begin{align*}
\text{public} & \quad \text{double} \quad \text{classifyInstance} \quad (\text{Instance} \quad \text{instance}) \quad \text{throws} \quad \text{Exception} \\
& \quad \text{Node} \quad \text{leaf} \quad = \quad \text{getLeaf} \quad (\text{instance}); \\
& \quad \text{double} \quad \text{costOfLabelingPositive} \quad = \quad 0.0; \\
& \quad \text{double} \quad \text{costOfLabelingNegative} \quad = \quad 0.0; \\
& \quad \text{Instances} \quad \text{instances} \quad = \quad \text{leaf}.\quad \text{getInstances}(); \\
& \quad \text{costOfLabelingPositive} \quad = \quad \text{costOfLabellingPositive} \quad (\text{instances}); \\
& \quad \text{costOfLabelingNegative} \quad = \quad \text{costOfLabellingNegative} \quad (\text{instances}); \\
& \quad \text{if} \quad (\text{costOfLabelingPositive} \quad \leq \quad \text{costOfLabelingNegative}) \quad \text{else} \quad \text{return} \quad (\text{double}) \quad \text{positiveClassIndex}; \\
& \quad \text{return} \quad (\text{double}) \quad \text{negativeClassIndex};
\end{align*}
\]

\[
\begin{align*}
\text{public} & \quad \text{void} \quad \text{buildClassifier} \quad (\text{Instances} \quad \text{instances}) \quad \text{throws} \quad \text{Exception} \\
& \quad \text{rootNode} \quad = \quad \text{new} \quad \text{Node} \quad (\text{instances}, \quad \text{null}); \\
& \quad \text{rootNode}.\quad \text{grow}(); \\
& \quad \text{prune} \quad (\text{rootNode});
\end{align*}
\]

\[
\begin{align*}
& \quad \text{public} \quad \text{void} \quad \text{buildClassifier} \quad (\text{Instances} \quad \text{instances}) \quad \text{throws} \quad \text{Exception} \\
& \quad \text{rootNode} \quad = \quad \text{new} \quad \text{Node} \quad (\text{instances}, \quad \text{null}); \\
& \quad \text{rootNode}.\quad \text{grow}(); \\
& \quad \text{prune} \quad (\text{rootNode});
\end{align*}
\]

\[
\begin{align*}
\text{Function Costs} \\
\text{Sorted by: Cost (highest → lowest)}
\end{align*}
\]

\[
\begin{align*}
\text{buildClassifier} \quad $58 & \quad \text{classifyInstance} \quad $132 & \quad \text{pruneTree} \quad $126 & \quad \text{estimateCost} \quad $102 & \quad \text{decompose} \quad $98 & \quad \text{measureEntropy} \quad $93 & \quad \text{optimiseParameter} \quad $85 & \quad \text{countRules} \quad $78 & \quad \text{extractSiblings} \quad $71 & \quad \text{nullClassifier} \quad $68
\end{align*}
\]

FIGURE 8.5: The user has opened the dialog which displays an ordered list of all functions and their expected costs.
8.2 Challenges with implementing the described concept

8.2.1 Converting rule sets to everyday language

Each rule set in the system needs to be understandable by the programmer. This may require some translation to everyday language. Each attribute in the training dataset should have a strategy for converting it to everyday language. More specifically, if an attribute $\alpha$ is present in a rule set, it requires a translation method such that the programmer will understand and be able to take action as a result. If $\alpha$ is percentage of lines which are comments, then the translation may read: "This module currently has 130 lines and 10 of them are comments. For this module, the estimated optimal number of lines that are comments is 17." Note that the user was not told the optimal percentage directly as this is less actionable than telling the programmer the number of lines that are suggested. This means that the optimal number of comments is calculated by the system rather than the programmer. Therefore, when implementing the proposed concept, it is up to the implementers to define a method for translating the possible rule sets to everyday language.

8.2.2 Meaningfulness of expected cost

The expected cost requires accurate definitions of $C_{TP}$, $C_{TN}$, $C_{FP}$ and $C_{FN}$. If this is not possible, the expected costs reported for each function could lack meaning for the end user. In this situation, the probability of being defective would be more useful. This is simply taken as the probabilistic output of the classifier. For example, rather than reporting that a function $f$ has an expected cost of $357$, it would instead report that function $f$ has a 80.3% chance of being defective.

8.3 The Future

Industry Application Study: An industry-based evaluation of the proposed system would be useful for two major reasons. First, using the net profit approach of evaluation SDP systems (Ling et al., 2006; Sheng et al., 2014) would discover whether the proposed approach is worth the overhead costs. Secondly, it would discover how much money is gained through net savings by using the proposed system.
Enhancing the System with More Knowledge Visualisation Techniques: Currently, the knowledge reported to the user is in the form of plain text and decision trees models. Any future work which incorporates more ways of communicating knowledge to a developer will be useful.

Investigation of the change in performance by introducing active learning into BCF: Some patterns may be difficult to learn solely by a machine. Active learning helps to overcome this issue by asking a human expert on how to classify difficult to classify records. This could be introduced into BCF by setting a confidence threshold. For example, if a leaf is produced by BCF which has confidence less than the defined threshold, a human expert is used to redefine it’s splitting test condition. Once the test has been redefined, the process would continue.

BCF model self awareness of weakness: Consider a scenario where a BCF model is deployed to regularly solve problems. Each Sunday at midnight, the model’s last 7 days of predictions are evaluated based on their accuracy. More specifically, each logic rule is evaluated on it’s accuracy. The model is then retrained. If the new model contains logic rules which were similar to poorly performing rules from the last model then the new model could prune that logic rule from the forest.

Application of BCF to other domains: This thesis focuses on applying cost-sensitive learning to software defect prediction. However, the proposed approaches could be applied to other domains. For example, predicting whether or not credit card transactions are fraudulent (Bahnsen et al., 2015). Such a study could identify several domains which are cost-sensitive and explore which domains BCF works well for, which domains BCF does not work well for, and why.

A general framework for cost-sensitive knowledge discovery: It would be useful to have a framework similar to BCF but each major component could be swapped out for another algorithm. For example, the major components might be the balancing component, the cost-sensitive component and the dataset. Using such a framework, the best combinations could be found for each application of BCF.

Streaming data: An algorithm for updating the model built by BCF might allow streamed data. If a new record has been written to the source database, then it could be used to update BCF’s model. For example, the leaf that the new record falls in
within the BCF model could be added to the class counts of that leaf. This would re-
sult in the new record affecting any future predictions. If this could be implemented
efficiently then this would allow streamed data. It is worth mentioning that adding
records to the trained model this way would result in a model which is not the same
as if the model was retrained from scratch using all the data. However, this approach
could be used for a short period before the entire model is re-trained. That is, the
model might be re-trained from scratch each Sunday, but throughout the week new
records can be added in the way described above.
Chapter 9

Conclusion

Classification is a major task in data science. Not only can it be used for making predictions on future data, it can be used to discover patterns in existing data. Chapters 6, 7 and 8 used classification to predict which sections of source code contained bugs. Chapters 7 and 8 used classification to discover patterns which lead to bugs in source code. Datasets used for training classifiers commonly have at least one of many challenges. These challenges can include high dimensionality (Chen and Wasikowski, 2008), class overlap (Alejo et al., 2013), missing values (Rahman and Islam, 2013), cost-sensitivity (Sheng et al., 2014) and class imbalance (Tahir et al., 2012). This thesis focused on the latter two challenges which are both present in software defect data (Wang and Yao, 2013), (Ling et al., 2006).

The class imbalance problem can negatively affect the performance of classifiers. It occurs when records of one class are much more frequent than records of other classes. When trained on such a dataset, a classifier will typically be biased towards predicting future records as belonging to the highest frequency class (called the majority class). Existing methods for negating the effects of class imbalance typically do so by undersampling and oversampling to create a more balanced training dataset (Tahir et al., 2012), (Chawla et al., 2002), (Lin et al., 2017). Chapter 4 proposed an approach for negating the effects of class imbalance named Standoff. A strength of the proposed approach is that the entirety of the original training dataset is used. The experiments found that the proposed Standoff achieved a higher AUC than existing techniques for performing classification on class imbalanced datasets (See Table 9.1 for the results. They are also visualised in Figure 9.1).

A drawback of Standoff is its computation speed. This is largely due to Standoff’s
### Table 9.1: AUC Comparison of the Methods (Higher the Better)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Original</th>
<th>CSTree</th>
<th>SMOTE</th>
<th>Adasyn</th>
<th>MetaCost</th>
<th>Standoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pima</td>
<td>0.735</td>
<td>0.720</td>
<td>0.740</td>
<td>0.700</td>
<td>0.725</td>
<td>0.757</td>
</tr>
<tr>
<td>glass0</td>
<td>0.770</td>
<td>0.820</td>
<td>0.817</td>
<td>0.764</td>
<td>0.784</td>
<td>0.836</td>
</tr>
<tr>
<td>haberman</td>
<td>0.604</td>
<td>0.572</td>
<td>0.627</td>
<td>0.641</td>
<td>0.646</td>
<td>0.647</td>
</tr>
<tr>
<td>ecoli1</td>
<td>0.924</td>
<td>0.899</td>
<td>0.925</td>
<td>0.877</td>
<td>0.866</td>
<td>0.929</td>
</tr>
<tr>
<td>BreastCancer</td>
<td>0.939</td>
<td>0.943</td>
<td>0.935</td>
<td>0.901</td>
<td>0.945</td>
<td>0.952</td>
</tr>
<tr>
<td>Average</td>
<td>0.794</td>
<td>0.791</td>
<td>0.809</td>
<td>0.777</td>
<td>0.793</td>
<td>0.824</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.140</td>
<td>0.149</td>
<td>0.130</td>
<td>0.112</td>
<td>0.117</td>
<td>0.126</td>
</tr>
</tbody>
</table>

**Figure 9.1:** AUC Comparison of the Methods (Higher the Better) visualised. Those who are reading this in black-and-white are referred to Table 9.1 which presents the same information but in further detail.

The requirement to cluster the training dataset twice; once on the majority records and once on the minority records. This thesis refers to this as class-specific-clustering. To perform it, an existing clustering algorithm is run twice on the training dataset. This can be time consuming since clustering algorithms are typically computationally expensive (Rahman and Islam, 2014), (Frey and Dueck, 2007). In order to further experiment with Standoff, the computation time needed to be reduced. To address this issue, this thesis proposed a method to perform class-specific-clustering quicker than the existing methods without compromising on the quality. This method, RB-Clust, was proposed in Chapter 5. RBClust is the first clustering algorithm designed for class-specific-clustering. This means that it only needs to be run once on the training dataset. It also performs quicker than traditional clustering algorithms since it uses the results from a decision tree to find the class-specific-clusters. This thesis’
Chapter 9. Conclusion

153

experiments on RBClust found that the proposed RBClust achieved a higher silhouette coefficient than existing clustering algorithms (See Table 9.2. The results in that table are also visualised in Figure 9.2).

<table>
<thead>
<tr>
<th>&lt;Dataset-ClassValue&gt;</th>
<th>#Records</th>
<th>#Attributes</th>
<th>kMeans</th>
<th>AP</th>
<th>RBClust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfusion-1</td>
<td>178</td>
<td>5</td>
<td>0.307</td>
<td>0.179</td>
<td>0.408</td>
</tr>
<tr>
<td>Transfusion-0</td>
<td>570</td>
<td>5</td>
<td>-0.028</td>
<td>0.074</td>
<td>0.227</td>
</tr>
<tr>
<td>ecoli1-P</td>
<td>77</td>
<td>8</td>
<td>0.26</td>
<td>0.194</td>
<td>0.345</td>
</tr>
<tr>
<td>ecoli1-N</td>
<td>259</td>
<td>8</td>
<td>-0.006</td>
<td>0.184</td>
<td>0.235</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td>0.133</td>
<td>0.158</td>
<td><strong>0.304</strong></td>
</tr>
<tr>
<td><strong>Standard Deviation</strong></td>
<td></td>
<td></td>
<td>0.175</td>
<td>0.056</td>
<td>0.088</td>
</tr>
</tbody>
</table>

**TABLE 9.2: Methods Comparison - Silhouette Coefficient (The higher the better.)**

![Silhouette Coefficient Graph](image)

**FIGURE 9.2: Methods Comparison - Silhouette Coefficient (The higher the better) - visualised. Those who are reading this in black-and-white are referred to Table 9.2 which presents the same information but in further detail.**

One of the major topics of this thesis is cost-sensitive classification. The algorithms mentioned above are cost-insensitive. That is, they do not produce cost-sensitive classifiers. This thesis’ next aim was to extend upon the above-mentioned proposed algorithms such that a cost-sensitive classifier could be produced from
class imbalance data. To achieve this, this thesis proposed Balanced Costs Framework or BCF in Chapter 6. The beginning of Chapter 6 proposed two names to distinguish between two different types of costs. The first is balancing costs. These costs are passed to a cost-sensitive classifier to negate the negative effects of class imbalance. This is typically done by reflecting the imbalance ratio in the cost-matrix (see Chapter 6 for more details). The second type of costs is domain costs. These are the costs which represent the real life costs incurred by predictions. For example, the cost of predicting some defective code as defect-free has a high domain cost since it will cause buggy behaviour in the software.

Chapter 3 described how cost-sensitive classification techniques and class balancing techniques are very similar. This similarity is that they both aim to adjust bias. The former induces bias and the latter reduces bias. When performing cost-sensitive classification on class imbalanced data, a data scientist might use two techniques, one for reducing the class imbalance bias, and one for inducing cost-sensitive bias. Chapter 6, proposed a technique for performing cost-sensitive classification on class imbalanced data. The proposed framework, named BCF, combines balancing costs and domain costs so that bias is only adjusted once. The BCF framework uses the above-mentioned Standoff and RBClust to generate balancing costs which are combined with the user-provided domain costs. The experiments found that BCF builds cost-sensitive classifiers on class imbalanced datasets which achieve a lower total cost than existing methods (see Table 9.3 and Figure 9.3).

An advantage of BCF is that it produces a cost-sensitive decision forest where each tree in the forest directly describes the entire original training dataset. Since decision tree classifiers can be visually inspected to discover patterns in the data, BCF provides a solid foundation for cost-sensitive knowledge discovery. A challenge of using BCF in this way is that there is no process to follow. To address this challenge of BCF, three concepts were proposed in Chapter 7. First, an equation for measuring the interestingness of a pattern based on the potential cost savings it may provide. Second, an algorithm for the automatic extraction of all patterns including patterns which exist in all possible sub-trees. Third, a process for a data scientist to follow which uses all of the proposed methods to produce a cost-sensitive knowledge discovery report. The proposed techniques and concepts were demonstrated
by using them to discover knowledge from NASA software defects. On average, the proposed method for knowledge discovery found patterns which were 50% more interesting than an existing method on average (see Figure 9.4).

As a discussion point at the end of this thesis (see Chapter 8, a design of a conceptual approach for how the proposed algorithms and concepts could be utilised in practise was presented. More specifically, a software tool was designed for integrating the proposed methods into the software development process. It allows a programmer to see cost-sensitive knowledge and analysis of their code as they are programming. The aim of this tool would be to guide the programmer in producing code which is optimised for business cost.

Chapter 1 specified the three research questions of this thesis. This chapter now re-states each research question and describes how it has been answered.

**Research Question 1**: How can classification performance be further improved when the training data is class imbalanced?

This research question was explored in Chapter 4. That chapter proposed Stand-off which achieves a higher AUC than existing methods for performing classification

---

**Table 9.3**: Total cost comparison (lower is better). Highlighted cells indicate the best performing methods for the corresponding dataset. The upper portion of this table indicates the properties of the corresponding method. ✓ indicates the presence of a property and X denotes the opposite.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BCSForest</th>
<th>CSForest</th>
<th>CSTree</th>
<th>IRUS</th>
<th>Adasyn + C4.5</th>
<th>SysFor + Voting 1</th>
<th>SysFor + Voting 2</th>
<th>ForestPA</th>
<th>BCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>KC3'</td>
<td>166</td>
<td>174</td>
<td>170</td>
<td>157</td>
<td>155</td>
<td>156</td>
<td>160</td>
<td>178</td>
<td>153</td>
</tr>
<tr>
<td>MC2'</td>
<td>128</td>
<td>135</td>
<td>128</td>
<td>125</td>
<td>132</td>
<td>167</td>
<td>160</td>
<td>175</td>
<td>123</td>
</tr>
<tr>
<td>MW1'</td>
<td>128</td>
<td>136</td>
<td>173</td>
<td>179</td>
<td>147</td>
<td>122</td>
<td>130</td>
<td>136</td>
<td>118</td>
</tr>
<tr>
<td>CM1'</td>
<td>215</td>
<td>206</td>
<td>218</td>
<td>228</td>
<td>208</td>
<td>212</td>
<td>214</td>
<td>210</td>
<td>203</td>
</tr>
<tr>
<td>AR1</td>
<td>193</td>
<td>125</td>
<td>188</td>
<td>185</td>
<td>154</td>
<td>133</td>
<td>133</td>
<td>121</td>
<td>129</td>
</tr>
<tr>
<td>AR3</td>
<td>81</td>
<td>72</td>
<td>135</td>
<td>78</td>
<td>101</td>
<td>73</td>
<td>69</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>AR4</td>
<td>188</td>
<td>113</td>
<td>220</td>
<td>215</td>
<td>201</td>
<td>126</td>
<td>131</td>
<td>115</td>
<td>107</td>
</tr>
<tr>
<td>AR5</td>
<td>53</td>
<td>36</td>
<td>44</td>
<td>51</td>
<td>53</td>
<td>41</td>
<td>44</td>
<td>39</td>
<td>36</td>
</tr>
<tr>
<td>AR6</td>
<td>214</td>
<td>111</td>
<td>228</td>
<td>187</td>
<td>197</td>
<td>215</td>
<td>218</td>
<td>105</td>
<td>99</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>151.78</td>
<td>123.11</td>
<td>167.11</td>
<td>156.11</td>
<td>149.78</td>
<td>139.11</td>
<td>139.89</td>
<td>127.00</td>
<td>114.67</td>
</tr>
</tbody>
</table>

---
on class imbalanced datasets. From Table 9.1, one can observe that Standoff produces competitive results, outperforming the studied existing techniques. A weakness of Standoff was its reliance on a clustering algorithm. Standoff required a clustering algorithm to execute on both the majority class records and the minority class records. Since clustering is typically a computationally expensive task (Rahman and Islam, 2014), (Frey and Dueck, 2007), Standoff could require a long time to execute. To address this, this thesis defined the class-specific-clustering problem and proposed RBClust to solve it. The time-savings allowed us to further experiment with Standoff and include it in the explorations of Research Questions 2 and 3.
Research Question 2: For the purposes of building lower cost classifiers, how can strategy for addressing class imbalance be integrated into a cost-sensitive decision forest algorithm?

Building upon the proposed algorithms Standoff and RBClust (proposed in Chapters 4 and 5 respectively), this thesis introduced BCF, a method for training cost-sensitive classifiers from class imbalanced data. To perform cost-sensitive classification on a dataset which is class imbalanced, a data scientist might use a class balancing technique, then use a cost-sensitive classification method. Using the terminology defined in Chapter 3, the data scientist would be reducing bias, and then inducing bias. Since BCF combines balancing costs and domain costs, it adjusts bias once, removing one step in the process, and therefore, one avenue for error. The experiments found that BCF builds classifiers which achieve lower total cost than the studied existing techniques (see Table 9.3).

Research Question 3: How can a cost-sensitive decision forest be better utilised to perform cost-sensitive knowledge discovery on class imbalanced datasets?

BCF, proposed in Chapter 6 provided a strong foundation for cost-sensitive knowledge discovery from class imbalanced data. However, it lacked a process which would guide a data scientist in doing so. This challenge was addressed in Chapter 7. A method was proposed for measuring the cost-sensitive interestingness of a pattern extracted from a decision forest. An algorithm was also proposed which automatically searches each tree in a forest for all possible sub-trees and extracts a stack of all patterns, ordered by the proposed interestingness measure. Also proposed was

Figure 9.4: Average SibCost for the top 5 most interesting rule sets (higher the better).
Breadth-First-Mining, a cost-sensitive knowledge discovery process which uses the above-mentioned stack of patterns to produce a knowledge discovery report. This is demonstrated in Chapter 7 in which knowledge discovery is performed on software defect data collected from several software projects undertaken at NASA.
Appendix A

Datasets used in Survey Experiments

Chapter 3 presented the results of experiments over 99 datasets. The characteristics of those datasets are presented in Tables A.1 and A.2. These datasets were accessed from the KEEL dataset library (Fernández et al., 2007). The dataset characteristics were also accessed from KEEL. The imbalance ratios of these datasets ranges from slightly imbalanced to heavily imbalanced. The smallest imbalance ratio of these datasets is 1.82 and the highest is 129.44. This provides a diverse set of datasets to perform experiments on class imbalance treatment.
## Appendix A. Datasets used in Survey Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Records</th>
<th># Attributes</th>
<th># Categorical Attributes</th>
<th># Numerical Attributes</th>
<th>Imbalance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>glass1</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>1.82</td>
</tr>
<tr>
<td>ecoli-0_vs_1</td>
<td>220</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>1.96</td>
</tr>
<tr>
<td>wisconsin</td>
<td>683</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>1.96</td>
</tr>
<tr>
<td>pima</td>
<td>768</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>1.87</td>
</tr>
<tr>
<td>iris0</td>
<td>150</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>glass0</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>2.06</td>
</tr>
<tr>
<td>yeast1</td>
<td>1484</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>2.46</td>
</tr>
<tr>
<td>haberman</td>
<td>306</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>2.78</td>
</tr>
<tr>
<td>vehicle2</td>
<td>846</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>2.88</td>
</tr>
<tr>
<td>vehicle1</td>
<td>846</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>2.9</td>
</tr>
<tr>
<td>vehicle3</td>
<td>846</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>2.99</td>
</tr>
<tr>
<td>glass-0-1-2-3_vs_4-5-6</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>3.2</td>
</tr>
<tr>
<td>vehicle0</td>
<td>846</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>3.25</td>
</tr>
<tr>
<td>ecoli1</td>
<td>336</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>3.36</td>
</tr>
<tr>
<td>new-thyroid1</td>
<td>215</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>5.14</td>
</tr>
<tr>
<td>new-thyroid2</td>
<td>215</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>5.14</td>
</tr>
<tr>
<td>ecoli2</td>
<td>336</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>5.46</td>
</tr>
<tr>
<td>segment0</td>
<td>2308</td>
<td>19</td>
<td>0</td>
<td>19</td>
<td>6.02</td>
</tr>
<tr>
<td>glass6</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>6.38</td>
</tr>
<tr>
<td>yeast3</td>
<td>1484</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>8.1</td>
</tr>
<tr>
<td>ecoli3</td>
<td>336</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>8.6</td>
</tr>
<tr>
<td>page-blocks0</td>
<td>5472</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>8.79</td>
</tr>
<tr>
<td>yeast-2_vs_4</td>
<td>514</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>9.08</td>
</tr>
<tr>
<td>yeast-0-5-6-7-9_vs_4</td>
<td>528</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>9.35</td>
</tr>
<tr>
<td>vowel0</td>
<td>988</td>
<td>13</td>
<td>0</td>
<td>13</td>
<td>9.98</td>
</tr>
<tr>
<td>glass-0-1-6_vs_2</td>
<td>192</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>10.29</td>
</tr>
<tr>
<td>glass2</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>11.59</td>
</tr>
<tr>
<td>shuttle-c0-c8-c4</td>
<td>1829</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>13.87</td>
</tr>
<tr>
<td>yeast-1_vs_7</td>
<td>459</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>14.3</td>
</tr>
<tr>
<td>glass4</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>15.47</td>
</tr>
<tr>
<td>ecoli4</td>
<td>336</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>15.8</td>
</tr>
<tr>
<td>page-blocks1-3_vs_4</td>
<td>472</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>15.86</td>
</tr>
<tr>
<td>abalone9-18</td>
<td>472</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>16.4</td>
</tr>
<tr>
<td>glass-0-1-6_vs_5</td>
<td>184</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>19.44</td>
</tr>
<tr>
<td>shuttle-c2-c8-c4</td>
<td>129</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>20.5</td>
</tr>
<tr>
<td>yeast-1-4-5-8_vs_7</td>
<td>693</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>22.1</td>
</tr>
<tr>
<td>glass5</td>
<td>214</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>22.78</td>
</tr>
<tr>
<td>yeast-2_vs_8</td>
<td>482</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>23.1</td>
</tr>
<tr>
<td>yeast4</td>
<td>1484</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>28.1</td>
</tr>
<tr>
<td>yeast-1-2-8-9_vs_7</td>
<td>947</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>30.57</td>
</tr>
<tr>
<td>yeast5</td>
<td>1484</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>32.73</td>
</tr>
<tr>
<td>ecoli-0-1-3-7_vs_2-6</td>
<td>281</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>39.14</td>
</tr>
<tr>
<td>yeast6</td>
<td>1484</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>41.4</td>
</tr>
<tr>
<td>abalone19</td>
<td>4174</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>129.44</td>
</tr>
<tr>
<td>ecoli-0-3-4_vs_5</td>
<td>200</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>ecoli-0-6-7_vs_3-5</td>
<td>222</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.09</td>
</tr>
<tr>
<td>ecoli-0-2-3-4_vs_5</td>
<td>202</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.1</td>
</tr>
<tr>
<td>glass-0-1-5_vs_2_</td>
<td>172</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.12</td>
</tr>
<tr>
<td>yeast-0-3-5-9_vs_7-8</td>
<td>506</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>9.12</td>
</tr>
<tr>
<td>yeast-0-2-5-7-9_vs_3-6-8</td>
<td>1004</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>9.14</td>
</tr>
<tr>
<td>yeast-0-2-5-6_vs_3-7-8-9</td>
<td>1004</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>9.14</td>
</tr>
<tr>
<td>ecoli-0-4-6_vs_5</td>
<td>203</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>9.15</td>
</tr>
<tr>
<td>ecoli-0-1_vs_2-3-5_</td>
<td>244</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.17</td>
</tr>
<tr>
<td>ecoli-0-2-6-7_vs_3-5_</td>
<td>224</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.18</td>
</tr>
<tr>
<td>glass-0-4_vs_5_</td>
<td>92</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>9.22</td>
</tr>
<tr>
<td>ecoli-0-3-4-6_vs_5</td>
<td>205</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.25</td>
</tr>
<tr>
<td>ecoli-0-3-4-7_vs_5-6</td>
<td>257</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9.28</td>
</tr>
<tr>
<td>ecoli-0-6-7_vs_5</td>
<td>220</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>ecoli-0-1-4-7_vs_2-3-5-6</td>
<td>336</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>10.59</td>
</tr>
<tr>
<td>led7digit-0-2-4-5-6-7-8-9_vs_1</td>
<td>443</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>10.97</td>
</tr>
<tr>
<td>glass-0-6_vs_5_</td>
<td>108</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>ecoli-0-1_vs_5</td>
<td>240</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>glass-0-1-4-6_vs_2</td>
<td>205</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>11.06</td>
</tr>
<tr>
<td>ecoli-0-1-4-7_vs_5-6</td>
<td>332</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>12.28</td>
</tr>
<tr>
<td>Cleveland-0_vs_4</td>
<td>177</td>
<td>13</td>
<td>0</td>
<td>13</td>
<td>12.62</td>
</tr>
</tbody>
</table>

Table A.1: Characteristics of the datasets used in Chapter 3. - Part 1
### Appendix A. Datasets used in Survey Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Records</th>
<th># Attributes</th>
<th># Categorical Attributes</th>
<th># Numerical Attributes</th>
<th>Imbalance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecoli-0-1-4-6_vs_5</td>
<td>280</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>dermatology-6</td>
<td>358</td>
<td>34</td>
<td>0</td>
<td>34</td>
<td>16.9</td>
</tr>
<tr>
<td>zoo-3</td>
<td>101</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>19.2</td>
</tr>
<tr>
<td>shuttle-6_vs_2-3</td>
<td>230</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>lymphography-normal-fibrosis</td>
<td>148</td>
<td>18</td>
<td>15</td>
<td>3</td>
<td>23.67</td>
</tr>
<tr>
<td>flare-F</td>
<td>1066</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>23.79</td>
</tr>
<tr>
<td>car-good</td>
<td>1728</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>24.04</td>
</tr>
<tr>
<td>car-vgood</td>
<td>1728</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>25.58</td>
</tr>
<tr>
<td>kr-vs-k-zero-one_vs_draw</td>
<td>2901</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>26.63</td>
</tr>
<tr>
<td>kr-vs-k-one_vs_fifteen</td>
<td>2244</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>27.77</td>
</tr>
<tr>
<td>winequality-red-4</td>
<td>1599</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>29.17</td>
</tr>
<tr>
<td>poker-9_vs_7</td>
<td>244</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>29.5</td>
</tr>
<tr>
<td>kddcup-guess_passwd_vs_satan</td>
<td>1642</td>
<td>41</td>
<td>15</td>
<td>26</td>
<td>29.98</td>
</tr>
<tr>
<td>abalone-3_vs_11</td>
<td>502</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>32.47</td>
</tr>
<tr>
<td>winequality-white-9_vs_4</td>
<td>168</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>32.6</td>
</tr>
<tr>
<td>kr-vs-k-three_vs_eleven</td>
<td>2935</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>35.23</td>
</tr>
<tr>
<td>winequality-red-8_vs_6</td>
<td>656</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>35.44</td>
</tr>
<tr>
<td>abalone-17_vs_7-8-9-10</td>
<td>2338</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>39.31</td>
</tr>
<tr>
<td>abalone-21_vs_8</td>
<td>581</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>40.5</td>
</tr>
<tr>
<td>winequality-white-3_vs_7</td>
<td>900</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>44</td>
</tr>
<tr>
<td>winequality-red-8_vs_6-7</td>
<td>855</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>46.5</td>
</tr>
<tr>
<td>kddcup-land_vs_portsweep</td>
<td>1061</td>
<td>41</td>
<td>15</td>
<td>26</td>
<td>49.52</td>
</tr>
<tr>
<td>abalone-19_vs_10-11-12-13</td>
<td>1622</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>49.69</td>
</tr>
<tr>
<td>kr-vs-k-zero_vs_eight</td>
<td>1460</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>53.07</td>
</tr>
<tr>
<td>winequality-white-3-9_vs_5</td>
<td>1482</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>58.28</td>
</tr>
<tr>
<td>poker-8-9_vs_6</td>
<td>1485</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>58.4</td>
</tr>
<tr>
<td>shuttle-2_vs_5</td>
<td>3316</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>66.67</td>
</tr>
<tr>
<td>winequality-red-3_vs_5</td>
<td>691</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>66.1</td>
</tr>
<tr>
<td>abalone-20l_vs_8-9-10</td>
<td>1916</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>72.69</td>
</tr>
<tr>
<td>kddcup-buffer_overflow_vs_back</td>
<td>2233</td>
<td>41</td>
<td>15</td>
<td>26</td>
<td>73.43</td>
</tr>
<tr>
<td>kddcup-land_vs_satan</td>
<td>1610</td>
<td>41</td>
<td>15</td>
<td>26</td>
<td>75.67</td>
</tr>
<tr>
<td>kr-vs-k-zero_vs_fifteen</td>
<td>2193</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>80.22</td>
</tr>
<tr>
<td>poker-8-9_vs_5</td>
<td>2075</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>82</td>
</tr>
<tr>
<td>poker-8_vs_6</td>
<td>1477</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>85.88</td>
</tr>
</tbody>
</table>

**Table A.2:** Characteristics of the datasets used in Chapter 3. - Part 2
Bibliography


Cieslak, David A. and Chawla, Nitesh V. “Learning decision trees for unbalanced data”. In: Proceedings of the 2008 Joint European Conference on Machine Learning and


Han, Jiawei, Pei, Jian, and Kamber, Micheline. Data mining: concepts and techniques. Elsevier, 2011.


Lin, Wei-Chao, Tsai, Chih-Fong, Hu, Ya-Han, and Jhang, Jin-Shang. “Clustering-based undersampling in class-imbalanced data”. In: Information Sciences 409 (Oct. 2017), pp. 17–26. URL: https://doi.org/10.1016/j.ins.2017.05.008.


Shepperd, Martin, Song, Qinhao, Sun, Zhongbin, and Mair, Carolyn. “Data quality: Some comments on the nasa software defect datasets”. In: IEEE Transactions on Software Engineering 39 (Sept. 2013), pp. 1208–1215. URL: https://doi.org/10.1109/TSE.2013.11.


Siers, Michael J. and Islam, Md Zahidul. “Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem”. In: Information Systems 51 (July 2015), pp. 2–71. URL: https://doi.org/10.1016/j.is.2015.02.006.

Siers, Michael J. and Islam, Md Zahidul. “Standoff-balancing: a novel class imbalance treatment method inspired by military strategy”. In: Proceedings of the 28th Australasian Joint Conference on Artificial Intelligence. Ed. by Bernhard Pfahringer and


Wasikowski, Mike and Chen, Xue-wen. “Combating the Small Sample Class Imbalance Problem Using Feature Selection”. In: IEEE Transactions on Knowledge and Data Engineering 22 (Oct. 2010), pp. 1388–1400. URL: https://doi.org/10.1109/TKDE.2009.187.


