The Development of a Semi-Automated Testing Method for Medical Imaging Systems

By

Karen J. Laurin, B. Eng

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Applied Science

Ottawa-Carleton Institute for
Electrical and Computer Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
December 2009

Copyright ©
2009 – Karen J. Laurin

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-63463-9
Our file *Notre référence*
ISBN: 978-0-494-63463-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

**Development of a Testing Framework for Medical Imaging Systems
Verification and Validation**

Submitted by **Karen J. Laurin**
in partial fulfilment of the requirements for the degree of
Master of Applied Science

Dr. L.C. Briand, Thesis Co-Supervisor

Dr. Y. Labiche, Thesis Co-Supervisor

Dr. H.M. Schwartz, Chair, Department of Systems and Computer Engineering

Carleton University

2009

Abstract

Image segmentation algorithms identify and delineate objects of interest in an image. Currently, the process to ensure the segmentation is correct is slow and time consuming, as evaluation is done manually. This thesis proposes a testing workflow that can be applied for testing image segmentation software. Pairs of segmentations from different versions of the software are compared using measures defined in literature and this data is used to train classifiers to identify if the pair is consistent. If the classifier produced is acceptable, the evaluation process can be automated. Otherwise, another version of the software must be produced, evaluated manually and added to the training data in order to achieve an acceptable classifier. The case study demonstrates the application of the workflow using software that segments tumors from 3D MRI scans. The workflow is then extended further to show a cost effective way of accelerating the learning process by producing artificially created segmentations in order to create a large training set for machine learning without requiring multiple versions of the software. This thesis also presents a framework for automating the testing workflow process.

Acknowledgements

I would like to thank my supervisors, Dr. Yvan Labiche and Dr. Lionel Briand for their guidance and support throughout the research and writing of this thesis. I would also like to thank Leo Grady, Marie-Pierre Jolly and Rajesh Subramanyan from Siemens Corporate Research software and medical imaging departments. Their help in the technical aspects of this research was greatly appreciated. Finally I would like to thank my family and friends for their support.

Table of Contents

1	Introduction.....	1
1.1	Overview	1
1.1	Scope of Work and Contributions.....	5
1.2	Research Questions	6
2	Related Work	7
2.1	Non – testable Programs and the Oracle Problem.....	7
2.2	Left Ventricle of the Heart Case Study	10
3	Background.....	11
3.1	Image Segmentation Comparison Techniques.....	11
3.1.1	Overlap Measures	13
3.1.2	Volume Difference Measures	14
3.1.3	Geometrical Measures	14
3.2	Machine Learning	15
3.2.1	JRIP.....	18
3.2.2	PART	19
3.2.3	J48.....	19
3.2.4	Wrapper Attribute Selection	21
3.2.5	Classifier Attribute Selection.....	21
3.2.6	CFS Filter Attribute Selection	21
3.2.7	Performance Metrics.....	22
4	Description of the Testing Workflow for Medical Imaging Software.....	24
4.1	Segmentation Evaluations Swimlane.....	26
4.2	Learning Classifier Swimlane	27
5	Case Study Design and Description.....	30
5.1	Software Under Test.....	30
5.2	Tools.....	31
5.3	OncoCare Algorithms	31
5.4	Extended Functionality of OncoCare Version 3	32
5.5	Segmentation Evaluation.....	34

5.6	Comparison Measures	35
5.7	Attribute Selection.....	35
5.8	Attribute Sets.....	36
5.9	Description of Classifiers	37
5.10	Experiment Set Up and Description.....	38
5.10.1	Experiment 1	39
5.10.2	Experiment 2.....	40
5.10.3	Experiment 3.....	41
5.10.4	Experiment 4.....	43
5.11	Poorly Performing Rules or Leaves Description.....	45
6	Results of Case Study	47
6.1	Results of Experiment 1	47
6.2	Results of Experiment 2	50
6.3	Results of Experiment 3	53
6.4	Results of Experiment 4	58
6.5	Results of Identifying Poorly Performing Rules and Leaves	64
6.5.1	JRIP.....	65
6.5.2	PART	66
6.5.3	J48.....	67
6.5.4	Conclusion for Identifying Poorly Performing Rules or Leaves	69
6.6	Answering the Research Questions.....	70
6.7	Improving the Certainty of the Results	73
6.8	Comparison of Results Obtain from a Similar Case Study	74
7	Testing Framework	80
7.1	Testing Framework Structure and Activity Diagram.....	80
7.1.1	Preparing the Data.....	84
7.1.2	Algorithm Pairs.....	86
7.1.3	Evaluating Pairs of Test Case Results (<i>TCR</i>)	87
7.1.4	Comparing Test Case Results	89
7.1.5	Learning or Classifying.....	91

7.2	Application of the Testing Framework to Medical Imaging Systems	93
8	Conclusion and Future Work	98
	References	101
	Appendix A Statistical Data on Comparison Measures.....	103
	Appendix B Statistical Data on Classifiers.....	107
	Appendix C ID's of Test Case Result.....	116

Table of Tables

Table 3-1 Comparison Measures	12
Table 4-1 How to determine evaluation of $S_{n,i}$ using evaluation of $S_{n,j}$ and the consistency class as determined by the learnt classifier	27
Table 4-2 Consistency of segmentation pair ($S_{n,i}$, $S_{n,j}$) – Categorical Input for Machine Learning	28
Table 5-1 Algorithms that were used in the case study experiments.....	32
Table 5-2 Machine learning classes – Categorical Input.....	35
Table 5-3 Attribute sets.....	37
Table 5-4 Common characteristics of the set up of all experiments.....	39
Table 5-5 Details of experiment 1	40
Table 5-6 Details of experiment 2	41
Table 5-7 Details of experiment 3	42
Table 5-8 Details of experiment 4.1	44
Table 5-9 Details of Experiment 4.2.....	44
Table 5-10 Details of experiment 4.3	44
Table 6-1 Average accuracy, kappa statistic and AUC values of the trained classifiers for different training set sizes	60
Table 6-2 The attribute sets and classification techniques of the trained classifiers chosen for investigating poorly performing rules and leaves	65
Table 6-3 Results of manually evaluating poorly performing rules for the JRIP classifier	66

Table 6-4 Results of manually evaluating poorly performing rules for the PART classifier	67
Table 6-5 Results of manually evaluating poorly performing leaves for the J48 classifier	69
Table 6-6 Comparison of Frounchi case study to experiment 3 of this case study	75

Table of Figures

Figure 1-1 Manual image segmentation algorithm evaluation process [11].....	3
Figure 3-1 Entropy equation [19]	20
Figure 4-1 Two-swimlane activity diagram of the workflow for testing medical imaging software as proposed by Frounchi in [11].....	25
Figure 6-1 The resulting graphs for experiment 1	48
Figure 6-2 The resulting graphs for experiment 2	51
Figure 6-3 The resulting graphs for experiment 3	54
Figure 6-4 Experiment 3 – The resulting tree of using the J48 Classifier and the J48 Wrapper attribute selection technique as selected from all attributes.....	57
Figure 6-5 Experiment 3 – The resulting tree of using the J48 Classifier and all attributes	57
Figure 6-6 The resulting accuracies and kappa statistic values for experiment 4	59
Figure 6-7 The resulting AUC values for experiment 4	60
Figure 6-8 Experiment 4 (200) – J48 Classifier using all attributes	63
Figure 6-9 Experiment 4 (300) – J48 Classifier using all attributes	63
Figure 6-10 Experiment 4 (400) – J48 Classifier using all attributes	63
Figure 6-11 Tree produced by J48 classification technique using attributes selected by the J48 Wrapper attribute selection technique.....	68
Figure 6-12 Accuracy of the trained classifiers from the Frounchi case study	76
Figure 6-13 Kappa statistic values of the trained classifiers from the Frounchi case study	77

Figure 6-14 Area under the ROC curve (AUC) values of the trained classifiers from the Frounchi case study.....	77
Figure 7-1 Testing framework class diagram	82
Figure 7-2 Activity diagram of the <i>templateLP()</i> method in <i>LearningProcess</i>	83
Figure 7-3 Activity diagram for <i>evaluate()</i> method in <i>LearningProcess</i>	87
Figure 7-4 Activity Diagram of <i>evaluate()</i> method in <i>EvaluationMethod</i>	89
Figure 7-5 Activity diagram for <i>calculateComparisonMeasures()</i> method	90
Figure 7-6 Activity diagram for the <i>trainClassifier()</i> method	91
Figure 7-7 The testing framework class diagram expanded to show the application of the framework to a medical imaging system	94

1 Introduction

1.1 Overview

Image segmentation is the process in which objects of interest are identified and delineated from an image [14],[16],[20]. Each individual object of interest is defined as a homogeneous region with respect to a specific characteristic, (i.e. texture or intensity) that does not overlap any other region [15]. There are two processes associated with image segmentation. The first is the high level recognition of the object of interest in the image. The second process is the low level delineation, where the exact spatial extent and composition of each point in the object of interest must be determined [16]. In a 3D image the measurement used to define a location on the 3D grid of the image is known as a voxel [14]. As computers are weak in recognition, the computer is often aided by human assistance to recognize the objects on interest in the image. There has been many image segmentation algorithms developed over the decades, but the image segmentation still remains a challenging problem [16].

A more specific problem is the use of image segmentation algorithms in the medical imaging field. Health professionals use diagnostic imaging, such as 3D Magnetic Resonance Imaging (MRI) scans, to identify and gather information for different types of medical imaging problems. The information gathered from the scans is then used to diagnosis the patient and plan the treatment required [14]. The medical imaging software that is used in the case study of this thesis was created by Siemens Corporate Research.

The software aids in the identification of tumors in a 3D MRI scan. As computers are weak in recognition, the software uses the diameter of the tumor, as drawn by a medical professional, as the starting point for the segmentation. Building software that segments tumors is a very difficult process as the sizes and shapes of tumors vary throughout the body. Tumors may also be heterogeneous in the image or may have calcifications or necrotic regions. The scans also contain regions that make it more difficult to identify the object of interest, such as blood vessels that show up as high contrast structures in a MRI scan [13].

While building the medical imaging software is difficult, another problem is the testing of the medical imaging software. Currently, testing of the medical imaging software is done manually by medical professionals, who evaluate whether the segmentations produced are acceptable. Figure 1-1 shows the process for which medical imaging segmentation algorithms must go through to be evaluated. Activity A involves the creation and revision of the software algorithm. The objects of interest are then segmented from the image in Activity B using the test suite, which is comprised of a set of test images. The medical professional must then manually evaluate the segmentations in Activity C and decide if the segmentations are acceptable. If the segmentations are not acceptable, the process must be repeated until a version is produced that meets the medical professional's standards.

Figure 1-1 Manual image segmentation algorithm evaluation process [11]

Having medical professionals manually evaluate each test case for a new version of the software is a very expensive and time consuming task. The testing is also subject to human error and bias, as the evaluation of whether the segmentation is acceptable or not is only the opinion of a medical professional. By providing a way to semi automate the software testing, the developer of the software will save time and money, as well as remove some of the human error and bias that is present with manual testing.

In order to test image segmentation algorithms there needs to be a way to evaluate the image segmentations that are produced by the algorithm. Image segmentation evaluation has lagged behind the development of image segmentation algorithms, especially in the field of medical imaging software. Reasons for this include the fact there are no widely available data sets to be used for evaluating the segmentation software, as different segmentation problems require different approaches [20]. Also it is difficult to define performance metrics and statistics to evaluate the image segmentations. A third reason is that the ground truth (GT) or true segmentation (TS) is not always available in the

medical imaging field. The GT or TS is the segmentation that is considered to be the correct segmentation for the particular object of interest [16]. In the medical field the GT is not available and the opinion of a medical professional is used as the basis for the GT. For this case study the GT or TS was not available to be used for comparison with the segmentation, as is the case with many medical imaging problems. Therefore, to test the software, the first few versions must be evaluated manually. Once a working version of the software was obtained, the segmentations from that version of the software could be considered the set of true segmentations (TS) to which all future versions could be compared. The set of segmentations from the software being tested is known as the Segmentations Under Test (SUT). The image segmentation evaluation uses metrics, that are defined in literature, that decide if two image segmentations from two different versions of the software are consistent with one another for diagnostic purposes. In other words, would the two image segmentations come to the same diagnostic result? If the two sets of segmentations, TS and SUT, can be considered to be consistent, then the software algorithms can also be considered to be consistent. This is the basis for the workflow used in the case study of this thesis.

The remainder of the thesis is laid out as follows. Section 1.1 discusses the scope of work of this thesis and the contributions. Section 1.2 lists the research questions that the case study performed for this thesis aims to answer. The related work is in Section 2, which includes a case study that was performed using the same testing workflow. Background information on image segmentation comparison measures and machine learning is presented in Section 3. Section 4 describes the testing workflow that was

followed for the case study. Section 5 is the description of the case study performed for this thesis and the results for the case study follow in Section 6. Using the results from the case study performed for this thesis and a previous case study, a generalized testing framework for a tool to semi-automate the testing of medical imaging software is presented in Section 7. The conclusion of the thesis and the future work is presented in Section 8.

1.1 Scope of Work and Contributions

Through the case study that is presented in Sections 5 and 6, the research for this thesis aims to accomplish the following.

This thesis aims to show the testing workflow that was proposed in [11] and discussed in Section 4 can be adapted to test any medical imaging software. By applying the proposed testing workflow to a different medical image segmentation problem than was done in [11], this is the next step towards being able to generalize the application of the workflow towards developing a generalized testing framework. The case study set up and results can be seen in Sections 5 and 6 and the testing framework can be seen in Section 7.

The thesis also aims to show which attribute sets, classification techniques, and attribute selection techniques create the best classifiers and whether the selections from this case study are the same as what was found in [11]. A comparison of the results from both case studies can be found in Section 6.7.

The third thing this thesis aims to show is that the gathering of data for machine learning can be accelerated by artificially creating image segmentations in one version of the software, instead of waiting for multiple versions of the software to be available. This allows for a large training set to be created and used for machine learning without having to manually evaluate multiple versions of the software, which is very time consuming. This is discussed in more detail in Section 5.10.4.

The last thing this thesis aims to show is that by manually evaluating a subset of the image segmentations that are being tested, the classifier accuracy can be improved. The discussion of the tradeoff between classifier accuracy and amount of manual work to be done can be seen in Section 6.5.

1.2 Research Questions

The following questions are the research questions this thesis aims to answer through the case study. Section 6.6 discusses the answers to the research questions posed here.

1. Which measures (overlap and volume differences measures, geometrical measures or all measures) perform best for machine learning?
2. Which classifier technique performs best?
3. Does attribute selection result in better classifiers?
4. Does the use of artificially created segmentations accelerate learning?
5. By investigating ways of improving poorly performing rules or nodes, can an acceptable tradeoff between classifier accuracy and manual segmentation evaluation be found?

2 Related Work

As this thesis is not a study on how to do image segmentation or how to improve image segmentation techniques, this area of work will not be discussed here. However, the testing of medical imaging software is in the field of testing non-testable programs, which is defined and discussed in Section 2.1. A similar case study to the one performed for this thesis is discussed in Section 2.2.

2.1 Non – testable Programs and the Oracle Problem

When testing software programs a mechanism, known as an oracle, is used to make the decision as to whether the output of the program is correct. An oracle must be able to make the decision on the correctness of the output of the program in reasonable amount of time, with a reasonable amount of effort [7]. A non-testable program is a program for which there is no oracle or the output of the program is not easily determined [18]. Non testable programs fall into three distinct classes:

1. the programs that were written to determine the answer to a specific problem, as the answer is not already known,
2. the programs that have too much output to evaluate all of it in a reasonable amount of time, and
3. the programs for which there is a misconception made by the tester. In this third case the output may be right, but the specification used for testing may not be the specification from the original problem.

In [7] and [18] there are suggestions on how to test software that is deemed to be non-testable. Both papers suggest the use of a pseudo-oracle. A pseudo-oracle is a program that is produced in parallel to the original software program, but is produced by an independent team or programmer. The pseudo-oracle is produced to fulfill the same requirements as the original program and is used in testing to ensure the correctness of the original program by comparing the output of the two programs. If the two independently developed programs give the same output for the same input, the original program is deemed as giving the correct output. As producing two programs for the same requirements is time consuming, it is suggested to use a high level language for the pseudo-oracle to ensure it is developed quickly [7,[18].

As not all problems can be quickly developed using a high level language, another suggestion for testing non-testable programs is to only test the simple cases, that there are known answers for. While this suggestion does allow for testing of the software program, it does not allow for thorough testing. It is the complicated cases that will not be tested that tend to be the error prone sections of the software [18].

A third suggestion given in literature is to accept plausible results. Since the exact answer is not known, using a range of plausible results with a likelihood measure assigned to the values in the range, the program can at least be determined to be running as one would expect [18].

Medical imaging software programs fall into the category of non-testable programs. The ground truth, which would be considered the correct output, of a segmented image is not easily determined. Currently medical imaging software is tested by having medical

professionals evaluate all of the image segmentations in the test set. This process is very time consuming and expensive, as all of the testing is done manually. Also, the medical professional is giving their opinion as to what the ground truth would be for the segmented images. The process of manual evaluation is subject

to human bias and error. Therefore this research aims to reduce the time needed to test medical imaging software and reduce the bias and error of manual segmentation evaluation.

The use of machine learning techniques to evaluate image segmentations has been done in [21] and [11]. The work done in [21] uses machine learning to identify the best segmentation, given two segmentations for the same image. The two segmentations are produced by two different algorithms. The research uses a set of segmentation evaluation techniques, each a stand-alone method, as input to the learning component. A stand-alone method evaluates the goodness of a segmentation using features of the segmentation and the original image such as, color or texture. A decision tree is built for each of the evaluation techniques, and a weighted vote from all of the evaluation techniques decides the goodness of the segmentation. The research done in [21] differs from this current study, as the segmentations are evaluated individually and information on the actual image is required for the technique proposed. The current study uses measures to compare the two image segmentations and uses machine learning to learn if two image segmentations are consistent from a medical diagnostic standpoint. The work done in this current study is very similar to the work done in [11]. Further information on the case study from [11] can be found in Section 2.2.

2.2 Left Ventricle of the Heart Case Study

A case study similar to the one performed for this thesis has been completed using software that segments the left ventricle of the heart by Frounchi [11]. The proposed testing workflow from the Frounchi case study was the same workflow that was used in the current case study. Details of the proposed testing workflow can be seen in Section 4. Additionally, the same image segmentation measures and machine learning techniques are used both case studies in order to compare the results. A discussion of the measures and techniques used in these case studies can be seen in Section 3. For more detailed definitions of the measures and techniques refer to [11]. The comparison of the results from the two case studies is discussed in Section 6.7.

The purpose of using the same proposed testing workflow as the Frounchi case study was to see if the proposed testing workflow can be applied to a broad range of medical imaging software. Section 7 discusses a proposed framework for a general medical imaging software testing tool that was developed using the results from the two case studies.

The selection of the comparison techniques and machine learning techniques we used in this thesis was all done with the idea that our work would be compared to a previous case study [11]. While we could have explored other techniques, our purpose was to repeat the work done in [11] with a different medical imaging system to start to see if there was a consistent trend in results using two different medical imaging software systems.

3 Background

There are two areas of interest for which the terminology and background information will be covered for this thesis. The first area of interest is the measures used as comparison techniques to compare two segmentations. The measures will be presented in Section 3.1.1. For an overview of what image segmentation is refer to Section 1.1. The second area of interest is a brief background on machine learning that is presented in Section 3.2.

3.1 Image Segmentation Comparison Techniques

For the purpose of the case study done for this thesis, a set of measures that compared two segmentations was required. The following sections describe the overlap, volume and geometrical measures that were used in case study for this thesis. As the proposed testing workflow in this thesis is the same as defined in [11], the measures used to compare two image segmentations will also be the same. Therefore refer to [11] for a formal description and the mathematical definitions of the measures used.

When comparing two sets of segmentations, the segmentations that are from the software that is under test will be known as the segmentations under test (SUT). The set of SUT are compared to the set of segmentations known as the ground truth or true segmentations (TS).

Table 3-1 identifies the comparison measures used in this thesis and the short hand notation used for each of the measures in the rest of this thesis. These comparison measures are the non-categorical data used as input for machine learning.

Comparison Measure	Type	Description
FPVF	Overlap	False Positive Volume Fraction
TPVF	Overlap	True Positive Volume Fraction
TC_MF	Overlap	Tanimoto Coefficient - Multi label fuzzy segmentation
DSC_MF	Overlap	Dice Similarity Coefficient - Multi label fuzzy segmentation
AVD	Volume Difference	Absolute Value of Volume Difference
ANVD	Volume Difference	Absolute Value of Normalized Volume Difference
AvgBoundDist	Geometrical	Average Distance to Boundary Difference
HausDist	Geometrical	Hausdorff's Distance
BaddeleyDist	Geometrical	Baddeley's Distance
PrattFOM	Geometrical	Pratt's Figure of Merit
Peli_Malah_Mean	Geometrical	Peli and Malah Mean Error
Peli_Malah_Var	Geometrical	Peli and Malah Variance
RMSDist	Geometrical	Root Mean Square Distance
SODI	Geometrical	Odet's degree of over segmentation (scaling)
ODI	Geometrical	Odet's over segmentation divergence (non-scalable)
SUDI	Geometrical	Odet's degree of under segmentation (scaling)
UDI	Geometrical	Odet's under segmentation divergence (non-scalable)
PrincipalAxisDist	Geometrical	Principal Axis Distance

Table 3-1 Comparison Measures

Currently these measures are not used in the testing of the medical imaging system used in this case study. However the 18 measures presented have been determined to be

potentially relevant to the medical imaging problem by the engineer developing the algorithm at Siemens. Therefore the purpose of using so many measures is to see if it can be determined which measures will result in the same evaluation of the segmentation, when compared to the medical professional manually evaluating the segmentation.

The reasoning behind grouping the measures by type is to determine if the simple measures, overlap and volume, will result in classifiers of the same or better performance than those trained using geometrical or all measures. Simple measures are much simpler and less time consuming to calculate, which will result in a testing framework that is much quicker to execute.

The overlap measures will be described in Section 3.1.1. Section 3.1.2 describes the volume difference measures and Section 3.1.3 describes the geometrical measures.

3.1.1 Overlap Measures

The overlap measures in this thesis are the measures that are concerned with the number of voxels within the segmentation and the number of voxel the two segmentations being compared have in common. In this thesis four overlap measures are used, true positive volume fraction (TPVF), false positive volume fraction (FPVF), tanimoto coefficient (TC), and dice similarity coefficient (DSC).

TPVF and FPVF are statistical decision theory measures, which are used to determine the frequency of individual voxels being correctly or incorrectly classified when comparing the segmentation under test to the true segmentation [1].

TC and DSC are commonly used for measuring the overlap of two regions that have been selected in an image. The TC is a ratio of the number of voxels in the intersection of the two overlapped regions to the number of voxels in the union of the two overlapped regions. The DSC is a ratio of the number of voxels in the intersection of the two overlapped regions to the number of voxels in the mean label volume [6].

3.1.2 Volume Difference Measures

To measure the volume difference between two segmentations there are two calculations used in this thesis, the absolute value of volume difference (AVD) and the absolute value of the normalized volume difference (ANVD). The AVD is simply the absolute value of the difference between the volumes of the two segmentations being compared. The ANVD is the AVD divided by the volume of the true segmentation [8].

3.1.3 Geometrical Measures

Geometrical measures are considered to be more complex than overlap and volume measures. Geometrical measures take into account the defined boundaries, sizes and axis of the segmentations being compared. By evaluating an image segmentation algorithm, the tester and developer want to learn if the algorithm is over or under segmenting the object of interest. Over-segmentation is when algorithm is selecting voxels as being a part of the image segmentation that should not be. Under-segmentation is when the algorithm is not identifying all voxels that are a part of the object of interest. Simple measures, such as volume and overlap, may account for if an algorithm is over segmenting or even if there is severe under-segmentation problems. However, there is a

need for better measurements to identify slight over and under segmentations, which may cause for a segmentation to not be acceptable for diagnostic purposes. The geometrical measures used in this thesis are identified in Table 3-1.

3.2 Machine Learning

Machine learning is the act of using a computer to learn patterns that may be present in a set of data. Using these patterns the data can be classified into similar classes [19]. In this thesis, machine learning is used to learn a classifier that can predict the consistency between two image segmentations, given a set of comparison measures.

The machine learning algorithm must find a decision boundary that splits the data into the number of classes that the algorithm is trying to learn. The problem then becomes of how complex the decision boundary should be. A complex decision boundary can lead to over-fitting of the training data and poor results when the classifier is used in practice. A simple decision boundary might mean the classifier performs poorly on the training data, but performs well on future data [10]. To avoid over-fitting the classifier to the training data it is suggested to start with the simplest methods, such as simplest-first search, and then proceed to more complex methods. As there is no general method that produces the best classifier, multiple methods must be investigated to identify the method that fits each specific problem [19].

Machine learning algorithms can be divided into four learning styles, classification learning, association learning, clustering and numeric prediction. Classification learning is when the algorithm is given a set of examples to learn the patterns from in order to

learn to classify unseen data. Classification learning is the area of interest in this thesis. Association learning is when the algorithm is searching for any association among the features, not just the features that predict the class of the data. Clustering uses the examples provided to the algorithm to form groups that are similar. Finally, numeric prediction is when the algorithm must predict a numeric outcome from the data provided, not a class [19].

Classification learning can be further split into categories, depending on the type of method used to train the classifier. In this thesis the focus is on rule induction techniques and decision trees, as the output for these classifier algorithms is easy to read and interpret. Other methods, such as probabilistic model, linear models and instance based learning do not have output that easily shows how the classifier arrives at a specific class for a specific training instance. We are interested in being able to understand the reasoning behind the classifier's decision in order to find ways to improve the performance of the classifier by using partial manual evaluation.

In particular, this thesis uses the rule induction based techniques, JRIP and PART, which are presented in Sections 3.2.1 and 3.2.2 respectively. The decision tree algorithm used is J48, which is presented in Section 3.2.3.

In order to ensure the learnt classifier is not over-fitting the data, it is best to verify the learnt classifier. This is done by having separate training and test sets. The classifier is first trained using the training set and the error rate of the classifier is verified on the test set. This ensures more realistic error rates, then just using all of the data available for training. In order to split the data available between the training and test set, stratification

is used. Stratification randomly splits the available data, while ensuring that the classes to be learned are well represented in both the training and test set. However, stratification alone does not eliminate uneven representation. Therefore cross-validation must also be used. Cross-validation is when the data is split into a fixed number of folds, and the training procedure is repeated for each fold. Each fold has a different subset of training data that is held back as the testing set. The standard method used is stratified tenfold cross-validation, in which the data is split into ten folds and the error rate is averaged over the ten error estimates to give the overall error [19]. This method will be used in this thesis.

Another method used to aid the classification techniques is attribute selection. Attribute selection limits the number of features used for learning, as it finds a set of features that are not highly correlated to one another. This is important, as it selects a set of features that removes unsuitable attributes, so that when training the classifier, the data set is optimized. There are two main approaches to selecting attributes, the filter method and the wrapper method. The filter method uses the general characteristics of the data to make an independent assessment as to what attributes are best to use for learning. The wrapper method uses a machine learning algorithm to evaluate a subset of the attributes in order to decide if the subset is better than the entire set for learning [19]. The wrapper attribute selection techniques used in this thesis are the Wrapper and Classifier, which are presented in Sections 3.2.4 and 3.2.5. The filter attribute selection technique used was the Correlation-based Feature Selection (CFS), which is presented in Section 3.2.6. The CFS filter and wrapper attribute selection techniques were chosen because of the results

of the benchmarking performed in [12]. The classifier attribute selection technique was chosen as it is the other wrapper method implemented in WEKA.

In order to decide if a learnt classifier is acceptable, performance metrics must be defined to evaluate the learnt classifier. The methods used to evaluate the classifiers are presented in Section 3.2.7.

3.2.1 JRIP

In the WEKA tool, the JRIP classification technique is an implementation of a propositional rule learner that was proposed by William W. Cohen. The JRIP classification technique implements the Repeated Incremental Pruning to Produce Error Reduction (RIPPER). Rules are built for each class starting with the smallest class. Incremental reduced error pruning is used to generate the initial set of rules for each class. This means once a rule is created using the growing set; it is immediately pruned using the pruning set. The growing set and pruning set are made up of the training data available. If the created rule performs better than the previous rule it is kept. The RIPPER algorithm uses an extra stopping condition, known as the description length (DL) that takes into account the number of bits needed to execute the rule. Once the set of rules is created for each class, reduced-error pruning is performed again to remove the redundant rules between classes, choosing the rule that has a better description length [19]. Further details on this technique can be found in [5].

3.2.2 PART

The PART classification technique is also a rule induction technique, like the JRIP classification algorithm. However, PART uses the C4.5 algorithm, which creates partial decision trees, and turns each partial decision tree created into a decision list. This method avoids global optimization, by removing the training instances for which a rule is created, before moving on to build the next rule. For each rule created, the leaf with the largest coverage is selected from the pruned decision tree built from the remaining training data. This method is quick and flexible, as it does not build a fully expanded decision tree for each rule it creates. Instead the method seeks a sub-tree that can be simplified no further. Once this sub-tree is found, the rule can be created. The sub-tree that is selected to be expanded is the node with the lowest entropy, as it is more likely to produce a small sub-tree and a more generalized rule. Entropy is used to describe the uncertainty of a measure [19]. Details on the C4.5 algorithm can be found in the next section.

3.2.3 J48

J48 is a decision tree classification technique that implements the C4.5 algorithm. The C4.5 algorithm uses the divide and conquer approach to create a decision tree to be used for classification. At a high level this approach selects an attribute to divide the data and then creates branches for each possible value. These branches divide the data into subsets for which the approach can be repeated for. Once a point is reached where all the data at a node is of the same classification, the process can stop as that data has been

classified. The nodes which contain data of the same classification are known as leaves of the decision tree [19].

Deciding on which attribute to start dividing the data on is the difficult part of the algorithm. The attribute selected should have high purity, meaning it splits the data in such a way that leaves are obtained quickly. The measure used is called *entropy* and its equation is shown below in Figure 3-1.

$$\text{entropy}(p_1, p_2, \dots, p_n) = -p_1 \log p_1 - p_2 \log p_2 - \dots - p_n \log p_n$$

Figure 3-1 Entropy equation [19]

The p attributes are expressed as fractions that add up to 1. The fractions represent the number of instances at the node to the number of instances total [19].

The one drawback to entropy is that it tends to select attributes that have a large number of branches or possible values, in order to divide the data up as quickly as possible. This may not always be beneficial, so another measure is used to compensate. This measure is known as the *gain ratio*. The gain ratio looks at the number of nodes that were created by selecting to split on a certain attribute and the size of each of these nodes. This value stops the algorithm from choosing an attribute that splits the data into too many subsets near the beginning of the tree [19].

There are two ways to prune a decision tree. If pruning is done once the tree is completely built, it is called *postpruning*. *Prepruning* is when the algorithm makes the decision whether or not to develop sub-trees as the decision tree is built. *Pruning* is the act of simplifying the decision tree, as to not over fit the training data [19].

3.2.4 Wrapper Attribute Selection

For the wrapper attribute selection technique the J48 classifier was chosen in this case study. The wrapper attribute selection technique uses a learning scheme to find attributes that are highly correlated. In order to estimate the accuracy on a set of attributes, cross validation is used. The best first search method was used, as it performs a greedy hill climb with backtracking through the set of attributes without being very time consuming. This search method does not stop when performance drops, so it is better than a simple forward search that cannot backtrack to a better subset [19].

3.2.5 Classifier Attribute Selection

The classifier attribute selection technique uses a classifier to estimate the merit of the given attributes. For this case study the J48 classifier was chosen. This attribute selection technique evaluates subsets of the attributes on the training data to select a set of attributes that are not highly inter-correlated. The exhaustive search method was used for this attribute selection technique. This search method searches the entire space of attribute subsets, but is very time consuming, so it was only used for one of the three attribute selection techniques [19].

3.2.6 CFS Filter Attribute Selection

The Correlation-based Feature Selection (CFS) is a filter attribute selection technique. This attribute selection technique tries to find a subset of attributes that have low inter-correlation, but are highly correlated with the class. When choosing the best attributes for learning this method takes into account the individual predictive ability of each

attribute and the redundancy among all of the attributes. Filter methods are concerned with the individual characteristics of an attribute, and are not concerned with how these methods perform in a subset [19]. The search method used for this attribute selection technique was the best first search algorithm, which was also used for the wrapper attribute selection technique.

3.2.7 Performance Metrics

Performance metrics are needed in order to evaluate the learnt classifier. In this thesis three performance metrics are used, the accuracy, the kappa statistic value and the area under the Receiver Operating Characteristic (ROC) curve. These values were all calculated in the software used for machine learning, WEKA.

Accuracy is the percentage of correctly classified instances. The closer the result is to 100%, the better the accuracy of the learnt classifier.

The kappa statistic tries to remove the chance of correct classifications, as it can be expected that there will be some correct classifications that occur by chance and not from the process of learning. How the kappa statistic is derived and calculated is discussed in [4] and [9]. Kappa statistic values close or equal to 1 are considered to show the classifier is classifying the classes accurately with little or no predictions made by chance. A 0 kappa statistic shows the classifier is tending to classify all attributes by chance. Values that are negative show the probability an instance is being classified by chance is larger than the probability the classifier has learned to classify an instance correctly.

The ROC curve plots the true positive rate of the classifier versus the false positive rate. The area under this plotted curve indicates the performance of the classifier. Information on how to obtain the area under the curve (AUC) is in [3]. The larger the AUC, the better the classifier performs, as the classifier produces more true positive results than false positive results [2].

4 Description of the Testing Workflow for Medical Imaging Software

The proposed workflow for testing medical imaging software is adopted from similar research on testing medical imaging software that segments the left ventricle of the heart [11]. The workflow is mostly performed manually, with the exceptions of the use of Matlab for calculating the overlap, volume and generalized measurements and WEKA is used to perform machine learning. Figure 4-1 shows the workflow for testing medical imaging software.

Figure 4-1 Two-swimlane activity diagram of the workflow for testing medical imaging software as proposed by Frounchi in [11]

There are two series of activities in the workflow that can take place concurrently. The segmentation evaluations series of activities, shown on the left side of the figure, involves the creation and revision of the software algorithm using manual or automated evaluation. The right side of the activity diagram is the learning classifier activities, which involves the creation of a classifier to automate the evaluation of the software algorithm versions as they are produced. Manual evaluation of the image segmentations takes place on the left side of the activity diagram until a classifier that is considered to be accurate enough is produced from the workflow on the right side.

4.1 Segmentation Evaluations Swimlane

The segmentation evaluations swimlane starts with devising the first version of the software algorithm in Activity A. If the segmentations produced by the current version of the software are not adequate, the workflow returns to Activity A and a new version of the software is produced.

Activity B uses the newest version of the segmentation algorithm (version i) that was produced in Activity A to create the test suite. The test suite is the set of segmentations under test (SUT), that will be evaluated to determine if version i is producing segmentations that are adequate. The segmentations produced by version i are known as set_i .

The next step is the evaluation of the segmentations in set_i , either manually in Activity C or automatically using a learnt classifier in Activity D. If an acceptable learnt classifier is not yet available the segmentations are evaluated manually by a medical professional in Activity C. If there is an accurate learnt classifier Activity D uses the similarity measures that are calculated in Activity E in the Learning Classifier Swimlane to determine if the segmentations in set_i are consistent with segmentations produced by a previous version of the software, version j , where $j < i$. The notation used to identify a single segmentation is $S_{n,i}$, where n is the segmentation number and i is the revision number of the software algorithm.

By pairing the segmentation from version i with a segmentation in a previous version, version j , that are both from the same image, the classifier uses the similarity measures to determine if $S_{n,i}$ and $S_{n,j}$ are consistent. Using the consistency and the evaluation of $S_{n,j}$,

the evaluation of $S_{n,i}$ is determined as shown in Table 4-1. If $S_{n,i}$ and $S_{n,j}$ are predicted to be consistent, then the evaluation of $S_{n,i}$ is the same as the evaluation of $S_{n,j}$. If the pair is inconsistent and $S_{n,j}$ evaluated as correct, then $S_{n,i}$ is incorrect. Finally if the pair is inconsistent and the $S_{n,j}$ evaluated as incorrect, then $S_{n,i}$ must be evaluated manually by a medical professional as no conclusion can be drawn about the correctness of $S_{n,i}$.

Once the evaluation of set_i is complete and a majority of the segmentations are considered to be acceptable, the testing process is considered to be finished and the segmentation algorithm is deemed to be correct. Otherwise, the segmentation algorithm must be revised in Activity A and the workflow must be repeated for the new version of the segmentation algorithm.

Evaluation of $S_{n,j}$	Predicted consistency of the pair ($S_{n,i}$, $S_{n,j}$)	Evaluation of $S_{n,i}$
Correct	Consistent	Correct
Correct	Inconsistent	Incorrect
Incorrect	Consistent	Incorrect
Incorrect	Inconsistent	Requires manual evaluation to determine correctness

Table 4-1 How to determine evaluation of $S_{n,i}$ using evaluation of $S_{n,j}$ and the consistency class as determined by the learnt classifier

4.2 Learning Classifier Swimlane

The right hand side of Figure 4-1 is the activities involved in learning a classifier to use for automating the evaluation of the software algorithm. The first activity, Activity E, is where pairs of segmentations from different algorithm versions are compared using the similarity measures that were introduced in Section 3.1. At least two versions of the

algorithm must be produced and evaluated manually, in order to pair the segmentations from two versions.

Given two versions of the software algorithm, version i and version j , there are two sets of segmentations available for testing, set_i and set_j . For each segmentation, n , the pair $(S_{n,i}, S_{n,j})$ is considered to be consistent, inconsistent or unusable, based on the manual evaluations of a medical professional. The segmentation pairs that are considered to be useable in the machine learning training set are those of the consistent or inconsistent class. Table 4-2 shows how to determine the class from the manual evaluations of the segmentation pairs.

Manual Evaluation of $S_{n,i}$	Manual Evaluation of $S_{n,j}$	Class
Correct	Correct	Consistent
Correct	Incorrect	Inconsistent
Incorrect	Correct	Inconsistent
Incorrect	Incorrect	Unusable (not a class)

Table 4-2 Consistency of segmentation pair $(S_{n,i}, S_{n,j})$ – Categorical Input for Machine Learning

Activity F performs the machine learning with the similarity measures as the data for each segmentation pair and the class to be learned as determined in Table 4-2. The segmentation pairs that are considered to be unusable are not used in Activity F, as two segmentations that are considered to be unacceptable does not help the classifier to learn what makes two image segmentations diagnostically similar. When two segmentations lead to the same diagnosis by a medical professional they are considered to be diagnostically similar.

Once a learnt classifier is produced, Activity F determines if the classifier produced is valid by using performance measures as discussed in Section 3.2.7. If the classifier is determined to have a high accuracy, a kappa statistic that is close to 1 and the area under the ROC curve is close to 1, then the classifier is valid and can be used in Activity D, the automated evaluation of the segmentation algorithm. If the classifier is not determined to be valid, the process is repeated again, using a larger training set for the classifier, as the next version of the algorithm will be manually evaluated and the data can be added to the training set to improve the classifier produced.

5 Case Study Design and Description

The following sections outline the set up for the case study done for this thesis. A description of the software that was tested in this case study can be found in Section 5.1 and descriptions of the other tools used in the case study can be found in Section 5.2. Section 5.3 introduces the four versions of the OncoCare algorithm that were used in this case study and Section 5.4 describes the extended functionality of OncoCare Version 3. Section 5.5 describes the scale that was used to manually evaluate the segmentations in the case study. The comparison measures used to compare two segmentations are identified in Section 5.6. The details of the attribute selection techniques used are presented in Section 5.7 and the attribute sets that were created for machine learning are in Section 5.8. Classifier techniques used for machine learning are outlined in Section 5.9. Section 5.10 describes the four experiments in detail, outlining the algorithms used, the training and test set sizes and the objectives of each experiment. The final section, Section 5.11, discusses the background and reasoning behind identifying poorly performing rules or leaves to improve classifier accuracy.

5.1 Software Under Test

The purpose of this case study is to apply the proposed workflow for testing medical imaging software to an existing software algorithm used in the medical imaging field. The software that is the subject of this case study is OncoCare and is produced by Siemens Corporate Research. The software identifies tumors in a 3D MRI scan of a

patient. The algorithm needs an input point to start the segmentation. In this study the starting point for the algorithm was chosen by a medical professional and saved, so that the same segmentation could be reproduced multiple times. The software is able to identify tumors in all parts of the body.

OncoCare saves the information of the segmented image in a RAW file format. This file format identifies if each voxel in the image is a part of the segmentation or not. Using this information the similarity measures between two image segmentations can be calculated.

5.2 Tools

The software tools that were used in this case study include OncoCare, MATLAB and WEKA. OncoCare is the software under test for this case study. MATLAB software is designed for computational intensive problem and was used in this case study to calculate all of the overlap, volume and geometrical measures used to compare two image segmentations.

WEKA is data mining software implemented in Java that was developed at the University of Waikato and stands for Waikato Environment for Knowledge Analysis. In this case study WEKA was used in the machine learning section of the proposed workflow. In particular WEKA was used for attribute selection, classifier training and classifier testing.

5.3 OncoCare Algorithms

Four different versions of the OncoCare algorithm were used for this case study, and are referred to as A1, A2, A3 and A4, respectively. These algorithms were used for the

training and testing of the classifiers in the case study. Table 5-1 shows the algorithms used for each of the experiments.

Experiment	SUT Algorithm	TS Algorithm	Test Set Algorithm
1	A2	A1	A3
2	A3	A2	A3 extended functionality
3	A2, A3	A1, A2	A4
4	A3 extended functionality	A2	A4

Table 5-1 Algorithms that were used in the case study experiments

All of the segmentations from the four versions were manually evaluated, in order to create training sets for machine learning and to be able to calculate the accuracy on the testing set. From the manual evaluations it was determined that the algorithms were always an improvement over the previous version, although sometimes the improvement was small. The extended functionality of A3 is discussed in detail in the next section.

5.4 Extended Functionality of OncoCare Version 3

Experiment 4 of the case study was done to show how artificially created segmentations could be used to accelerate the machine learning process. In the workflow presented in Section 4, if the classifier was not considered to be valid the next version of the algorithm was used to train the next classifier. Therefore, the only way to get a larger training set for training the classifiers was to manually evaluate multiple versions of the software. It is very time consuming to wait for many versions of the software to be ready in order to be able to begin evaluating segmentations automatically. Therefore finding a way to

produce multiple, artificial but realistic segmentations from a single version would make the process quicker.

Another problem found in Experiment 1 and 2 was that there were few segmentation pairs that were from the inconsistent class. Therefore for this case study a method was needed to produce artificially created segmentations that were unacceptable in A3.

To address the size of the training set and the lack of instances in the inconsistent class, A3 was created with extended functionality to produce extra segmentations. In order to create extra segmentations in a single algorithm, the starting point from which the algorithm produced the segmentation needs to be changed. Originally, for each tumor, a medical professional chose two points across the tumor, which represented the diameter of that tumor. These two points were saved, so that the same segmentation could be reproduced multiple times in multiple versions of the software for consistency when testing. The extended functionality of A3 took the two saved input points and randomly displaced them from their original locations, creating a new diameter which the software used as the input to segment the tumor.

The decision to randomly displace the two input points from their saved locations was to simulate a realistic input error by a medical professional. Realistically, the medical professional could be slightly off in selecting the end points of the diameter. Therefore, producing segmentations from a slightly different diameter will still result in realistic segmentations. It was found, through manual evaluation, that this method produced segmentations that were similar to the segmentations produced by the actual working

versions of the software and in some cases the segmentations produced were evaluated to be 4 or 5 on the evaluation scale.

Therefore for a single tumor, multiple segmentations could be produced quickly by selecting the update button in the OncoCare software and saving the corresponding RAW file. This allowed for Experiment 4 to have training set sizes as large as 400 instances by using just two versions, A2 and A3, and there was no need to wait for multiple versions of the software.

5.5 Segmentation Evaluation

To determine the machine learning class for each pair of segmentations in the test set, the segmentations were evaluated manually in the OncoCare software. The evaluation scale was as follows:

1. The segmentation is completely unacceptable and incorrect
2. The segmentation is unacceptable and incorrect, however it does have a slight resemblance to the ground truth
3. The segmentation is considered acceptable, but will need editing before it is usable for diagnostic purposes
4. The segmentation is acceptable and may need a minor edit or two before it is usable for diagnostic purposes
5. The segmentation is acceptable and usable as is for diagnostic purposes

Table 5-2 shows how the classes for machine learning were determined in this case study based on the above evaluation scale.

Class Name	Evaluation of Segmentation Pair ($S_{n,i}$, $S_{n,j}$)
Consistent	Both $S_{n,i}$ and $S_{n,j}$ are ranked 3 or higher
Inconsistent	One of $S_{n,i}$ or $S_{n,j}$ is ranked 3 or higher, while the other is ranked 2 or lower
Unusable (not a class)	Both $S_{n,i}$ and $S_{n,j}$ are ranked 2 or lower

Table 5-2 Machine learning classes – Categorical Input

The unusable segmentations were not used for machine learning, as it cannot be determined that $S_{n,i}$ is correct or not from the fact $S_{n,j}$ is incorrect and the two segmentations are not diagnostically consistent. It may be that $S_{n,i}$ is still incorrect, but in a different way from $S_{n,j}$.

5.6 Comparison Measures

In this case study various overlap, volume and geometrical measures were used to compare two image segmentations from two different versions of the OncoCare software. These comparison measures were then used as the data for machine learning in order to identify if specific comparison measures helped identify if two segmentations were consistent or inconsistent. The comparison measures were calculated in Matlab using the voxel information stored in the segmentation's RAW file. All of the measures used in this thesis are described in Section 3.1.

5.7 Attribute Selection

Attribute selection was performed using WEKA for this case study. The purpose of attribute selection was to identify if there was a correlation between the attributes (i.e., measures).

Attribute selection is used to remove attributes that are irrelevant distracters for the classifier. For example, when decision trees are being created by a divide and conquer approach, each time the data is split, there is less data available to make a decision as to which attribute to use next or whether the algorithm should stop. There may be cases where the classifying algorithm may select an attribute that looks good by chance, but when applied to another set of data, the testing set, the attribute may not be useful in classification at all. Therefore by only allowing the classification algorithm to use the attributes that are considered to be good descriptors of the data, a better classifier can be produced and the problem avoided [19].

For this case study three attribute selection techniques were investigated, J48 Wrapper, J48 Classifier and a CFS Filter. These are described in more detail in Section 3.2.

5.8 Attribute Sets

In order to compare the different types of comparison measures used, multiple classifiers were trained, each with a different attribute set. The purpose of using multiple attribute sets was to answer research question 1, which was to determine which types of comparison measures, among overlap, volume, or geometrical measures, were necessary to obtain optimal results for the medical imaging problem presented in this cases study. Also to answer research question 3, the measures that were selected by the attribute selection techniques must also be used to train the classifiers. A summary of the attribute sets and the short hand notation that will be used below can be seen in Table 5-3.

Attribute Set	Description
OV	Overlap and Volume Difference Measures
OVCfs	Overlap and Volume Difference Measures chosen by the CFS filter using best first search.
OVW	Overlap and Volume Difference Measures chosen by the J48 Wrapper using best first search.
OVC	Overlap and Volume Difference Measures chosen by the J38 Classifier using exhaustive search
G	Geometrical Measures
GCfs	Geometrical Measures chosen by the CFS filter using best first search.
GW	Geometrical Measures chosen by the J48 Wrapper using best first search.
GC	Geometrical Measures chosen by the J38 Classifier using exhaustive search
A	All measures
ACfs	All measures chosen by the CFS filter using best first search.
AW	All measures chosen by the J48 Wrapper using best first search.
AC	All measures chosen by the J38 Classifier using exhaustive search

Table 5-3 Attribute sets

5.9 Description of Classifiers

The case study investigated the use of rule induction techniques and decision trees for machine learning. Three different machine learning classification techniques were chosen, namely JRIP, PART and J48. These three classifiers are the ones used in a similar case study [11] so that the results of the two studies could be compared. During training, 10-fold cross-validation was used, to ensure the overall estimate of error was reasonable. The minimum number of instances associated with a rule or leaf was changed to 10 for all three classification algorithms. This was done to help ensure the

classification algorithms did not over-fit the training data, by creating rules or tree branches that are only valid for the training instances.

The trained classifiers were compared through statistical analysis to determine which classifier and set of attributes yielded the best classifier for the testing framework. The evaluation criteria used were accuracy, the kappa statistic and the area under the ROC curve. Definitions are provided in Section 3.2.

5.10 Experiment Set Up and Description

In this case study four different experiments were performed in an effort to answer the five research questions proposed in Section 1.2. Experiments 1, 2, and 3 all aim to answer research questions 1, 2 and 3, by training multiple classifiers with each of the attribute sets identified in Section 5.8. From there the resulting accuracy, kappa statistic value and area under the ROC curve for each classifier can be compared to determine the answers to research questions 1, 2 and 3. Experiment 4 examines a way to accelerate learning by artificially producing segmentations to obtain a larger training set without using multiple versions of the segmentation algorithm. By comparing the performance of the classifiers produced with and without artificially created segmentations an answer to research question 4 can be provided. In Experiment 3 the trained classifiers are examined to see if there is a way of improving classifier accuracy by manually evaluating a subset of the segmentations in the test set, in an effort to answer research question 5.

Segmentation Target	Tumor
Number of attributes used	18
Comparison Measure Types	Overlap, Volume Difference, Geometrical
Machine Learning Classification Techniques	J48, JRIP, PART
Attribute Selection Techniques	J48 Wrapper, J48 Classifier, CFS Filter
Classifier Evaluation Technique	Stratified 10-fold cross-validation
Performance Metrics	Accuracy, Kappa Statistic, Area under ROC curve (AUC)

Table 5-4 Common characteristics of the set up of all experiments

Table 5-4 contains summarizes characteristics common to all four experiments. The following sections contain the additional information for each of the four experiments in this case study.

5.10.1 Experiment 1

The first experiment was a straight forward application of the proposed workflow for testing medical imaging software. Image segmentations were produced from two versions of the OncoCare algorithm. Algorithm 1 produced segmentations of poor quality when compared manually to segmentations produced by Algorithm 2. To test the classifiers trained in this experiment, Algorithm 3 was used. Table 5-2 is a summary of the details of Experiment 1. Refer to Section 6.1 for the results of Experiment 1.

True Segmentations (TS) Algorithm	A1
Segmentations Under Test (SUT) Algorithm	A2
Number of test cases	145 pairs of tumors
Number of unusable test cases	44 pairs of tumors
Training set size (Number of usable test cases)	101 pairs of tumors
Number of consistent class instances	78 pairs of tumors
Number of inconsistent class instances	23 pairs of tumors
Test set algorithm	A3
Test Set size	145 pairs of tumors

Table 5-5 Details of experiment 1

Using the segmentations produced from A1 and A2, 101 pairs of segmentations were considered to be useable as a training set, of which 78 pairs of segmentations were of the consistent class. For the testing set segmentations from A3 were paired with segmentations from A2 to create a testing set of 145 pairs of segmentations. Though the training set size for this experiment, and in particular the number of segmentation pairs that represented the inconsistent class, are in our view too small to effectively support machine learning, this is considered a first step.

5.10.2 Experiment 2

The second experiment performed was another straight forward application of the proposed workflow for testing medical imaging software. Segmentations were produced from two versions of the OncoCare algorithm. Algorithm 2 and Algorithm 3 produced segmentations of similar quality when compared manually, but Algorithm 3 did perform slightly better when the comparing the number of segmentations that were manually evaluated at 3 or higher. The testing set was comprised of segmentations produced by extended functionality in Algorithm 3, as discussed in Section 5.4. Table 5-6 is a

summary of the details of Experiment 2. Refer to Section 6.2 for the results of Experiment 2.

True Segmentations (TS) Algorithm	A2
Segmentations Under Test (SUT) Algorithm	A3
Number of test cases	145 pairs of tumors
Number of unusable test cases	37 pairs of tumors
Training set size (Number of usable test cases)	108 pairs of tumors
Number of consistent class instances	83 pairs of tumors
Number of inconsistent class instances	25 pairs of tumors
Test set algorithm	A3 extended
Test Set size	135 pairs of tumors

Table 5-6 Details of experiment 2

Using the segmentations produced from A2 and A3, 108 pairs of segmentations were considered to be useable for machine learning, of which 83 pairs of segmentations were of the consistent class. For the testing set segmentations from A3 extended were paired with segmentations from A2 to create a testing set of 135 pairs of segmentations. As in Experiment 1, the training set size for this experiment and the number of segmentation pairs that represented the inconsistent class were relatively small.

5.10.3 Experiment 3

Experiment 3 followed the same procedure as 1 and 2, but it used a larger training set. By manually testing the first few versions of the software, the data available for the training set would be much larger, thus leading to more accurate machine learning classifiers.

In this experiment the data from Experiment 1 and Experiment 2 were combined to make a larger training set. The details for Experiment 3 can be seen in Table 5-7. Refer to Section 6.3 for the results of Experiment 3.

True Segmentations (TS) Algorithm	A1 and A2
Segmentations Under Test (SUT) Algorithm	A2 and A3
Number of test cases	292 pairs of tumors
Number of unusable test cases	83 pairs of tumors
Training set size (Number of usable test cases)	209 pairs of tumors
Number of consistent class instances	161 pairs of tumors
Number of inconsistent class instances	48 pairs of tumors
Test set algorithm	A4
Test Set size	146 pairs of tumors

Table 5-7 Details of experiment 3

Using the segmentations produced from A1, A2 and A3, 209 pairs of segmentations were considered to be useable for machine learning, of which 161 pairs of segmentations were of the consistent class. For the testing set segmentations from A4 were paired with segmentations from A3 to create a testing set of 146 pairs of segmentations.

A1 and A3 could also have been paired to create a larger training set, but this work was not pursued as the quality of A1 was very poor and it was felt would not actually add to helping the classifier learn what would be diagnostically similar between two segmentations.

Once the classifiers in Experiment 3 were trained, the individual rules and leaves produced by the classifier algorithms were examined to identify rules or leaves that were performing poorly. Then, in the testing of Algorithm 4, the pairs of segmentations that were classified by the poorly performing rules or leaves were selected to be manually

evaluated. This was done in order to see if by selecting a set of segmentations to manually evaluate, the accuracy of the classifier could be greatly improved. The set up and procedure followed are described in Section 5.11. The results of identifying poorly performing rules and leaves can be seen in Section 6.5.

5.10.4 Experiment 4

The fourth experiment examined the use of artificially creating image segmentations to use when training a new classifier. The purpose of using the artificially created image segmentations was to get a large training set size quickly and to accelerate the machine learning process. The training sets used in Experiment 1 and 2 were small and did not contain a balanced proportion of the two classes that were to be learned. The training set sizes in those experiments were just over 100 instances, with only approximately a quarter of those instances being of the inconsistent class. Therefore, the purpose of Experiment 4 was both to increase the training set size and to increase the number of instances in the inconsistent class.

Experiment 4 was comprised of three iterations. Each iteration differed by increasing the size of the training set by approximately 100 instances, which was a new segmentation for each tumor available for testing. The point of repeating Experiment 4 for three iterations is to see how the results evolve as the number of instances in the training set increases. The details for the first iteration, denoted Experiment 4.1, can be seen in Table 5-8. Table 5-9 contains the details for Experiment 4.2 and Table 5-10 contains the details for Experiment 4.3. Refer to Section 6.4 for the results of Experiment 4.

True Segmentations (TS) Algorithm	A2
Segmentations Under Test (SUT) Algorithm	A3 extended
Training set size	205 pairs of tumors
Number of consistent class instances	118 pairs of tumors
Number of inconsistent class instances	87 pairs of tumors

Table 5-8 Details of experiment 4.1

Using the 108 usable segmentations produced from A2 and A3, the extended functionality of A3 was used to bring the training set to 205 pairs of segmentations for Experiment 4.1, of which 118 pairs of segmentations were of the consistent class.

True Segmentations (TS) Algorithm	A2
Segmentations Under Test (SUT) Algorithm	A3 extended
Training set size	303 pairs of tumors
Number of consistent class instances	154 pairs of tumors
Number of inconsistent class instances	149 pairs of tumors

Table 5-9 Details of Experiment 4.2

Experiment 4.2 used the training set from Experiment 4.1 and added to it again using the extended functionality of A3 to bring the training set to 303 pairs of segmentations that were considered to be useable for machine learning, of which 154 pairs of segmentations were of the consistent class.

True Segmentations (TS) Algorithm	A2
Segmentations Under Test (SUT) Algorithm	A3 extended
Training Set size	403 pairs of tumors
Number of consistent class instances	188 pairs of tumors
Number of inconsistent class instances	215 pairs of tumors

Table 5-10 Details of experiment 4.3

Experiment 4.3 again used a larger training set by adding to the Experiment 4.2 training set with another set of segmentations produced by the extended functionality of A3.

Experiment 4.3 had 403 pairs of segmentations that were considered to be useable for machine learning, of which 188 pairs of segmentations were of the consistent class.

5.11 Poorly Performing Rules or Leaves Description

In an effort to continue to improve the accuracy of the trained classifiers, the classifiers from Experiment 3 were examined to identify poorly performing rules or decision tree leaves. The set of rules or the tree that was produced by WEKA contained the following additional information for each rule or leaf in the tree:

- the number of training instances that were classified by the rule or leaf (N)
- the number of training instances that were classified incorrectly by the rule or leaf (N_i)

From there it can be determined if a rule or leaf was performing poorly by calculating the accuracy for each rule and leaf as $1-(N_i/N)$. The threshold set for this case study was that the rules or leaves that had an accuracy of 75% or less in training were considered to be performing poorly. This threshold was chosen, as the average accuracy of the classifiers in Experiment 3 was 77.6%, so the threshold was rounded down to 75%, as to work with a simpler value. Therefore if the classifier had a specific rule or node performing worse than the average accuracy, identifying and manually evaluating at the specific rule or node would allow for an improvement in overall accuracy.

Once the rules or leaves were evaluated and tagged as being poorly performing, the test set was applied to the trained classifier. It was then determined manually which testing instances would be classified by which rule or leaf. This was done by entering the data

and rules into an excel spreadsheet, and determining which test instances were classified by which rules or leaves. From there it could be determined the number of testing instances that would need to be manually evaluated for the specific classifier and how many classification errors could be caught by manually evaluating the segmentations at poorly performing rules or leaves. By manually evaluating the segmentations classified by the poorly performing rules or leaves, there is a decrease in the cost savings associated with using machine learning to automatically classify the segmentation pairs. The advantage to this is the increase in overall accuracy gained. If a rule or leaf is really a poor classifier, the cost of manual evaluation is worth it, but the increase in accuracy also depends on the accuracy of the person doing the manual evaluation. Here it is assumed the manual classifications are perfect and all incorrectly classified segmentation pairs are caught during manual evaluation. The results for poorly performing rules or leaves can be found in Section 6.5.

6 Results of Case Study

The following sections are a discussion of the results for the four experiments performed for this case study. Recall Experiments 1, 2 and 3 aim to answer research questions 1, 2 and 3 that were presented in Section 1.2, Experiment 4 aims to answer research question 4, and research question 5 is answered using the results from Experiment 3 in Section 6.5. The numeric results for the performance of the classifiers can be seen in Appendix B Statistical Data on Classifiers.

6.1 Results of Experiment 1

Experiment 1 involved a straightforward application of the proposed testing workflow from Section 4. The three classifiers were each trained with each of the attribute sets in Table 5-3. The resulting accuracy, kappa statistic and AUC values of the trained classifiers can be seen in Figure 6-1. The graphs on the left indicate the accuracy, kappa statistic and AUC values that were achieved by the classifiers when the training set was applied. The graphs on the right are the values for when the testing set was applied to the learnt classifiers.

Figure 6-1 The resulting graphs for experiment 1

For most of the attribute sets, the accuracy of the classifiers was very low on the testing set, when compared to the training set. The average drop in accuracy was 5.8% and the largest drop in accuracy was 19.4%, which was the classifiers trained using the J48 Wrapper attribute sets (GW, AW). The exceptions of this drop in accuracy were the overlap and volume attribute sets (OV, OVCfs, OVC) for PART and J48 classification techniques and the J48 Classifier attribute sets (GC, AC) for all three classifiers. The

drop in performance could be a result of a relatively small training set size that did not have a good proportion of consistent and inconsistent pairs of segmentations. With a very small training set, the number of possibilities the classifier learns is limited, and the classifier is more likely to end up with an instance it does not know how to classify, and classifies it by chance. Also, by not having a balanced proportion of the two classes being learnt, the classifier may not actually learn the difference between consistent or inconsistent pairs of segmentations. Therefore, the more instances that are available for training means a better classifier can be produced.

The overlap and volume attribute sets (OV) mostly have kappa statistic values very close to 0, meaning the classifier was not faring better than chance and in this case all of the instances were being classified to be of the same class, specifically in the consistent class. This is a result of not having enough training instances of the inconsistent class, so these classifiers do not learn what measures and their values make two segmentations inconsistent.

The overlap and volume difference measures chosen by the J48 Wrapper attribute set (OVW) have an accuracy, kappa statistic value and AUC of 0 in both training and testing, because this attribute selection technique chose no attributes, therefore there was nothing to train from.

When using the accuracy to compare the three different measure classifications (OV, G, A), it can be said the geometrical (G) and all (A) measures outperformed overlap and volume differences (OV) measures on this specific training set. The OV measures accuracy was just above 71%, while the G and A measures were both over 77% for the

PART and J48 classifiers. On the testing set OV measures did have higher accuracies than G and A measures, but the kappa statistic being close to 0 showed that most classifiers that used OV measures were classifying all testing instances to be in the consistent class. The AUC values for G and A measures were all larger than 0.68, which indicates better performance than OV measures, which were all 0.67 or less in training. Therefore, in this experiment, G and A measures were outperforming OV measures.

In this experiment, the three classification techniques used gave very similar results. J48 and PART trained classifiers did have better accuracy on the training set, but performed very poorly on the testing set. Therefore, this experiment does not suggest that one of these three classification techniques outperforms the other two.

The J48 Classifier attribute selection technique (OVC, GC, AC) did outperform the other two attribute selection techniques used when the classifiers are applied to the test set with respect to accuracy and kappa statistic results. The J48 Classifier did have a lower AUC in training, but is equal to the other attribute selection techniques in testing. However, both PART and J48 classification techniques with the OVC attribute set classified all test instances as being part of the consistent class, as shown by the kappa statistic being very close to 0. The J48 Classifier attribute selection technique was best used with the geometrical (GC) or all (AC) measures in this experiment.

6.2 Results of Experiment 2

Experiment 2 is very similar to Experiment 1. Experiment 2 was another straight forward application of the proposed testing workflow from Section 4. The algorithms used to

create the training set for machine learning in Experiment 2 were very similar when the quality of segmentation produced was compared. The accuracy, kappa statistic and AUC values of the trained classifiers for Experiment 2 can be seen in Figure 6-2. The graphs on the left indicate the accuracy, kappa statistic and AUC values that were achieved by the classifiers when the training set was applied. The graphs on the right are the values for when the testing set was applied to the learnt classifiers.

Figure 6-2 The resulting graphs for experiment 2

The testing set in Experiment 2 was comprised of artificially created segmentations produced by Algorithm 3 extended functionality. The details on the creation of the artificial segmentations are discussed at length in Section 5.10.4.

The accuracy for Experiment 2 is much lower on the testing set when compared to the training set. The reasoning behind the drop in accuracy from training to testing was discussed in Experiment 1.

There was not a significant difference between the accuracies of the overlap and volume (OV), geometrical (G) and all (A) measures in training. The worst performing of these three measures in training was the geometrical measures. However, when the trained classifiers were applied to the test set, A measures did perform significantly better than OV and G measures on average A performed 7.5% better than the other two measure types. The kappa statistic for A measures was also significantly higher, on average 0.21, than the other two measure sets. This means when using A measures to train the classifier, the classifier was classifying less by chance. The AUC values for OV, G and A measures are very similar, with the majority being between 0.65 and 0.75 during training. Therefore using the results from the accuracy and kappa statistic values, similarly to Experiment 1, A measures performed best in this experiment.

The three classification techniques used performed very similarly when the accuracy, kappa statistic and AUC values were compared. JRIP performed significantly differently than PART and J48 for some attribute sets in testing. In particular, JRIP using geometrical (G) measures performed significantly better, just over a 9% increase in accuracy in training, than the other two classification techniques. For overlap and

volume (OV) and all (A) measures the three classification techniques performed the same. Therefore this experiment, like Experiment 1, does not suggest the use of one classification technique is better than the other two.

When comparing the three attribute selection techniques by the accuracy of the classifiers (ACfs, AW, AC), all three were very similar to using all measures (A) on the training set. However, the kappa statistic values for the J48 Wrapper technique using geometrical measures (GW) were very close to 0, meaning almost all instances were classified by chance. Also, the AUC values for GW are significantly lower in training, all 0.5 or lower, meaning the classifier is producing just as many true positives as false positives. When the J48 Wrapper using geometrical measures (GW) was applied to the test set, it performed very well for the PART and J48 classification techniques. Based on the kappa statistic values obtained from training, the J48 Wrapper technique using all measures (AW) was the best attribute selection technique. When this classifier was applied to the testing set, it still performed well.

6.3 Results of Experiment 3

Experiment 3 in this case study used the training data from both Experiment 1 and Experiment 2 to create a larger training set without using artificially created segmentations that were used in Experiment 4. The accuracies of the trained classifiers can be seen in Figure 6-3.

The trained classifiers were then tested using the testing set composed of segmentations from Algorithm 3 (TS) and Algorithm 4 (SUT). The graphs on the left indicate the

accuracy, kappa statistic and AUC values that were achieved by the classifiers when the training set was applied. The graphs on the right are the values for when the testing set was applied to the learnt classifiers.

Figure 6-3 The resulting graphs for experiment 3

While there was a major drop in accuracy for some trained classifiers from training to testing, in particular G, GCfs, and ACfs using JRIP, there were classifiers that maintained a similar accuracy on the testing set to the training set.

In training, all (A) measures outperformed the overlap and volume (OV) and geometrical (G) measures for the J48 and PART classification techniques, as all (A) measures had higher accuracy, kappa statistic and AUC values in training. The accuracy was on average 5% higher, the kappa statistic value was on average 0.13 higher and the AUC value was on average 0.1 higher. All of these values indicate A measures were outperforming OV and G measures for J48 and PART classifiers. For the JRIP classification technique, geometrical (G) measures performed best. In testing, the geometrical (G) measures performed the worst, out of the three. Overlap and volume measures (OV) and all (A) measures performed very similarly on the test set. Experiment 3 does not suggest that one of the three measures will outperform the others.

In training and testing there was no significant difference between the three classification techniques used. The only very significant difference was the large kappa statistic value. (0.55) for the JRIP classification technique using the J48 Wrapper attribute selection technique applied to all attributes (AW). The large kappa statistic value indicates the PART classifier using the AW attribute set performed significantly better than chance on the test set, which is the case for most of the other classifiers on the test set. There is no indication why this specific classifier is significantly outperforming the others on the test set, as it is not a pattern seen in the first two experiments of this case study.

In this experiment the J48 Classifier attribute selection technique (OVC, GC, AC) performed the worst in training. Both the J48 Wrapper and the CFS filter using all measures (AW, ACfs) performed better than or very similarly to using just all measures (A) in training. These two attribute selection techniques also had accuracy values very

close to all (A) measures in testing. However, the kappa statistic values showed the JRIP classification technique using the J48 Wrapper attribute selection technique applied to all attributes (AW) was the best classifier in testing. AW is the only classifier without a kappa statistic value that was close to 0, meaning it was classifying the test instances better than chance.

As mentioned above, all measures outperformed the others for the PART classification technique. The rules shown below were produced by this data set and classification technique:

1. IF (PrattFOM > 0.925202) THEN Class=Consistent
2. ELSE IF (TPVF > 0.827211) AND (ODI <= 1.780082) THEN
Class=Consistent
3. ELSE Class=Inconsistent

The best performing classifier that used the JRIP classification technique was obtained using the attribute set as selected by the J48 Wrapper attribute selection technique. The rules for this classifier are shown below:

1. IF (PrattFOM > 0.877484) AND (TPVF <= 0.811513) THEN
Class=Inconsistent
2. ELSE Class=Consistent

Finally, in Figure 6-4 is the best performing classifier, for this set of training data, using the J48 Classifier. The attribute set used here was selected by the J48 Wrapper technique from all attributes.

Figure 6-4 Experiment 3 – The resulting tree of using the J48 Classifier and the J48 Wrapper attribute selection technique as selected from all attributes

In comparison, Figure 6-5 shows the tree that was built using the J48 Classifier from all attributes, using not attribute selection technique.

Figure 6-5 Experiment 3 – The resulting tree of using the J48 Classifier and all attributes

As it can be seen by viewing the set of rules and trees presented here, the classifiers and attribute selection techniques did tend to select the same attributes as being the most decisive for determining if a pair of segmentations are consistent or not. It is interesting to note that PART and J48 came up with identical decisions, as both techniques use the C4.5 algorithm. By viewing these examples it is easy to see why all 3 classification

techniques are performing similarly, as they are selecting the same attributes. Another thing to note from these examples is that the use of an attribute selection technique did come up with a slightly different outcome in this case, which resulted in a better classifier that was not over fitted to the training data.

In conclusion, the fact that the kappa statistic is very close to 0 for most of the classifiers in testing, shows that even with the training set that is larger than Experiment 1 and Experiment 2, the classifiers are still not learning the difference between a consistent segmentation pair and an inconsistent one. The AUC values in testing also show that the ratio of true positive results to false positive results is not very high, as the AUC values are all under 0.64. Therefore, more work needs to be done in finding ways to expand the training set size and ensure the balance of consistent and inconsistent examples in training, in order to produce better classifiers.

6.4 Results of Experiment 4

Experiment 4 in this case study used artificially created segmentations to accelerate machine learning and thus solve the problems raised in the first three experiments. The details on the creation of the artificial segmentations are discussed in Section 5.10.4. Experiment 4 was run three times, each time the number of training instances was increased by approximately 100. The accuracy and kappa statistic values of the trained classifiers can be seen in Figure 6-6 and the AUC values can be seen in Figure 6-7.

Figure 6-6 The resulting accuracies and kappa statistic values for experiment 4

Figure 6-7 The resulting AUC values for experiment 4

Approximate Training Set Size	Average Accuracy	Average Kappa Statistic	Average AUC
100 (Experiment 2)	75.8%	0.27	0.66
200	81.1%	0.60	0.79
300	84.9%	0.70	0.86
400	86.7%	0.73	0.87

Table 6-1 Average accuracy, kappa statistic and AUC values of the trained classifiers for different training set sizes

As expected, as more artificial created segmentations were created and the number of instances in the training set went up, the accuracies and kappa statistic values also increased. Table 6-1 presents the average accuracy, kappa statistic and AUC values for each iteration of Experiment 4. The classifiers that were trained with approximately 100 instances are from Experiment 2 and no artificially created segmentations were used. The 100 instances in the training set size are presented as a baseline as to how the

classifiers were performing with a small training set size. In Experiment 2 the average accuracy of the trained classifiers was 75.1%, the average kappa statistic value was 0.27 and the average AUC value was 0.66. Training the classifiers with a training set size of 200 segmentation pairs resulted in classifiers that were an improvement over Experiment 2 with an average accuracy of 81.1%, an average kappa statistic value of 0.60 and an average AUC value of 0.79. The training set size of 300 instances saw the classifier accuracy to be an average of 84.9%, the kappa statistic value average to be 0.70 and the AUC value to be an average of 0.86. The classifiers trained by the training set with 400 instances, was again an improvement over 300 instances, with an average accuracy of 86.7%, an average kappa statistic value of 0.73 and an average AUC value of 0.87. These values show that increasing the training set size by using artificially created segmentations does significantly improve the accuracy, kappa statistic and AUC values of the trained classifier. This result of high practical importance means large learning sets can be artificially generated to help with the prediction of segmentation correctness. It was also found in Experiment 4, with the greater number of training instances, the differences between the different attribute sets and the different classification techniques got much smaller. For example the difference in accuracy between the classifier with the best accuracy and worst accuracy for 200 training instances is 27.3%. The same difference for 400 training instances is only 4.5%. Therefore, the best trained classifier did not depend on the type of classification technique being used or the attribute set the classifier was trained from. The differences in accuracy, kappa statistic and AUC values of all of the trained classifiers for Experiment 4 with a training set size of 400 were very

small, as shown in the graphs marked Training Set 400 Instances in Figure 6-6 and Figure 6-7.

Another aspect to examine is the rules produced, as the training set size increases. The following set of rules are all for the JRIP classifier using all of the attributes.

First with approximately 200 training instances the following rules were produced:

1. IF (RMSDist >= 2.457709) AND (TC_MF <= 2.738905) THEN
 Class=Inconsistent
2. ELSE Class=Consistent

Next, the rules for approximately 300 training instances were:

1. IF (PrattFOM >= 0.739748) THEN Class=Consistent
2. ELSE Class=Inconsistent

Then finally for approximately 400 training instances, the rules produced were:

1. IF (RMSDist >= 3.564319) THEN Class=Inconsistent
2. IF (RMSDist >= 1.164335) AND (TPVF<=0.796109) THEN
 Class=Inconsistent
3. ELSE Class=Consistent

It is interesting to note that as the training set size increased the attributes that were selected changed. The change in the amount of data did have an effect on what attributes were considered to be most descriptive of the data set.

This trend is also noted with the J48 classification technique. The following three figures show the trees produced for approximately 200, 300 and 400 training instances respectively.

Figure 6-8 Experiment 4 (200) – J48 Classifier using all attributes

Figure 6-9 Experiment 4 (300) – J48 Classifier using all attributes

Figure 6-10 Experiment 4 (400) – J48 Classifier using all attributes

As more information becomes available to the classifier, because of larger training set size, the classifier is able to build more complex trees without over fitting to the training data. Here, between 300 and 400 training instances the same attributes are selected, but a slight change in the order they are used to split the data is observed.

When comparing these examples to those presented in Experiment 3, Section 6.3, it can be noted that the same attributes seem to be used in most of these cases. However, further analysis of the number of times each attribute was selected throughout all four experiments would need to be done in order to determine if there are a select few attributes that seem to describe the data set best. Since there are 12 attribute sets presented in this thesis, and in total 6 experiments (as Experiment 3 was performed 3 times), in order to do this type of evaluation on just one classification technique, 72 classifiers would need to be examined and the data organized. Time constraints did not allow for this analysis to be done in this case study, as more effort was spent trying to find a way to get a large training set size without waiting for multiple versions of the algorithm under test to be available. It was felt this was potentially more promising because it is noted that with machine learning the more training data available, the better the classifiers that can be produced.

6.5 Results of Identifying Poorly Performing Rules and Leaves

For each of the three classification techniques used, JRIP, PART and J48, the trained classifier from Experiment 3 with the highest accuracy was chosen for this investigation. The graph showing the accuracies of all of the classifiers on the training set from Experiment 3 can be seen in Figure 6-3. Table 6-2 identifies the three chosen classifiers by listing the classification technique and attribute set that were used in training. The three chosen classifiers will be discussed in the following three sections respectively.

Classification Technique	Attribute Set	Classifier Accuracy
JRIP	AW - All measures chosen by the J48 Wrapper using best first search	80.4%
PART	ACfs – All measures chosen by the CFS Filter using best first search	82.8%
J48	AW - All measures chosen by the J48 Wrapper using best first search	82.3%

Table 6-2 The attribute sets and classification techniques of the trained classifiers chosen for investigating poorly performing rules and leaves

6.5.1 JRIP

Using the JRIP classification technique with the measures chosen by the J48 Wrapper attribute selection technique from all of the attributes, the following rules were produced:

1. IF (PrattFOM <= 0.877484) AND (TPVF <= 0.811513) THEN
Class=Inconsistent (28.0/7.0)
2. ELSE Class=Consistent (181.0/27.0)

Rule 1 had 28 training instances of which 7 were incorrectly classified, meaning the rule's performance was 75.0%. Rule 2 had 181 training instances, with 27 incorrectly classified. Rule 2 was performing at 85.1%. Therefore, for the JRIP classification technique using J48 Wrapper attribute selection, Rule 1 was identified as being a poorly performing rule. When the testing set was applied, the testing instances that were classified by Rule 1 were to be evaluated manually. The results of this can be seen in Table 6-3.

Rules to be manually evaluated	Rule 1
Number of testing instances to be manually evaluated	42
Number of testing instances incorrectly classified by poorly performing rule	37
Accuracy of classifier from WEKA on testing set	69.0 %
Accuracy of classifier once testing instances have been manually evaluated	94.5%
Increase in accuracy	25.5%

Table 6-3 Results of manually evaluating poorly performing rules for the JRIP classifier

For the JRIP classifier chosen, 42 segmentation pairs were to be evaluated manually. By evaluating these 42 segmentation pairs it was found that 37 segmentation pairs were incorrectly classified. Therefore, by correcting the classification of the 37 incorrectly classified segmentation pairs, the number of instances classified correctly goes from 100 to 137 and the classifier accuracy is now 94.5% (137 correctly classified instances divided by 145 total instances). Therefore the increase in accuracy by performing the manual evaluation was 25.5%.

6.5.2 PART

The same procedure was applied to the best performing classifier trained by PART. The classifier used the attributes selected from all attributes by the CFS Filter attribute selection technique. The rules produced by this classifier were:

1. IF PrattFOM > 0.925202 THEN Consistent (106.0/1.0)
2. ELSE IF TPVF > 0.827211 AND ODI <= 1.780082 THEN Consistent (62.0/16.0)
3. ELSE Inconsistent (41.0/10.0)

In training, Rule 1 was performing at 99.1% and Rule 3 was performing at 75.6%. Rule 2 was performing poorly at 74.2%. The results of manually evaluating the instances that terminated at the poorly performing rules during testing can be seen in Table 6-4.

Rules to be manually evaluated	Rule 2
Number of testing instances to be manually evaluated	77
Number of testing instances incorrectly classified by poorly performing rule	6
Accuracy of classifier from WEKA on testing set	66.9%
Accuracy of classifier once testing instances have been manually evaluated	71.0%
Increase in accuracy	4.1%

Table 6-4 Results of manually evaluating poorly performing rules for the PART classifier

For the PART classifier chosen, there were a total of 77 segmentation pairs out of 145 to be manually evaluated. From the 77 segmentation pairs, six were discovered to be classified incorrectly. The total of correctly classified instances then goes from 97 instances to 103 instances out of a total 145 instances and the accuracy of the classifier becomes 71.0%. By re-classifying the 6 segmentation pairs, the increase in accuracy for this classifier was 4.1%.

6.5.3 J48

The J48 classification technique using the attributes selected by the J48 wrapper attribute selection technique was the third classifier used in this investigation. The tree produced from WEKA is shown in Figure 6-11.

Figure 6-11 Tree produced by J48 classification technique using attributes selected by the J48 Wrapper attribute selection technique

It can be seen in Figure 6-11 that the tree produced has three leaves. The leaves will be identified in the remainder of this section as Leaf 1, Leaf 2 and Leaf 3 from left to right on the bottom of the tree. Leaf 1 had 31 training instances classified at that node, with eight of them being incorrect. Leaf 1 was a poorly performing leaf with an accuracy of 74.2% in training. Leaf 2 also was a poorly performing leaf, with 24 of the 72 training instances that were classified at that leaf being classified incorrectly. The accuracy of Leaf 2 was only 66.7%. Leaf 3 was performing at 99.1%. The results of manually evaluating the instances that terminated at the poorly performing leaves during testing can be seen in Table 6-5.

Leaves to be manually evaluated	Leaf 1	Leaf 2
Number of testing instances to be manually evaluated	43	79
Number of testing instances incorrectly classified by poorly performing leaf	38	6
Accuracy of classifier from WEKA on testing set	68.3%	
Accuracy of classifier once testing instances have been manually evaluated	98.6%	
Increase in accuracy	30.3%	

Table 6-5 Results of manually evaluating poorly performing leaves for the J48 classifier

The chosen classifier for the J48 classification technique ended up with 122 segmentation pairs out of 145 to be evaluated manually. From the 122 segmentation pairs that were evaluated manually it was found that 44 segmentation pairs were classified incorrectly. By manually evaluating the 44 instances, the number of correctly classified instances goes from 99 to 143 out of 145 total instances. The resulting accuracy of the classifier after manual evaluation is 98.6% and results in an increase of 30.3% in accuracy for the J48 classifier using the J48 Wrapper attribute selection.

6.5.4 Conclusion for Identifying Poorly Performing Rules or Leaves

From the three above examples, it can be seen that doing some manual evaluation can indeed improve the classifier's accuracy. However, for the PART and J48 classifiers, 122 testing instances out of 145 (84%) were manually evaluated to see the increase of approximately 30% in accuracy. There is a significant trade-off between the amount of manual work to be done and the increase in accuracy that was gained by doing the manual work. The JRIP classifier had a much better trade-off, as only 42 testing instances out of 145 (29%) were manually evaluated and the increase in accuracy was

just over 25.5%. Therefore, it would be suggested that before using manual evaluation to improve classifier accuracy, the percentage of instances that would need manual evaluation would have to be investigated.

Another option to improve this technique would be to use the individual accuracy of the classifier as the threshold, instead of a general value for the threshold. This would mean that the rules or leaves that were performing less than the overall accuracy of the classifier would be chosen, and the accuracy would be improved, as the weakest rules or leaves would be performing better because of manual evaluation.

However, in all cases the number of segmentation pairs that are manually evaluated would be less than evaluating the entire set, so the time required to test the software algorithm would be much shorter.

6.6 Answering the Research Questions

With respect to research question 1, (i.e. Which measures perform best for machine learning?) it was found through experiments 1, 2 and 3 that geometrical and all measures tend to outperform overlap and volume measures. Overlap and volume measures tended to produce classifiers with very low kappa statistic values, meaning the classifier was classifying no better than chance. The accuracy was high for overlap and volume measures, but that was because the classifier tended to classify everything as being in the consistent class and in the training and test sets the number of pairs that were of the inconsistent class was small. Therefore, simple measures that only take into account the tumor size and the amount of overlap between the two segmentations being compared do

not create good classifiers. This could be because tumors are of many different shapes and sizes, and these simple measures do not account for segmentations that may be inconsistent because the actual shape that has been marked is wrong diagnostically. The geometrical measures seemed to be of more use when trying to find the differences between consistent and inconsistent segmentation pairs, as these measures use the boundaries of the segmentations, the axis of the segmentations and the over and under segmented areas to determine if two segmentations are diagnostically consistent.

With respect to research question 2, (i.e. Which classifier technique performs best?) it was found that the three classification techniques used, JRIP, PART and J48, performed similarly in experiments 1, 2, and 3. In the experiments there was always an exception where one classification technique may perform extremely well compared to the other two on a certain attribute set, but there was not a consistent trend between the three experiments to draw a conclusion that one classification technique consistently outperformed the other two.

With respect to research question 3, (i.e. Does attribute selection result in better classifiers?) it was found that the use of attribute selection techniques can slightly improve accuracy and kappa statistic values in experiments 1, 2 and 3. Specifically it was found that the wrapper techniques, specifically J48 Wrapper and J48 Classification, tended to produce attribute sets that performed well. However, the gain in accuracy, kappa statistic and AUC values was never significant enough on both training and testing sets to say that attribute selection techniques can significantly improve the performance when compared to not using them. It was also shown in Experiment 3 that the use of

attribute selection techniques does create different classifiers, when compared to just using a classifier on the same set of data.

With respect to research question 4, (i.e. Does the use of artificially created segmentations accelerate learning?), Experiment 4 found that the larger training sets created by using artificially created segmentations did significantly accelerate the learning process. The classifiers trained by the training sets with 400 segmentation pairs had much higher accuracy, kappa statistic and AUC values than those classifiers that were trained with only 200 segmentation pairs. Experiment 3 also dealt with training classifiers using a larger training set size, but did not use any of the segmentations from the extended functionality of A3. The accuracy, kappa statistic and AUC values obtained when training the classifiers in Experiment 3 are very similar to the values obtained in Experiment 4 with 200 training instances. This shows that training sets with segmentations that had been artificially created performed similarly to training sets of the same size that did not contain artificially created segmentations. Therefore it can be concluded that the use of artificially created segmentations for this problem can accelerate the learning process by allowing the tester to not have to wait for multiple versions of the algorithm that is being tested to be available. The method of artificially creating segmentations still needs further investigation to see if this method could be applied to any medical imaging system in general. Experiment 3 could have also been expanded to 300 training instances, by creating pairings between A1 and A3. By doing this, the use of artificial segmentations could be further confirmed.

With respect to research question 5, (i.e. By investigating ways of improving poorly performing rules or leaves, can an acceptable tradeoff between classifier accuracy and manual segmentation evaluation be found?) the best performing classifiers from Experiment 3 were selected and it was found that manually evaluating segmentation pairs that were classified by poorly performing rules or leaves can improve classifier accuracy. For all three cases selected, the accuracy increased with the aid of manual evaluation. The JRIP classifier selected saw the percentage of accuracy increase of 37%. The manual work required to obtain this increase was the tester would have to manually evaluate 37 pairs of segmentations (25.5% of the testing set). The PART classifier's percentage of accuracy increase was only 6.19%, but the tester would only need to evaluate 6 pairs of segmentations (4.1% of the testing set). The third classifier, the J48 classifier, had a percentage of accuracy increase of 44.44%, but required 122 segmentations pairs to be evaluated manually (84.1% of the testing set). Overall it would be worth investing the time to find out how much of the test set would need manual evaluation before deciding to use the chosen classifier and trying to improve its accuracy by manual evaluation.

6.7 Improving the Certainty of the Results

The results presented for each experiment select the measure set, classification technique and attribute selection technique that performed best for the specific training or testing set. Since only four experiments were performed, the data collected is not enough to draw a general conclusion in order to answer the first three research questions with

complete certainty. Ideally more experiments would be needed to gather the mean performance and standard deviation for each measure set, classification technique and attribute selection technique, to have a more definite answer to the first three research questions. Through this statistical analysis, the answers to the research questions could be determined with more confidence if a certain attribute set, classification technique or attribute selection technique was outperforming the others.

However, the cost of performing a large number of experiments is quite high. For each version of the algorithm there are many steps to perform. Each image must be segmented and the segmentation must be saved. The comparison techniques must then be calculated in Matlab, which currently takes on average two weeks. Once the data is ready, the machine learning can be performed. Since there are three classification techniques, three attribute selection techniques and three measure groups, each experiment has 36 classifiers that must be trained and tested. Therefore using statistical analysis over a large number of experiments was not feasible for this case study.

6.8 Comparison of Results Obtain from a Similar Case Study

A similar case study to the one performed for this thesis was performed by Frounchi in [11]. In the Frounchi case study the medical imaging software that was tested was software that segments the left ventricle of the heart. The experiment performed by Frounchi was very similar to Experiment 3 done in this case study. A comparison of the set up of the two experiments can be seen in Table 6-6.

Case Study	Frounchi case study	Experiment 3 of this case study
Segmentation Target	Left ventricle of the heart	Tumor
Number of test cases in test suite	181	209
Number of attributes (measures)	18	18
Number of positive instances	104	161
Comparison measure types	Overlap, Volume Difference, Geometrical	
Machine learning algorithms	C4.5, JRIP, PART	
Attribute selection techniques	CFS filter J48 Wrapper	CFS filter J48 Wrapper J48 Classifier
Classifier evaluation technique	Stratified 10-fold cross-validation	
Performance Metrics	Accuracy, Kappa Statistic, Area under ROC curve (AUC)	
Research Questions	Both studies aim to answer research questions 1, 2, and 3 that were posed in Section 1.2	

Table 6-6 Comparison of Frounchi case study to experiment 3 of this case study

The Frounchi case study trained the three different machine learning classification techniques for almost all of the same attribute sets as this case study. The exception is the J48 Classifier attribute selection technique was not used in the Frounchi case study. The Frounchi case study included only two versions of the algorithm under test and only one experiment was performed where the classifiers were trained with the test suite data. The case study in this thesis had four versions of the algorithm under test, showing the use of the workflow more in depth. For each experiment in this case study, the classifiers were trained, as well as tested with an independent test set. The Frounchi case study will

be compared directly with Experiment 3 of this case study, as they are the closest in training set size.

The results for the Frounchi case study can be seen in the graphs below. Figure 6-12 shows the accuracy values for the trained classifiers, Figure 6-13 shows the kappa statistic values and Figure 6-14 shows the area under the ROC curve values. These figures can be directly compared with Figure 6-3 from Experiment 3.

Figure 6-12 Accuracy of the trained classifiers from the Frounchi case study

Figure 6-13 Kappa statistic values of the trained classifiers from the Frounchi case study

Figure 6-14 Area under the ROC curve (AUC) values of the trained classifiers from the Frounchi case study

The Frounchi case study determined that all (A) measures performed significantly better than overlap and volume (OV) and geometrical (G) measures, as shown by the significant improvement in accuracy in Figure 6-12. In Experiment 3 for the current case study, all (A) measures did perform slightly better than the other two measures in training, but not to the extent that was seen in Frounchi case study. Therefore it could not be determined through these two case studies that a specific measure type would be best to use over the others for a generalized testing framework.

Neither case study observed a significant difference between the three chosen classification techniques used for machine learning. The Frounchi case study selected the J48 classification technique for all (A) attributes as being the best classification technique for that set of data. In Experiment 3 of the current case study PART performed the best for all (A) attributes using that specific data set. For geometrical (G) measures the JRIP classification technique performed best in both case studies, for their respective data sets. However, these single observations do not lead to an overall selection for a classification technique that performed best.

The Frounchi case study only investigated two attribute selection techniques, the CFS filter (OVC, GC, AC) and the J48 Wrapper (OVW, GW, AW). It was found in the Frounchi case study that the CFS filter degraded the performance of the classifier and the J48 Wrapper outperformed all (A) measures. In Experiment 3 of the current case study this was not observed. The CFS filter (ACfs) and the J48 Wrapper (AW) both performed similarly to all (A) measures and in some cases they did outperform all measures slightly in accuracy. However, the J48 Wrapper (AW) in Experiment 3 did not outperform all

measures to the extent that was observed in the Frounchi case study. Therefore by comparing these two case studies there is no determination that attribute selection techniques significantly improve the performance of the classifier for this type of problem.

Again, all of the results presented are specific to the data sets that were used in each case study, and the same results may not be observed if a different data set was used.

7 Testing Framework

In order to facilitate the creation of a tool that automates the work flow presented in Section 4, the following sections define a testing framework that could be implemented. The purpose of this testing framework is to automate as much as possible of the activity diagram presented in Figure 4-1. The tool would automatically calculate the implemented comparison measures and perform the attribute selection and classifier training. The tool would produce a classifier that could then be used for evaluating new software algorithms in Activity D of Figure 4-1.

The rest of this section is presented as follows. Section 7.1 describes the general testing framework structure using a class diagram and also illustrates the dynamic aspects of the framework in activity diagrams. Section 7.2 shows how this testing framework could be applied to medical imaging systems, like the one used in the case study of this thesis.

7.1 Testing Framework Structure and Activity Diagram

The proposed testing framework class diagram is shown in Figure 7-1. Following the figure is a description of the framework, with a series of activity diagrams that are used to describe the sequence of events that take place for specific aspects of the framework. The framework has two modes it can operate in. The first mode allows the user to train a valid classifier. Once a valid classifier is trained, the second mode is to use the framework and the trained classifier to test a new software algorithm. In other words, the

second mode is used to determine if a new algorithm is consistent with a previous version.

Figure 7-1 Testing framework class diagram

The *LearningProcess* class is the control class. This class performs the actions that are described in the right side of the activity diagram that was presented in Figure 4-1. The framework starts with the template method *templateLP()* of class *LearningProcess* that presents the main flow of the learning process. The activity diagram showing the workflow of the *templateLP()* method is shown in Figure 7-2.

Figure 7-2 Activity diagram of the *templateLP()* method in *LearningProcess*

The *templateLP()* method has 8 steps. In activities 1 to 3 the method prepares the data, which is described in more detail in Section 7.1.1. Then it identifies the pairs of algorithms it has to work with (Activity 4), using the domain expert input. This is discussed in Section 7.1.2. If a valid classifier has not yet been trained, Activity 5 in the above diagram is executed. Activity 5 evaluates the pairs of test case results for each pair

of algorithms (Section 7.1.3). The *templateLP()* method then compares the pairs of test case results using the comparison measures in Activity 6 (Section 7.1.4). The last step in *templateLP()* is to either train a classifier (Activity 7a) or to use the trained classifier to evaluate the algorithm under test (Activity 7b). Both these activities are discussed in more detail in Section 7.1.5. As its name indicate, *templateLP()* is a template method, using the template method design pattern.

7.1.1 Preparing the Data

The first step (Activity 1 in Figure 7-2) for the *templateLP()* method is to call the *addAlgorithm()* method. For each version of the software algorithm that is to be used in the current workflow, the *LearningProcess* uses the *AlgorithmFactory* class, which uses the Factory Method design pattern, to create an instance of the appropriate *Algorithm* class, either *AlgorithmAutomated* or *AlgorithmManual*. The decision depends on whether the framework can automatically run the algorithm being tested (*AlgorithmAutomated*), or the algorithm must first be run in some other way (manually, semi-automatically or automatically), but not by the framework, and the test case results are loaded in to the framework (*AlgorithmManual*). As each *Algorithm* is loaded, the *loadExecute()* method in the *Algorithm* class is also executed. This method loads the test suite (*TS* class) and all of the test cases (*TC* class) that belong to that test suite as identified by the domain expert. Note that this does not necessarily mean that in each execution of the *LearningProcess* the same algorithm version is executed each time on the whole test suite. This does not preclude saving of results of runs and manual

evaluations (on disk) to reuse them when needed. This can be achieved by the use of a Proxy (proxy design pattern). For a single execution of the framework the same test suite must be used for all algorithms, in order to be able to compare the results. The *executeLoad()* method then obtains the test suite result (*TSR* class) and all of the test case results (*TCR* class) either by executing the algorithm, if it is of type *AlgorithmAutomated*, or by loading the results, if the algorithm is of type *AlgorithmManual*. Regardless, the domain expert must manually evaluate each *TCR* for each version of the algorithm, if the framework is operated in the mode used to train a classifier. Each *TCR* instance is assigned a value from the *TCRScale* enumeration class that represents the assigned manual evaluation as done by the domain expert. This is saved to the *TCR* instance by calling *setRating()* in the *TCR* class.

In order to be able to perform any comparison measure calculations, to train a classifier or to classify using a trained classifier, the *LearningProcess* instance requires at least one True Algorithm (*TA*) instance and one Algorithm Under Test (*AUT*) instance created. There may be multiple *TA* instances, as there will be multiple versions of the algorithm available as the workflow progresses. However, at any one time there is only one *AUT*, which is set in the framework by calling the *defineAUT()* method in *LearningProccess* (Activity 2 in Figure 7-2). The method would be passed the *Algorithm* instance, among the ones already known to the framework, which is to be the *AUT* for this specific execution as defined by the domain expert. This can be specified, in the context of class *LearningProcess* as the following OCL expression: *self.TA->includes(self.AUT)*.

The next step (Activity 3 in Figure 7-2) in the *templateLP()* method is to define the data to be used for the classifier training or for classifying the *AUT* if a valid classifier is available. The *LearningProcess* must create the *ComparisonMeasures* to be used. The *addComparisonMeasures()* method uses the *ComparisonsMeasureFactory* to create all the instances of *ComparisonMeasure* - in fact, instances of its subclasses - that are required by the user. After all the instances of *ComparisonMeasure* (i.e., its subclasses) are created, *addComparisonMeasures()* calls *self.data.addMeasure()* in the *ComparisonMeasureData* instance for each *ComparisonMeasure* that was created to be stored for future use in the framework. The *ComparisonMeasureData* class needs to know which comparison measures are used in order to show the user of the framework the specific measures that were selected by the attribute selection technique, or were used by the classification technique.

7.1.2 Algorithm Pairs

The method used to decide if a new algorithm is acceptable is to compare the new algorithm to a previous algorithm and decide if the two algorithms are consistent from the viewpoint of the oracle (i.e., the expert evaluation). Therefore, the framework needs to know how the domain expert would like the *Algorithm* instances to be paired, in order to calculate the comparison measures used for the training data. This is done in the *templateLP()* method by calling the *definePair()* method for each pair of algorithm the user requires (Activity 4 in Figure 7-2). These pairs of algorithms are saved in the *ComparisonMeasureData* class to be used in the rest of the framework. The *definePair()*

method would call `self.data.addAlgorithmPair()` in *ComparisonMeasureData* for each pair of algorithms the user defines.

7.1.3 Evaluating Pairs of Test Case Results (TCR)

Once the *Algorithm*, *TS*, *TC*, *TSR*, and *TCR* instances are defined, the framework can then start to collect the data needed to train the classifier or to classify the *AUT*. If the mode of operation is to train a classifier (`self.theClassifier.isValid()==false`), then the first step is to execute the `evaluate()` method (Activity 5 in Figure 7-2). The activity diagram for this method is in Figure 7-3.

Figure 7-3 Activity diagram for `evaluate()` method in *LearningProcess*

The *evaluate()* method (Figure 7-3) must first retrieve the pairs of algorithms that were defined by the user from the instance of *ComparisonMeasureData* by calling *getAlgorithmPairs()*. The method uses the *Algorithm* instances to retrieve all of the *TC* and *TCR* instances for the *TS*. For each *TCR* pair in the *TS* the method must determine the evaluation, a value from the *Scale* enumeration class, based on the instance of the *EvaluationMethod* class. A *TCR* pair consists of two *TCRs* that are from the same *TC*, but are produced by two different *Algorithm* instances. Since a *TC* may have multiple *TCR* instances associated with it, the *id* of the *TCR* must be able to identify an order among the multiple *TCR* instances for a single *TC* so that it can be verified the pairs of *TCR* being compared are of the same object of interest. Appendix C ID's of Test Case Result suggests an implementation for this problem. The evaluation of the *TCR* pair is used as the class the classifier is trying to learn. For an instance of the *LearningProcess* only one *EvaluationMethod* implementation can be used. The framework cannot compare *Algorithm* instances using different *EvaluationMethod* instances. Once the method has retrieved the pairs of algorithms, the *evaluate()* method in *EvaluationMethod* is called to determine which value from the *Scale* enumeration class will be assigned to each *TCR* pair. The activity diagram for the *evaluate()* method in *EvaluationMethod* is shown in Figure 7-4.

Figure 7-4 Activity Diagram of *evaluate()* method in *EvaluationMethod*

The *evaluate()* method (Figure 7-4) calls the *unusable()* and *consistent()* methods in *EvaluationMethod* in order to determine the evaluation of the pair using the *threshold* value. The result is then saved in the *ComparisionMeasureData* instance by calling the *addEvaluation()* method (Figure 7-3) for the corresponding *PairOfAlgorithms* instance and *TCR* instance. The pairs of *TCR* determined to be unusable are not used when training the classifier and the comparison measures do not need to be calculated for unusable pairs, as per the description of the workflow in Section 4.

7.1.4 Comparing Test Case Results

The *templateLP()* method now calls the *calculateComparisonMeasures()* method (Activity 6 in Figure 7-2). Calculating the comparison measures also corresponds to Activity E in Figure 4-1. The activity diagram for the *calculateComparisonMeasures()* method can be seen in Figure 7-5.

Figure 7-5 Activity diagram for *calculateComparisonMeasures()* method

The *calculateComparisonMeasures()* method calculates the comparison measures for each TCR pair that is useable (i.e., consistent or inconsistent). The *calculateComparisonMeasures()* first obtains the algorithm pairs (*data.getAlgorithmPairs()* method) and set of comparison measures (*self.cm*). The method then loops through all of the TCRs for each pair of *Algorithm* instances to calculate all of the comparison measures. This is done by calling the *calculate()* method for each *ComparisonMeasure* instance and passing it two TCR instances, each from a

different *Algorithm* instance, but the same *TC*. The comparison measure results are saved in the *ComparisonMeasureData* instance by calling the *addValue()* method passing the corresponding *PairOfAlgorithms*, *ComparisonMeasure*, and *TCR* instances, along with the value calculated.

7.1.5 Learning or Classifying

At this point all of the data and class information required to train a classifier or to classify a new algorithm with a learnt classifier is stored in *ComparisonMeasureData*. If the framework is operating in a mode to train a new classifier, the *templateLP()* method would then call *trainClassifier()*. The activity diagram for the *trainClassifier()* method is shown in Figure 7-6.

Figure 7-6 Activity diagram for the *trainClassifier()* method

The first step for *trainClassifier()* is to perform attribute selection, if required, and then the method would train the classifier with the new *ComparisonMeasureData* instance. This corresponds to Activity F in Figure 4-1. The *chooseBestAttributes()* method would in turn call the *selectAttributes()* method of the *AttributeSelection* class for the attribute selection technique that was chosen and implemented and return a new *ComparisonMeasureData* instance with only the data for the selected attributes. The

trainClassifier() method in the *LearningProcess* instance would then evaluate the *Classifier* instance by calling the *evaluateQuality()* method in the *Classifier* instance (Activity G in Figure 4-1). The *evaluateQuality()* method would determine if the trained classifier is valid based on the performance statistics, accuracy, kappa statistic and area under the ROC curve. If an acceptable classifier is produced the *valid* attribute in the *Classifier* instance would be set to true and the process of evaluating new versions of the software algorithm would now be automated (Activity D in Figure 4-1). If the classifier is not acceptable a new learning process would have to begin, with the next version of the algorithm being manually evaluated (Activity C in Figure 4-1) and added as input to this testing framework. In essence, the *AttributeSelection* and *TrainClassifier* classes are adapter classes for the attribute selection algorithm (possibly a machine learning algorithm itself) and the machine learning algorithm used for training, respectively.

Once an acceptable classifier is trained, the framework would be able to automate the testing of new software algorithms. The *templateLP()* method would call the *classify()* method in the *Classifier* instance passing it the *ComparisonMeasuresData* instance without the evaluation data of the *TCR* pairs. This would in turn call the *classify()* method the *TrainClassifier* (adapter) class. Using the performance statistics the *classify()* method in the *LearningProcess* instance would need to decide if the algorithm under test (*AUT*) is consistent to the true algorithm (*TA*) and if the new algorithm is to be accepted by calling the *evaluateClassification()* method in the *Classifier* instance. The framework would also have to notify the user of any *TCR* pairs that would need manual evaluation.

7.2 Application of the Testing Framework to Medical Imaging Systems

In this section we show how the proposed testing framework could be applied to a medical imaging system example. The medical imaging example used is from the case study of this thesis, which involves the testing of the OncoCare software algorithm. Figure 7-7 shows the application of the testing framework to this case. In the figure, classes shown in gray have not changed from Figure 7-1. Classes that are shown in white have been added to the class diagram: the framework is used through extensions.

Figure 7-7 The testing framework class diagram expanded to show the application of the framework to a medical imaging system

An instance of the *Algorithm* class (in fact its subclasses) would need to be created for each version of the OncoCare software that is produced. The OncoCare versions were run by the domain expert or tester, so the *AlgorithmFactory* would be producing *AlgorithmManual* instances, and the *TS* and *TSR* would be loaded into the framework. The test suite (*TS*) would be composed of test images. Each test image is represented by an instance of the *TC* class. The test images are the 3D MRI scans from which the algorithm identifies the tumors. For each *TC*, there would be 1 or more segmentations produced by the algorithm. Each segmentation would correspond to an instance of the *TCR* class.

In order to implement the framework the abstract classes from the testing framework, *EvaluationMethod*, *ComparisonMeasure*, *TrainClassifier* and *AttributeSelection* all would need to be sub-classed. The evaluation method used for the case study is defined in Section 5.5 and would be implemented by inheriting from the *EvaluationMethod*, as shown by the *EvaluationFiveScale* class in Figure 7-7. The *TCRScale* enumeration would have 5 possible values, as shown in the figure above, as the segmentations in this case study were evaluated on a scale of 1 to 5, as discussed in Section 5.5. For this implementation, the *unusable()* method in the *EvaluationFiveScale* would return true if both *TCR* instances in the pair were evaluated to be 2 or lower. Otherwise the *unusable()* method would return false. The *consistent()* method would return true if both *TCR* instances in the pair were evaluated to be 3 or higher and false if one *TCR* instance in the pair was evaluated as 2 or lower and the other 3 or higher. For a pair of segmentations, it should be checked first if the pair is unusable, and then if the pair is consistent. The

evaluate() method in *EvaluationFiveScale* would then return the appropriate result from the *Scale* enumeration, based on the results from the *consistent()* method and the *unusable()* method.

The *ComparisonMeasure* class shown in Figure 7-7 is also an abstract class that would need to be sub-classed for each of the comparison measure that would be used for classifier training. For example, the 18 attributes that were used in the case study, as presented in Table 3-1, would each need to be implemented and they must inherit from the appropriate base class, based on the measure's measurement type. For simplicity of Figure 7-7, not all 18 comparison measures/classes are shown. In Figure 7-7 the overlap measures, such as *TPVF*, inherit from *OverlapAndVolumeMeasures* (abstract class) and the geometrical measures, such as *AvgBoundDist*, inherit from *GeometricalMeasures* (abstract class) (New abstract classes are used to structure the generalization of measures). The *calculate()* method in both of these classes would then calculate the measure and return the value to the *LearningProcess*, to be saved in the *ComparisonMeasureData* class.

The *TrainClassifier* and *AttributeSelection* classes are the last two classes that would need to be sub-classed and implemented. For the case study in this thesis, WEKA was used. WEKA is an open source software package that is written in JAVA. The framework would need a wrapper class for classifier training and a wrapper class for attribute selection in order to interact with the WEKA framework and to perform the set up needed. In Figure 7-7, as an example, class for implementing the J48Wrapper

attribute selection technique (*J48WrapperAS* class) and the JRIP classification technique (*JRIPClassifier* class) are shown.

8 Conclusion and Future Work

This thesis presented a work flow in Section 4 that is used to produce a test oracle to use when testing medical imaging software. The objective of the workflow is to collect enough training data in order to produce a classifier that can automatically verify a new version of the medical imaging software without the tester having to manually compare two versions of the software.

The work flow presented for testing medical imaging software was applied to OncoCare, software that segments tumors in 3D MRI scans with the aid of a medical professional. The medical professional must supply the software the diameter of the tumor as the starting point for the segmentation algorithm. Using four versions of OncoCare, the case study presented here showed the workflow was applicable to this software and also discussed ways to improve the workflow for future use.

As presented in Experiments 1, 2 and 3, the proposed workflow did create classifiers that performed well on both training and testing data. However, these 3 experiments did not show that a specific classification technique using a specific attribute set produced the best classifier consistently. Therefore it would be suggested when applying this work flow in the future or when building a tool for this workflow, the user produce all possibilities of classifiers as was done here, in order to achieve the best classifier for the data available. The process of training classifiers in WEKA is quick, so this would not be a time consuming task. As also suggested in [11], the classification techniques could be expanded to use neural networks or support vector machines to produce classifiers

with better accuracy. However, using these techniques would lead to classifiers that are not easily understandable and intuitive. These classifiers would make it much harder to apply a technique, such as was done in this case study of selecting segmentations requiring manual evaluation to improve classification accuracy.

This thesis also investigated the use of artificially created segmentations to accelerate learning in Experiment 4. By accelerating the learning process, a valid classifier for testing can be found faster, and less versions of the software algorithm are needed before the process can begin evaluating segmentations automatically. The process used in Experiment 4 saves the tester valuable time, as there is less manual testing to be done for the workflow. The method used in this case study was to alter the two end points of the segmentation diameter that was the input for the image segmentation, to obtain a new segmentation. This method did improve classifier accuracy and allowed for the training set for machine learning to be much larger without using multiple versions of the software to grow the training set size. As there is not much research done on artificially creating segmentations and the version of the algorithm used for the case study was stable and being used in practice, the method used would need further investigation. Applying the method to an earlier version of the segmentation software that may not be as stable as the version used here would confirm that the artificially created segmentations would indeed accelerate learning and a tester would not need to test the first few versions of the software manually in order to get enough data for machine learning.

The method of randomly displacing the end points of the saved diameter could also be used to test the sensitivity or robustness of the software algorithm. If a slight change in the diameter, for instance because of a slight error in input by the user, results in a segmentation that is severely unacceptable, the algorithm is very sensitive. A sensitive algorithm may not be as desirable to the user, as it is not as stable, as the result depends on slight variations in the input.

As image segmentation software is currently tested manually, a solution was also investigated to manually test only a subset of the segmentations in the test set that were classified by rules or leaves that were performing poorly. By limiting the number of segmentations to be evaluated manually the time and cost of using a medical professional for the testing was reduced when compared to testing the entire set of segmentations manually. There were cases where the rule or leaf selected as poorly performing classified a significant percentage of the test set. In these cases the tester would still not be evaluating the entire test set manually, so there is still a small savings of time obtained by using automated evaluation.

From this current case study and the case study performed in [11], a generalized testing framework for medical imaging software was developed and discussed in Section 7. This testing framework is the starting point from which in the future a tool could be developed to semi-automate the testing of medical imaging software. The tool proposed is to be adaptable to allow the user to define different evaluation scales for segmentations, comparison measures, and machine learning techniques. The adaptability of the testing tool allows it to be applied to any medical imaging software problem.

References

- [1] A. J. Baddeley, "An error metric for binary images," *Robust Computer Vision*, pp. 59-78, 1992.
- [2] A. Ben-David, "About the relationship between ROC curves and Cohen's kappa," *Engineering Applications of Artificial Intelligence*, vol. 21, pp. 874 - 882, 2008.
- [3] A. P. Bradley, "The use of the area under the curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145 - 1159, 1997.
- [4] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37 - 46, 1960.
- [5] W. W. Cohen, "Fast Effective Rule Induction," *Twelfth International Conference on Machine Learning*, pp. 115-123, 1995.
- [6] W. R. Crum, O. Camara, and D. L. G. Hill, "Generalized overlap measures for evaluation and validation in medical image analysis," *IEEE Transactions on Medical Imaging*, vol. 25, no. 11, pp. 1451-1461, November 2006.
- [7] M. D. Davis and E. J. Weyuker, "Pseudo-oracles for non testable programs," *Proceedings of the ACM '81*, pp. 254-257, 1981.
- [8] X. Deng, L. Zhu, Y. Sun, C. Xu, L. Song, J. Chen, R. D. Merges, M. Jolly, M. Suehling, and X. Xu, "On Simulating Subjective Evaluation Using Combined Objective Metrics for Validation of 3D Tumor Segmentation," *Medical Image Computing Computer-Assisted Intervention*, pp. 977-984, 2004.
- [9] B. Di Eugenio and M. Glass, "Squibs and Discussions The Kappa Statistic: A Second Look," *Computational Logistics*, vol. 30, no. 1, pp. 95 - 101, 2004.
- [10] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, Second Edition, John Wiley & Sons Inc, 2001.
- [11] K. Frounchi, "Learning a Test Oracle towards Automating Image Segmentation Evaluation," M.A.Sc. thesis, Carleton University, Ottawa, ON, Canada, 2008.
- [12] M. A. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining," *IEEE Trans. on Knowledge and Data Eng.*, vol. 15, no. 3, 2003.
- [13] M.P Jolly and L. Grady, "3D general lesion segmentation in CT," *5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 796-799, May 2008.
- [14] D. L. Pham, C. Xu, and J. L. Prince, "A Survey of Current Methods in Medical Image Segmentation," Department of Elec. and Comp. Eng., The John Hopkins

- University, and Laboratory of Personality and Cognition, National Institute on Aging, Technical Report JHU/ECE 99-01, Jan. 1998.
- [15] D. L. Pham, C. Xu, and J. L. Prince, "Current Methods in Medical Image Segmentation," *Annual Review Biomedical Engineering*, pp. 315-338, 2000.
 - [16] J. K. Udupa, V. R. LeBlanc, Y. Zhuge, C. Imielinska, H. Schmidt, L. M. Currie, B. E. Hirsch and J. Woodburn, "A framework for evaluating image segmentation algorithms," *Elsevier Computerized Medical Imaging and Graphics*, vol. 30, pp. 75-87, March 2006.
 - [17] S. K. Warfield, K. H. Zou, and W. M. Wells, "Simultaneous Truth and Performance Level Estimation (STAPLE): An Algorithm for the Validation of Image Segmentation," *IEEE Transactions on Medical Imaging*, vol. 23, no. 7, pp. 903-921, July 2004.
 - [18] E. J. Weyuker, "On Testing Non-testable Programs," *The Computer Journal*, vol. 25, no. 4, pp. 465-469, 1982.
 - [19] I. H. Witten, and E. Frank, *Data Mining; Practical Machine Learning Tools and Techniques*, Second Edition, Elsevier, 2005.
 - [20] Y. J. Zhang, "A Survey on Evaluation Methods for Image Segmentation," *Pattern Recognition*, vol. 29, no. 8, pp. 1335-1346, 1996.
 - [21] H. Zhang, S. Cholleti, and S. A. Goldman, "Meta-Evaluation of Image Segmentation Using Machine Learning," in *2006 Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

Appendix A Statistical Data on Comparison Measures

Experiment 1 Training Set

Comparison Measure	Mean	Standard Deviation	Maximum Value	Minimum Value
FPVF	0.000151	0.000389	0.003983	0
TPVF	0.906691	0.123123	1	0.331171
TC_MF	5.751664	3.75961	25.35238	0.36445
DSC_MF	0.877049	0.102003	0.980659	0.421598
AVD	1.437188	1.03251	9.465482	0.037448
ANVD	0.91581	0.120783	1	0.015018
AvgBoundDist	0.168045	0.31462	2.277305	0.000729
HausDist	8.598566	7.286963	39.71146	1
BaddeleyDist	2.79543	2.510928	13.66973	0.328357
PrattFOM	0.837874	0.146601	0.980778	0.200024
Peli_Malah_Mean	1.031196	1.366957	11.37517	0.181459
Peli_Malah_Var	5.541783	17.35564	233.3155	0
RMSDist	1.697811	1.816293	13.85401	0.422796
SODI	0.058382	0.224933	2.315857	0
ODI	1.512782	1.097161	10.63571	0
SUDI	0.077241	0.157712	1.228014	0
UDI	1.704317	1.200244	8.147421	0
PrincipalAxisDist	21.29943	47.44776	466.84	0.024281

Experiment 2 Training Set

Comparsion Measure	Mean	Standard Deviation	Maximum Value	Minimum Value
FPVF	0.000085	0.000218	0.001511	0
TPVF	0.885124	0.132708	1	0.297951
TC_MF	8.707395	8.33342	45.56623	0.424402
DSC_MF	0.895190	0.101938	0.989146	0.45911
AVD	0.679070	0.738417	4.906722	0.0083
ANVD	0.287420	0.234311	1	0.003822
AvgBoundDist	0.232867	0.300962	1.522548	0.000273
HausDist	6.409683	5.277582	25.92296	1
BaddeleyDist	1.912477	1.740534	10.33431	0.142755
PrattFOM	0.858695	0.137491	0.992407	0.382898
Peli_Malah_Mean	0.857362	0.754592	3.744237	0.035842
Peli_Malah_Var	3.053584	4.044094	41.15172	0
RMSDist	1.365715	1.100723	6.09023	0.210725
SODI	0.029658	0.054965	0.534471	0
ODI	1.264905	0.58982	4.773995	0
SUDI	0.040757	0.056701	0.34357	0
UDI	1.502170	0.668797	4.296637	0
PrincipalAxisDist	20.203455	34.32662	264.2419	0.001486

Experiment 3 Training Set

Comparsion Measure	Mean	Standard Deviation	Maximum Value	Minimum Value
FPVF	0.000118	0.000317	0.003983	0
TPVF	0.895907	0.128239	1	0.297951
TC_MF	7.22953	6.620976	45.56623	0.36445
DSC_MF	0.886119	0.102198	0.989146	0.421598
AVD	1.058129	0.973172	9.465482	0.0083
ANVD	0.601615	0.36563	1	0.003822
AvgBoundDist	0.200456	0.309044	2.277305	0.000273
HausDist	7.504125	6.445017	39.71146	1
BaddeleyDist	2.353953	2.201485	13.66973	0.142755
PrattFOM	0.848285	0.142256	0.992407	0.200024
Peli_Malah_Mean	0.944279	1.1056	11.37517	0.035842
Peli_Malah_Var	4.297683	17.67646	233.3155	0.035842
RMSDist	1.531763	1.305118	13.85401	0
SODI	0.04402	0.16408	2.315857	0
ODI	1.388844	0.888005	10.63571	0
SUDI	0.058999	0.119705	1.228014	0
UDI	1.603243	0.975153	8.147421	0
PrincipalAxisDist	20.75144	41.34212	466.84	0.001486

Experiment 4 Training Set 400 Instances

Comparison Measure	Mean	Standard Deviation	Maximum Value	Minimum Value
FPVF	0.00129	0.004537	0.052248	0
TPVF	0.706029	0.345605	1	0
TC_MF	3.077167	8.25691	114.8163	0
DSC_MF	0.55889	0.314137	0.995664	0
AvgBoundDist	0.816676	1.238583	10.43165	0.000464
HausDist	24.20812	23.84107	128.4406	1
BaddeleyDist	11.31362	12.57939	74.23565	0.126825
PrattFOM	0.509736	0.31379	0.997227	0.001199
Peli_Malah_Mean	6.324222	9.558633	86.4252	0.027731
Peli_Malah_Var	180.3905	649.2737	8068.35	0.027731
RMSDist	9.030721	9.782374	88.39393	0
SODI	1.714405	6.249122	80.6835	0
ODI	6.203091	8.947273	86.4252	0
SUDI	1.061179	3.65263	40.38713	0
UDI	4.632883	7.592887	63.11541	0
PrincipalAxisDist	163.8767	347.2449	2854.931	0.082723

Appendix B Statistical Data on Classifiers

Experiment 1 – JRIP

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	76.2376%	0.1721	0.5750	68.9655%	0.2382	0.6700
OVCfs	76.2376%	0.1398	0.5346	82.7586%	0.0000	0.5000
OVW	0	0	0	0	0	0
OVC	78.2178%	0.1447	0.5686	72.4138%	0.2658	0.6750
G	71.2871%	0.2628	0.6904	62.7586%	0.1517	0.6167
GCfs	78.2178%	0.3991	0.7430	68.2759%	0.2774	0.7133
GW	76.2376%	0.3813	0.7422	62.7586%	0.1517	0.6167
GC	75.2475%	0.3272	0.6204	71.0345%	0.2101	0.6350
A	69.3069%	0.2332	0.6892	69.3069%	0.2332	0.7133
ACfs	69.3069%	0.2332	0.6708	67.5862%	0.1863	0.6300
AW	76.2376%	0.3813	0.7422	62.7586%	0.1517	0.6167
AC	72.2772%	0.1017	0.5644	69.6552%	0.1924	0.6267

Experiment 1 – PART

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	71.2871%	0.0181	0.5635	82.7586%	0.0000	0.5000
OVCfs	73.2673%	0.0508	0.4969	82.7586%	0.0000	0.5000
OVW	0	0	0	0	0	0
OVC	71.2871%	0.0181	0.5635	82.7586%	0.0000	0.5000
G	80.1980%	0.5365	0.7968	62.0690%	0.1618	0.6903
GCfs	81.1881%	0.5651	0.8024	62.0690%	0.1618	0.6903
GW	81.5362%	0.5651	0.8024	62.0690%	0.1618	0.6903
GC	76.2376%	0.1721	0.6380	79.3103%	0.1049	0.6742
A	77.2277%	0.4153	0.7904	62.0690%	0.1618	0.6903
ACfs	80.1980%	0.5365	0.7968	62.0690%	0.1618	0.6903
AW	81.5362%	0.5651	0.8024	62.0690%	0.1618	0.6903
AC	73.2673%	0.1183	0.6672	77.2414%	0.1897	0.6833

Experiment 1 – J48

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	71.2871%	0.0181	0.5635	82.7586%	0.0000	0.5000
OVCfs	73.2673%	0.0508	0.4969	82.7586%	0.0000	0.5000
OVW	0	0	0	0	0	0
OVC	71.2871%	0.0181	0.5635	82.7586%	0.0000	0.5000
G	80.1980%	0.5365	0.7968	62.0690%	0.1618	0.6903
GCfs	81.1881%	0.5651	0.8024	62.0690%	0.1618	0.6903
GW	81.1881%	0.5651	0.8024	62.0690%	0.1618	0.6903
GC	76.2376%	0.1721	0.6377	80.6897%	0.1306	0.6828
A	78.2178%	0.4484	0.7982	62.0690%	0.1618	0.6903
ACfs	80.1980%	0.5365	0.7968	62.0690%	0.1618	0.6903
AW	81.1881%	0.5651	0.8024	62.0690%	0.1618	0.6903
AC	73.2673%	0.1183	0.6725	67.5862%	0.0822	0.6627

Experiment 2 – JRIP

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	75.0000%	0.2666	0.6320	51.1110%	0.0252	0.5127
OVCfs	80.5556%	0.4296	0.6745	61.4810%	0.2706	0.6438
OVW	77.7778%	0.2496	0.5978	51.1110%	0.0252	0.5127
OVC	75.9259%	0.3039	0.6675	57.7778%	0.2050	0.6096
G	78.7037%	0.4405	0.7133	59.2593%	0.2311	0.6233
GCfs	73.1481%	0.3126	0.7246	59.2593%	0.2311	0.6233
GW	76.8519%	0.0000	0.4381	54.0741%	0.0000	0.5000
GC	73.1481%	0.3126	0.6694	61.4815%	0.2689	0.6426
A	74.0741%	0.2713	0.7506	61.4815%	0.2672	0.6414
ACfs	72.2222%	0.1730	0.6224	54.0741%	0.0663	0.5328
AW	78.7037%	0.3153	0.6501	51.1110%	0.0252	0.5127
AC	78.7037%	0.3930	0.6523	51.1110%	0.0252	0.5127

Experiment 2 – PART

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	78.7037%	0.3565	0.6827	51.1111%	0.0252	0.5127
OVCfs	79.6296%	0.3345	0.6682	51.8519%	0.0175	0.5506
OVW	79.6296%	0.3751	0.6219	51.1111%	0.0252	0.5127
OVC	75.0000%	0.3256	0.6706	61.4815%	0.2689	0.6439
G	69.4444%	0.1758	0.6547	54.0741%	0.0000	0.5000
GCfs	69.4444%	0.1758	0.6547	54.0741%	0.0000	0.5000
GW	75.0000%	0.0055	0.4571	63.7037%	0.3007	0.6571
GC	71.2963%	0.2459	0.6798	60.7407%	0.2557	0.6358
A	76.8519%	0.2558	0.7506	60.7407%	0.2574	0.6370
ACfs	76.8519%	0.2558	0.7651	60.7407%	0.2574	0.6370
AW	78.7037%	0.4097	0.7557	58.5185%	0.2180	0.6164
AC	75.9259%	0.2833	0.6270	62.2222%	0.2838	0.6507

Experiment 2 – J48

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	78.7037%	0.3565	0.6723	51.1111%	0.0252	0.5127
OVCfs	79.6296%	0.3345	0.6354	53.3333%	0.0453	0.5223
OVW	79.6296%	0.3751	0.6219	51.1111%	0.0252	0.5127
OVC	76.8519%	0.3402	0.6523	61.4815%	0.2689	0.6426
G	69.4444%	0.1758	0.6547	54.0741%	0.0000	0.5000
GCfs	69.4444%	0.1758	0.6547	54.0741%	0.0000	0.5000
GW	75.0000%	0.0055	0.4571	63.7037%	0.3007	0.6571
GC	71.2963%	0.2459	0.6634	60.7407%	0.2557	0.6358
A	76.8519%	0.2558	0.7511	60.7407%	0.2574	0.6101
ACfs	76.8519%	0.2558	0.7655	60.7407%	0.2574	0.6101
AW	78.7037%	0.4097	0.7494	58.5185%	0.2180	0.6657
AC	75.0000%	0.2666	0.6241	62.2222%	0.2838	0.5832

Experiment 3 – JRIP

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	77.9904%	0.3592	0.7013	53.1034%	0.0879	0.6384
OVCfs	77.9904%	0.3393	0.6842	68.2759%	0.0474	0.5484
OVW	78.4689%	0.3779	0.6694	68.2759%	0.0474	0.5484
OVC	79.4258%	0.3965	0.6997	59.3103%	0.0996	0.6378
G	78.4689%	0.4516	0.7472	60.6897%	0.1077	0.6212
GCfs	78.9474%	0.4742	0.7484	36.5517%	0.0406	0.5822
GW	77.0335%	0.0000	0.4806	91.0345%	0.0000	0.5000
GC	75.1196%	0.2646	0.6362	79.3103%	0.1467	0.6090
A	78.9474%	0.4050	0.7131	68.2759%	0.0474	0.5484
ACfs	79.9043%	0.4236	0.7294	36.5517%	0.0406	0.5822
AW	80.3828%	0.3874	0.7057	68.9655%	0.5522	0.6259
AC	75.5981%	0.3154	0.6618	75.8621%	0.0272	0.5207

Experiment 3 – PART

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	77.5120%	0.3502	0.6949	74.4828%	0.0954	0.6279
OVCfs	77.5120%	0.3502	0.6949	74.4828%	0.0954	0.6279
OVW	77.5120%	0.3403	0.7111	59.3103%	0.0996	0.6378
OVC	77.5120%	0.3403	0.7111	59.3103%	0.0996	0.6378
G	74.6411%	0.2081	0.7602	36.5517%	0.0406	0.5699
GCfs	75.1196%	0.1621	0.7376	36.5517%	0.0406	0.5699
GW	78.9474%	0.2082	0.5397	88.9655%	-0.0348	0.4886
GC	72.2488%	0.0969	0.6661	91.0345%	0.0000	0.5000
A	82.2967%	0.4287	0.8271	66.8966%	0.0387	0.5361
ACfs	82.7751%	0.4394	0.8362	66.8966%	0.0387	0.5361
AW	82.2967%	0.3878	0.8198	68.2759%	0.0474	0.5425
AC	73.2057%	0.1699	0.6163	56.5517%	0.0385	0.5533

Experiment 3 – J48

Attribute Set	Training Set			Test Set		
	Accuracy	Kappa Statistic	AUC	Accuracy	Kappa Statistic	AUC
OV	77.5120%	0.3502	0.6998	74.4828%	0.0954	0.6279
OVCfs	77.5120%	0.3502	0.6998	74.4828%	0.0954	0.6279
OVW	77.5120%	0.3403	0.7111	59.3103%	0.0996	0.6378
OVC	77.5120%	0.3403	0.7111	59.3103%	0.0996	0.6378
G	74.6411%	0.2081	0.7332	36.5517%	0.0406	0.5699
GCfs	75.1196%	0.1621	0.7017	36.5517%	0.0406	0.5699
GW	74.9474%	0.2082	0.5397	88.9655%	-0.0348	0.4886
GC	74.6411%	0.0905	0.6205	91.0345%	0.0000	0.5000
A	79.9043%	0.3873	0.8183	66.8966%	0.0387	0.5332
ACfs	80.3828%	0.3971	0.8247	66.8966%	0.0387	0.5332
AW	82.2967%	0.3878	0.8198	68.2759%	0.0474	0.5425
AC	73.2057%	0.1699	0.6163	56.5517%	0.0385	0.5533

Experiment 4 – 200 Training Instances – JRIP

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	80.9756%	0.6040	0.8244
OVCfs	80.4878%	0.5907	0.7894
OVW	83.9024%	0.6710	0.7738
OVC	81.9512%	0.6278	0.8357
G	80.4878%	0.5895	0.7868
GCfs	82.4390%	0.6317	0.8036
GW	84.3902%	0.6746	0.8001
GC	81.4636%	0.6100	0.7773
A	84.3902%	0.6695	0.7892
ACfs	84.8780%	0.6833	0.8383
AW	83.9024%	0.6587	0.7914
AC	81.9512%	0.6232	0.8118

Experiment 4 – 200 Training Instances – PART

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	80.9756%	0.6088	0.8052
OVCfs	83.4146%	0.6616	0.7768
OVW	83.4146%	0.6616	0.7768
OVC	81.4634%	0.6206	0.8146
G	81.4634%	0.6124	0.7908
GCfs	80.4878%	0.5933	0.7963
GW	84.8780%	0.6872	0.7908
GC	80.4878%	0.5869	0.7381
A	57.5610%	0.0000	0.4810
ACfs	79.5122%	0.5755	0.8198
AW	84.3902%	0.6726	0.8079
AC	77.2414%	0.1932	0.8476

Experiment 4 – 200 Training Instances – J48

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	80.9756%	0.6088	0.8109
OVCfs	83.4146%	0.6616	0.7768
OVW	83.4146%	0.6616	0.7768
OVC	81.4634%	0.6206	0.8178
G	80.4878%	0.5920	0.7874
GCfs	80.4878%	0.5920	0.7874
GW	84.8780%	0.6872	0.7908
GC	80.4878%	0.5869	0.7381
A	76.5854%	0.5193	0.7984
ACfs	76.0976%	0.5070	0.8001
AW	84.3902%	0.6726	0.7958
AC	80.9756%	0.6076	0.8415

Experiment 4 – 300 Training Instances – JRIP

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	84.4884%	0.6891	0.8595
OVCfs	84.4884%	0.6891	0.8595
OVW	84.8185%	0.696	0.8621
OVC	82.5083%	0.6496	0.8474
G	86.4686%	0.7287	0.8442
GCfs	83.4983%	0.6692	0.8566
GW	86.1386%	0.7218	0.8265
GC	85.4785%	0.7092	0.8681
A	84.4884%	0.6890	0.8574
ACfs	83.8248%	0.6758	0.8718
AW	86.1386%	0.7218	0.8265
AC	84.4884%	0.6892	0.8888

Experiment 4 – 300 Training Instances – PART

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	85.4785%	0.7094	0.8737
OVCfs	85.4785%	0.7094	0.8737
OVW	84.4884%	0.6895	0.8563
OVC	86.1386%	0.7227	0.8780
G	86.4686%	0.7287	0.8404
GCfs	86.4686%	0.7287	0.8404
GW	86.7987%	0.7351	0.8207
GC	83.1683%	0.6636	0.9013
A	84.4884%	0.6895	0.9043
ACfs	84.4884%	0.6897	0.9053
AW	86.7987%	0.7351	0.8207
AC	85.8086%	0.7156	0.8806

Experiment 4 – 300 Training Instances – J48

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	83.8284%	0.6762	0.8626
OVCfs	83.8284%	0.6762	0.8626
OVW	84.4884%	0.6895	0.8563
OVC	84.8185%	0.6962	0.8659
G	86.4686%	0.7287	0.8437
GCfs	83.1683%	0.6629	0.8868
GW	86.7087%	0.7351	0.8207
GC	83.1683%	0.6634	0.8752
A	82.8383%	0.6564	0.8954
ACfs	83.1683%	0.6629	0.8868
AW	86.7087%	0.7351	0.8207
AC	85.8086%	0.7156	0.8796

Experiment 4 – 400 Training Instances – JRIP

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	88.3375%	0.7664	0.8953
OVCfs	88.3375%	0.7664	0.8953
OVW	86.8486%	0.7361	0.8683
OVC	86.8486%	0.7361	0.8683
G	87.0968%	0.7413	0.8768
GCfs	87.3449%	0.7469	0.6291
GW	87.3449%	0.7467	0.8687
GC	85.1117%	0.7021	0.8704
A	86.3524%	0.7255	0.899
ACfs	88.3375%	0.7662	0.8963
AW	87.5931%	0.7517	0.8761
AC	86.1042%	0.7214	0.876

Experiment 4 – 400 Training Instances – PART

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	87.0968%	0.7404	0.847
OVCfs	87.0968%	0.7404	0.847
OVW	86.1042%	0.7210	0.8819
OVC	86.1042%	0.7210	0.8819
G	87.0968%	0.7418	0.8816
GCfs	85.8561%	0.7169	0.8816
GW	88.0893%	0.762	0.8775
GC	85.1117%	0.7011	0.8791
A	85.8561%	0.7169	0.9093
ACfs	85.8561%	0.7175	0.9046
AW	87.0968%	0.7418	0.8834
AC	85.3598%	0.706	0.9075

Experiment 4 – 400 Training Instances – J48

Attribute Set	Training Set		
	Accuracy	Kappa Statistic	AUC
OV	87.0968%	0.7404	0.8428
OVCfs	87.0968%	0.7404	0.8428
OVW	86.1042%	0.721	0.8642
OVC	86.1042%	0.721	0.8642
G	86.6005%	0.7321	0.8776
GCfs	87.0968%	0.7421	0.8765
GW	88.0893%	0.762	0.8769
GC	83.8710%	0.6756	0.8845
A	85.8561%	0.7171	0.9037
ACfs	86.3524%	0.7272	0.9046
AW	87.0968%	0.7418	0.8866
AC	86.1042%	0.7199	0.9033

Appendix C ID's of Test Case Result

This appendix discusses an implementation for the test case result ID that could be used to identify if two test case result (TCR) instances are for the same object of interest in a test case (TC). Each TCR would be identified by two numbers, the first being the Algorithm ID and the second being the object of interest id. This way the developer has a way to check that they are comparing two TCRs that identify the same object of interest in a TC. This is illustrated in Figure C 1 below.

Figure C 1 Explaining the TCR ID values

For example, object TCR11 is from Algorithm 1 and is the first object of interest identified in the TC. This is also described using the OCL language below.

Test Case

```

Algorithm.allInstances -> for all(a1,a2|
self.tcr -> select(t|t.tsr.algorithm=a1).id =
self.tcr -> select(t|t.tsr.algorithm=a2).id

```