ABSTRACT

USING GENETIC ALGORITHMS FOR
FEATURE SET SELECTION IN TEXT MINING

by Benjamin Charles Rogers

The rationale behind design decisions are often recorded in different project documentation. One way to extract this rationale is by using text mining. Text mining involves data mining over natural language documents. The performance of a text mining system depends on many factors, including the feature sets used. Exhaustive searching for optimal combinations of feature sets is rarely feasible, often leading researchers to make guesses as to which combinations to use.

A genetic algorithm is used to find optimal combinations of feature sets for binary rationale, the argumentation subset, the arguments-all subset, decisions, and alternatives. The genetic algorithm uses GATE, WEKA, and a pipeline that allows the automatic passing of information from one to the other. This pipeline is also useable in other text mining contexts. The genetic algorithm produced medium sized feature sets which tended to prefer unigrams and bigrams over 4-grams and 5-grams when compared to random selection.
USING GENETIC ALGORITHMS FOR
FEATURE SET SELECTION IN TEXT MINING

A Thesis

Submitted to the
Faculty of Miami University
In partial fulfilment of
The requirements for the degree of
Master of Science
Department of Computer Science

by
Benjamin Charles Rogers
Miami University
Oxford, Ohio
2013

Advisor____________________
Janet Burge

Reader____________________
Dhananjai Rao

Reader____________________
Michael Zmuda
# TABLE OF CONTENTS

1 **Introduction** ................................................. 1
   1.1 Background ............................................... 1
   1.2 Problem Description and Motivation ....................... 2
   1.3 Research Questions ....................................... 2
   1.4 Approach ................................................. 3
   1.5 Outline .................................................. 3

2 **Rationale Extraction from Existing Documents** ................. 4
   2.1 Design Rationale ......................................... 4
   2.2 Text Mining ............................................. 5
   2.3 Text Mining Design Rationale ............................ 6
   2.4 Feature Set Selection .................................... 7
   2.5 The Need for a Pipeline .................................. 7
   2.6 Genetic Algorithms ....................................... 8

3 **GATE-WEKA Pipeline Design** .................................. 9
   3.1 Importing GATE XML Files ................................. 9
   3.2 Generating Features ...................................... 10
   3.3 Preparing and Running the Corpus ....................... 11

4 **GATE-WEKA Pipeline Implementation** .......................... 11
   4.1 Importing GATE XML Files ................................. 11
   4.2 Generating Features ...................................... 12
      4.2.1 Feature Templates .................................... 12
      4.2.2 Algorithm ........................................... 12
   4.3 Generating Corpus Database ............................... 13
   4.4 Generating WEKA Supported ARFF Files .................. 14
   4.5 Running Filters on Folds .................................. 14
   4.6 Running Classifiers and Saving Results .................. 15

5 **GAFFS Design** .................................................. 15
   5.1 Using the Pipeline ........................................ 16
   5.2 The Genetic Algorithm .................................... 16

6 **GAFFS Implementation** ......................................... 16
   6.1 Using the Pipeline ........................................ 17
   6.2 The Genetic Algorithm .................................... 17

7 **GAFFS Experiments** ............................................. 18
   7.1 Experiment Design ........................................ 19
      7.1.1 Fitness Functions ..................................... 19
      7.1.2 Population Size and Number of Generations ........ 19
      7.1.3 Combination and Mutation Rates .................... 20
      7.1.4 Testing for Validity ................................ 20
   7.2 Data Sets Used ............................................ 21
      7.2.1 Selection ............................................. 21
      7.2.2 Annotation .......................................... 21
      7.2.3 Pre-processing in GATE ............................... 21
7.3 Features Used ......................................................... 22
7.4 Classifier ................................................................. 23
7.5 Expected Outcomes ..................................................... 23

8 Results and Evaluation .................................................. 23
8.1 GAFFS Experiment Results ........................................... 23
8.1.1 Results from the Design Rationale Superset GAFFS ............ 23
8.1.1.1 Feature Sets Produced ........................................ 23
8.1.1.2 Tests for Validity ............................................... 24
8.1.2 Results from the Design Rationale Subset GAFFS .............. 24
8.1.2.1 Feature Sets Produced ........................................ 24
8.1.2.2 Tests for Validity ............................................... 26
8.1.3 Comparison .......................................................... 26
8.1.4 Relationship between Results and Hypotheses ................. 31
8.2 Usability ................................................................. 32
8.2.1 Pipeline Usability .................................................. 32
8.2.2 GAFFS Usability .................................................... 32
8.3 Summary of Results ................................................... 33

9 Future Research .......................................................... 33
9.1 Expanded Experiments ............................................... 33
9.2 Using the Pipeline ..................................................... 33

10 Conclusions ............................................................. 34

Bibliography ...................................................................... 35

A Lists of Rationale Words ................................................ 37
A.1 List of Domain Rationale Words .................................... 37
A.2 List of General Rationale Words .................................... 39

B Example Files .................................................................. 40
B.1 Example XML File Generated by GATE ......................... 40
B.2 Example XML File Generated by the GATE-WEKA Pipeline .. 40
B.3 Example ARFF File ...................................................... 41
B.4 Example Vectorized ARFF File ..................................... 42

C Feature Sets .................................................................. 43
C.1 Binary Rationale 10 Best Feature Sets ............................... 43
C.2 Argumentation Subset 10 Best Feature Sets ...................... 45
C.3 Arguments-All 10 Best Feature Sets ................................ 49
C.4 Alternatives 10 Best Feature Sets ................................... 52
C.5 Decisions 10 Best Feature Sets ...................................... 56
C.6 Full Feature Set List .................................................... 59

D Pipeline Instructions ....................................................... 60
## LIST OF TABLES

8.1 Binary Rationale Feature Set ................................................................. 24  
8.2 Rationale Subset Feature Sets ............................................................... 25  
8.3 Total Feature Set ................................................................................... 27  
8.4 Feature Sets by Percent ........................................................................ 28  
8.5 Binary Rationale Results ....................................................................... 29  
8.6 Argumentation Subset Results ............................................................... 29  
8.7 Arguments-All Results ........................................................................ 30  
8.8 Alternatives Results ............................................................................ 30  
8.9 Decisions Results .................................................................................. 30
LIST OF FIGURES

1.1 Approach Flow ................................................................. 3
3.1 Pipeline ................................................................. 9
3.2 GATE Annotation Screenshot ........................................... 10
4.1 Example Annotations ..................................................... 13
5.1 GAFFS ......................................................... 16
6.1 Example Chromosome ................................................... 17
6.2 Combination and Mutation ........................................... 18
8.1 Results by Generation ................................................... 31
ACKNOWLEDGEMENTS

This thesis would have not been possible without the help and support of colleagues, professors, friends, and family. Dr. Valerie Cross convinced me that, no really, I could get in to Miami’s Masters of Computer Science program if I tried. Dr. Janet Burge took me on as an undergraduate researcher, and then as one of her graduate research assistants, helping me develop into a published researcher while providing guidance and support along the way. I want to thank my undergraduate research colleagues, Yetchen Qiao and James Gung, with whom I started the research that would develop into this thesis. I would also like to thank my graduate research colleague, Tanmay Mathur, for helping with related research and for asking the questions that made me go back and make sure I was documenting all the things I should. I would also like to thank my thesis committee, Dr. Dhananjai Rao and Dr. Michael Zmuda, who helped guide my decision making and made sure that Dr. Burge and I didn’t do anything that would invalidate the results. My parents, Glenn and Dora Rogers, gave me financial and moral support, as well as practice explaining what it is exactly that I’m doing to laypeople. My fiancée, Ariel Baker, gave me moral support and kept me on task while writing. The National Science Foundation funded my master’s degree through Dr. Burge’s Career Grant. I would also like to thank everyone that I forgot that helped me these past two years, I couldn’t have done it without your support.
Chapter 1 Introduction

Design rationale are the decisions made and reasons behind those decisions made during a design process. [Jarczyk, et al., 1992] Design rationale contained in natural language documents can be extracted in two main ways, manually or automatically. With automatic design rationale extraction, finding good feature sets for text mining can have a large impact on success. The feature sets found in text mining, such as the parts of speech of words or the length of sentences, are numerous and it is difficult to determine which ones to use. While it is sometimes possible to use all of the features when text mining, this can cause problems when noisy features, features that do not relate to whether a sentence contains rationale or not, are used and less than optimal results are returned. Using all of the features is also very expensive when considering the time and space requirements of a text mining algorithm.

Genetic algorithms are used to select optimal or near optimal feature sets when text mining various subsets of rationale. Genetic algorithms imitate natural selection, by having possible feature sets represented as individuals of a population. How well a feature set preforms when used in text mining determines that individual’s fitness, which, in turn, determines how likely that individual will “mate” and pass on its features to later, higher performing feature sets. The genetic algorithm constructed for these experiments is called GAFFS, or the Genetic Algorithm For Feature Set optimization.

1.1 Background

When designing software, engineers make many decisions regarding its design and development. These decisions, considered alternatives, and arguments for and against those decisions and alternatives compose the rationale of the software system. [Jarczyk, et al., 1992] This rationale is useful in any iterative development process and invaluable during software maintenance. The ability to look at the arguments for and against alternatives and the decisions made by the designers, allows the software maintainers to make more informed decisions when making changes to the software. Examples include making sure the system goals are maintained and not implementing rejected alternatives.

Despite its value, design rationale is seldom captured in its own documentation during the software engineering process. Creating design rationale documentation is time consuming and potentially expensive, consuming scarce resources during software development. However, rationale is often otherwise recorded by software engineers in other documents. Rationale contained in these documents, ranging from emails and bug reports to meeting transcripts and requirements documents, can often be extracted manually. Unfortunately, manual extraction of rationale from existing documents incurs the same problems as recording it explicitly in the first place.

One attempt at addressing the problems of extracting rationale from existing documents is to automate the extraction process using text mining. [Rogers, et al., 2012] Text mining uses models created by showing documents whose rationale has already been annotated, marked to indicate where in the documents the rationale are, to a classifier to learn over. These models then take annotated documents and make guesses as to the existence and location of rationale. Some manual extraction of rationale will still need to take place to create features for the text mining systems to learn over, but the number of documents that will have their rationale manually extracted will be a small fraction of the documents that will have their rationale automatically extracted. Current efforts are focusing on creating text mining systems that return an acceptable amount of rationale with acceptable accuracy.

Performance of text mining systems is improved by changing the features sets used for classification as well as by changing the classifiers themselves. The feature sets and
classifiers are often initially chosen using the researcher’s best guess, given the best current methods used in similar problems. The researcher then changes these feature sets and classifiers to search for better results, but it is often impossible for the researcher to check every combination of options to find the optimal solution for their text mining problem in a practical amount of time.

1.2 Problem Description and Motivation

Determining the optimal feature sets in a text mining application is crucial to its performance. Selecting sub-optimal text mining parameters can lead to poorly performing applications and wasted resources when the data must be fixed after processing or when an automated system isn't implemented because it cannot out-perform manual annotation.

Currently, determining the optimal feature sets is difficult. Testing all of the possible combinations is seldom feasible for text mining applications. There are an infinite number of features that can be learned over, but this paper will address only 512 of some of the most common ones. One method is guessing the optimal solution, given what a researcher knows about the performance of various combinations or individual choices on similar data. This method can lead to choosing more familiar options because the researcher knows more about them, discovering only local optima, and choosing bad options for the particular dataset used by the researcher.

A simple solution to the optimal feature set problem is to include all of the possible features in the feature set. Aside from being uninteresting, this solution also encounters the problem of taking a very large amount of time and space to run, initial experiments ran out of heap space in WEKA (the Waikato Environment for Knowledge Analysis, the data mining application used)[Hall, et al., 2009] and took over two days to do so. Running a filter on the feature set would not reduce the time and space to run significantly, as the filter would end up taking up the processing burden. Again, initial experiments ran out of heap space in WEKA, but only after six hours. The problem then becomes determining the optimal feature sets and classifiers in a set amount of time.

1.3 Research Questions

The goal of the work described in this thesis is to answer the following research questions:
1. What feature sets are chosen by a feature set size weighted genetic algorithm when extracting design rationale from text documents? A feature set size weighted genetic algorithm would be forced to choose the feature sets that gave the best results with as small a feature set as possible, hopefully giving examples of fast and accurate feature sets for rationale extraction from text documents.
2. Alternatively, what feature sets are chosen by a feature set size weighted genetic algorithm when extracting subsets of design rationale from text documents? The set of design rationale contains eight different types of design rationale: alternatives, answers, arguments, assumptions, decisions, procedures, questions, and requirements. The subsets, the argumentation subset (alternatives, arguments, assumptions, decisions, and requirements), the arguments-all subset (arguments, assumptions, and requirements), alternatives (only alternatives), and decisions (only decisions). The examination of the subsets allow insight into the differences surrounding them in a text mining context. A feature set size weighted genetic algorithm would be forced to choose the feature sets that gave the best results with as small a feature set as possible, hopefully giving examples of fast and accurate feature sets for design rationale subset extraction from text documents.
1.4 Approach

In order to run a genetic algorithm using the classifiers provided by WEKA and the annotation tools provided by the General Architecture for text engineering (GATE) [Cunningham, et al., 2002], text annotation software that adds feature annotations, such as parts of speech, to text; a GATE-WEKA pipeline was created that took the documents annotated in GATE and put them in a database, which was then accessed to create Attribute-Relation File Format (ARFF), WEKA input, files given a desired feature set. The files were k-folded for cross validation, filtered using information gain, and ran in WEKA. The results were then saved. The GATE-WEKA pipeline was then augmented so that a genetic algorithm, the Genetic Algorithm for Feature Set Optimization (GAFFS), could create ARFF files based on chromosomes consisting of feature sets and classifiers, and create new chromosomes based on the output of the WEKA tests run on those ARFF files. Experiments were then conducted to observe the optimized feature sets for the design rationale superset (alternatives, answers, arguments, assumptions, decisions, procedures, questions, and requirements) and the design rationale subsets: the argumentation subset (alternatives, arguments, assumptions, decisions, and requirements), the arguments-all subset (arguments, assumptions, and requirements), alternatives (only alternatives), and decisions (only decisions). The Process used for extracting rationale from existing documents is shown in figure 1.1.

1.5 Outline

The remaining chapters of this thesis are structured as follows: Chapter 2 discusses rationale extraction from existing documents, Chapter 3 sets out the GATE-WEKA Pipeline Design, Chapter 4 discusses the implementation of the GATE-WEKA Pipeline, Chapter 5 describes the design of the Genetic Algorithm for Feature Set Optimization (GAFFS).
Chapter 6 discusses GAFSS implementation, Chapter 7 describes the experiments conducted using GAFSS, Chapter 8 discusses the results and evaluation of the experiments, and Chapter 9 sets out future research.

Chapter 2 Rationale Extraction from Existing Documents

To address the aforementioned research questions, we must first more formally discuss what design rationale is, what text mining is, how it is currently being applied to the problem of automatic rationale detection in natural language documents, what feature selection is and why it is so important, the need for a new GATE-WEKA pipeline, and genetic algorithms and why they are useful.

2.1 Design Rationale

When designing a system, decisions must be made. Every part of that system, large and small, will be the result of a decision making process. The reasoning behind these decisions are their rationale. [Jarczyk, et al., 1992] Each decision should have alternatives. The decision and alternatives associated with it will have associated requirements that must be met, assumptions that are made, and/or procedures that must be followed. Each decision and alternative will have arguments for and against it. These arguments are augmented with questions relating to the arguments, as well as answers to those questions.

Not all design decisions will have comparable amounts of rationale. Decisions may be well thought out, with many alternatives being considered and compelling arguments for and against those decisions, or they may be rushed with no considered alternatives and little to no argumentation. No matter how thorough or sparse the reasons behind the design decisions, this is the system’s design rationale.

Design rationale has some obvious uses during an iterative design process. As arguments are fleshed out, and as old requirements change or as new requirements are identified, alternatives that were considered can be switched to. The same can be said for the implementation. As new issues or requirements arise, previously considered alternatives may become more attractive. Design rationale is, however, probably the most useful during a system’s maintenance phase. Often, maintenance is taken up by people who did not design or implement the software, and without documentation or knowledge otherwise, can alter the system without considering the design rationale for the system. This can lead to the unnecessary creation of bugs as important arguments or procedures are forgotten, breaking requirements as some of the requirements may also be forgotten, or missing highly favourable alternatives, such as when an alternative that was only not chosen because of hardware issues is not considered in the maintenance phase when the hardware issues may no longer be a problem because of advancing technology.

Even with the promise of much smoother maintenance, few in industry have taken documenting design rationale seriously. One reason is that the usefulness of design rationale remains relatively unproven in industry, and taking the risk of spending time and money on something that may not work is not very palatable for many companies. This is a chicken and egg problem, as companies will only consider using it if it is proven to work, but if no one will use it, then it will never be proven to work. The main reason, however, is that companies have neither the time nor the resources for the large amount of documentation that recording the system’s design rationale would require. The software industry tends to have the problem of projects not meeting deadlines and going over budget already, and adding the additional burden of design rationale documentation would not necessarily help.

While documenting design rationale explicitly has not taken off, companies are
documenting design rationale in other places, such as in documents such as emails, bug reports, meeting transcripts, and requirements documents. Although often incomplete, the design rationale contained in these documents can still prove useful if it is extracted. Manual recovery of design rationale from existing documentation is possible, but in often runs into the same problems that prevented it from being recorded explicitly in the first place. It takes a lot of time and energy to extract, and usually don’t contain the complete original design rationale.

Automatic rationale extraction from existing documents is currently being explored. [Rogers, et al., 2012] While it will not be able to solve the completeness issue, automatic rationale extraction promises to do an acceptable job of identifying design rationale for a fraction of the time cost of manual extraction.

2.2 Text Mining

Text mining is the intersection of machine learning and natural language processing (NLP). [Hobbs, et al., 1982] Machine learning describes the programs and algorithms that recognize patterns and predict behaviour given input data. For example, a supervised machine learning algorithm, such as a picture-to-text converter, would take in discrete input data. This data would have attributes describing the data and a category that that data represented. To continue this example, the attributes would be the value of pixels of a 100x100 pixel picture of the letter ‘A’ and the category would be the letter ‘A’. The algorithm would then try to build a model to match attribute values to categories. We call the data used to generate the model the training set. Continuing our example again, we could have 100 different 100x100 pixel examples of all of the characters that we want our algorithm to learn over.

The model generated will then take new discrete input of the attribute values and output the guessed category. The model generated is seldom 100% accurate, and can often be very inaccurate. To test the accuracy of the model, a test set of data is used. This set consists of attribute values / category pairs, just like the training set. We compare the guessed category to the actual category to obtain the accuracy of the model.

To see if a machine learning algorithm is useful for a particular problem, k-fold cross validation is often used. This means that the data from the data pool is split into k groups. These groups are then used as k different test sets, with the data not in a specific test set in that test sets’ training set. K models are generated by using each of the training set and accuracies are produced by using the corresponding test sets. The accuracies are then averaged, showing the average accuracy the model is likely to produce.

There are many things that can affect the accuracy of the model, but none as large as poor selection of training data. The training data that is used should be pulled randomly and uniformly from the data set, as having a overrepresentation of some attributes and under-representation of others can lead to data over-fitting. That is, the model works very well for the data set used to train it, but it performs poorly with the test set or in production because the training set was not representative of the whole population.

Natural language processing describes the discipline of giving machines the ability to gain useful information from text created for human consumption. For example, automatically translating text between two languages. Natural language processing applications often use parts of machine learning to help them achieve their goals.

Data given to machine learning algorithms must be discrete and explicit, and cannot, alone, handle the context, subtlety, and complexity of human language. Natural language processing (NLP) takes the lead in manipulating the data into something that machine learning algorithms can use. This is no simple task, and much research must still be done before a machine can understand human language like humans can. Nevertheless,
advancements in NLP have created machines that can analyze and comprehend natural language text at levels that are useful for a variety of tasks.

NLP tools like the General Architecture for Text Engineering (GATE) [Cunningham, et al., 2002] allow the importing of a corpus (collection) of natural language documents, the text of which can then be annotated, so that all of the independent sentences can be marked as such, as well as other important information, such as what text contains design rationale. The tools often allow for automatic sentence detection/splitting and parts of speech tagging. These automated tools are based on models generated by machine learning. [Miller, 1995] Once annotation is complete, the corpus can be exported and discretized based on those annotations, allowing use by machine learning tools.

[He, et al., 2009] and [Li, et al., 2009] address an issue that is prevalent in text mining, but not exclusive to it: imbalanced data. Imbalanced data proves difficult to deal with in machine learning, as the more represented category will tend to dominate in the model, making the model much more likely to guess that category, as opposed to the lesser represented category, which is often misclassified. This becomes a large problem when trying to identify design rationale in text documents, as design rationale tends to be sparse in most documents and therefore becomes the lesser represented category in a set of imbalanced data.

2.3 Text Mining Design Rationale

The problem of text mining design rationale has some properties that make it unique when compared to some more general text mining problems. Specifically, design rationale is often structured, with each design decision having alternatives that were considered, as well as requirements, assumptions, and procedures to be followed. Each alternative that was considered and the decision that was made had arguments for and against and questions and answers relating to those arguments. Ideally, not only should the sentences containing the design rationale be extracted, but its structure, which is not always clear when recorded in text, should be extracted as well. Design rationale can be found in relatively unstructured informal documents, such as inter-developer emails, structured informal documents, such as bug reports, and structured formal documents, such as requirements documentation. The amount of design rationale contained in a specific document can range from very sparse, to very full depending on the document’s intent.

[Liang, et al., 2011] focused on extracting design rationale from patent documents. A three-tiered model was used to capture issues, design solutions, and artefacts. First, artefacts were identified using a modified PageRank [Brin and Page, 1998] algorithm on frequently appearing words. Next, issue summarization is performed using issue language patterns as part of manifold ranking. Finally, reason language patterns are used to identify reason sentences which are then paired with the remaining solution sentences to create reason-solution pairs. The achieved a 0.185 F-measure for artefact identification, a 0.520 F-measure for issue summarization, and a 0.562 F-measure for reason-solution extraction.

The next papers refer to arguments in legal texts, which tend to have a higher density of rationale, (about 50% for legal texts [Mochales Palau and Moens, 2009] and 11% for bug reports. [Rogers, et al., 2013]). [Prakken, et al., 2003] explored the formalization of rationale arguments in legal texts, using common legal argumentation structures as a base. [Moens, et al., 2007] explored the automatic extraction of rationale arguments from legal texts, experimenting with a number of different feature sets. They found that the best performing feature sets were unigrams, bigrams, word couples, and combinations including the three, with F-measures ranging from 0.704 to 0.738. [Mochales Palau and Moens, 2009] built on the previous rationale argumentation works by adding a context free grammar to automatically structure argumentation text, obtaining around 60% accuracy.
[Rogers, et al., 2012] explored using ontologies and parts of speech with a large number of different classifiers to extract design rationale from Chrome bug reports. While they showed that some rationale could be extracted using these features, the performance by even the best classifiers could not match manual extraction.

2.4 Feature Set Selection

The information contained in natural language documents can only be utilized by machine learning models if it is separated into discrete pieces of data. The main task of natural language processing is figuring out how to discretize the information contained in the document, preserving context and other meta-information in a way that is useful to the models. These features, such as n-grams, parts of speech, and sentence length, can then be passed on to machine learning classifiers. While generating many different features for a data set or document is a good first step, problems start arising when the whole group of features is sent to classifiers. Features that are irrelevant or noisy can confuse classifiers that will try to fit to them and lead to sub-optimal performance then if only a subset of the features were used.

Filtering is a process where a filter algorithm picks out features and attributes that it thinks are important and throws the rest away. Wrappers are similar, but they rely on the classifier to determine what features and attributes are important. The problem that arises with filters and wrappers is that they sometimes, if not often, take more time than the classifiers they are filtering or wrapping to run, which can end up being expensive if the classifier takes a very long time to run.

Fortunately, the time it takes wrappers and filters to run is a function of how many attributes they have to look at to determine the most relevant ones. If irrelevant features can be removed from the set sent to the filter or wrapper and then to the classifier, speedup will be achieved with little to no loss in performance.

[Wasikowski, et al. 2010] addresses the problems caused by imbalanced data by using feature selection among other methods. On average, feature selection produced the most improvement when compared to selection and algorithm solutions.

[Salcedo-Sanz, et al., 2002] used a genetic algorithm as part of a wrapper in order to provide timely accuracy, however, it is limited when applied to our large problem, where it is still too slow because it learns over the super-set of features.

[Tan, 2007] addresses the problem of feature selection by using a genetic algorithm and a hybrid filter. When used on the 20 Newsgroup and Reuters-21578 datasets, small to modest gains are observed over information gain filtering. However, the algorithms’ applicability to design rationale identification was not explored.

[Oliveria, et al., 2010] addresses the problem of feature selection again by using a number of genetic algorithms this time on six datasets. These experiments showed that the genetic algorithms produced small to moderate gains over classification without genetic algorithms. However, these datasets were related to software man/hour cost estimation, and did not cover design rationale identification.

2.5 The Need for a Pipeline

Because WEKA [Hall, et al., 2009] and GATE [Cunningham, et al., 2002] are both commonly used text mining tools for different stages of the text mining process (GATE for initial annotation and WEKA for classification), it would make sense that pipeline tools would be created to automate the flow and conversion of data between the two. The two pipelines that are published and publicly available are WSDGate [Joshi, et al., 2006] and
KEA [Witten, et al., 1999] GATE plugin. Both tools are no longer supported and do not work with current versions of GATE and WEKA. For any application that was to attempt the automatic feature set selection and running of tests with those feature sets would need such a pipeline to function.

2.6 Genetic Algorithms

Genetic algorithms are optimization algorithms that model themselves after the natural selection process found in biology. [Mitchell, 1998] Genetic algorithms encode the data trying to be optimized as genes in chromosomes. These chromosomes belong to individuals who have a fitness, based on their chromosomes, which helps determine how likely it is that they will mate and produce offspring with another individual. Consecutive groups of individuals are called generations, and each consecutive generation is made up of the children of the previous generation. The genetic algorithm stops after either a certain number of generations is reached or the best or average individual fitness fails to increase significantly.

Data is encoded as genes. These encodings can be simple binary encodings, such as the inclusion or exclusion of an item in a set, or they can be other more complex encodings, such as individual commands for a specific task. Individual genes are then collected into a set of chromosomes, such that the genes present are a complete set of data representative of a possible solution. These chromosomes, a single possible solution, make up a single individual.

Groups of individuals make up a generation. Each individual, or possible solution, should be compared and ranked in relation to the other individuals in their generation. This will show which of the current individuals are most likely closest to an optimal solution. The way individuals are ranked is based on the genetic algorithm’s fitness function. The fitness function is what the genetic algorithm is trying to optimize for. For example, in a genetic algorithm that is trying to automate political redistricting, the fitness functions might consider how homogeneous and compact the proposed districts are and award a higher fitness for highly homogeneous and compact districts, or it might consider how many “safe” districts a particular political party can gain and award a higher fitness for larger gains.

Once every individual has a fitness assigned to them, the genetic algorithm decides which individuals should mate and produce offspring. There are many different methods of selecting which individuals to mate, but they all promote the mating of more fit individuals over less fit ones. When two individuals are chosen to mate with each other, they will produce two offspring. To produce these offspring, the two parents’ genes are copied over into new chromosomes, with the parents switching which chromosomes they are coping to at predetermined crossover points. This allows for the evolution of the offspring, producing a better or worse fitness. The genes in the new chromosomes also have some chance to mutate, leading to further variance.

Because it is likely that the most fit individuals will produce a new generation that has a higher average fitness than the parent generation, genetic algorithms can be used to optimize solutions for the problems addressed in the fitness function. Genetic algorithms stop creating new generations either after a certain number of generations, or after the average or best fitness does not improve significantly between two consecutive generations. The best fit individual of the last generation is usually considered the solution found by the genetic algorithm and is often close to the real optimal solution.

Genetic algorithms are used primarily as fast optimizers when the real optimal solution is not required, but a close solution is, and when it would take too long to run a different, more accurate or complete optimization algorithm. [Mitchell, 1998]
Chapter 3 GATE-WEKA Pipeline Design

The design of the GATE-WEKA pipeline was a critical building block for the experiments. It had to be flexible enough to allow for both machine input and control (for GAFFS) and human input and control, for general use by other researchers, as a replacement for WSDGate [Joshi, et al., 2006] and KEA [Witten, et al., 1999]. The GATE-WEKA pipeline starts with exported GATE XML files and automatically runs WEKA tests as specified by the user and can even generate new GATE XML files based on WEKA results.

Figure 3.1 shows the flow of the pipeline. GATE XML files are read in, and their annotations are extracted and placed in a CSV database. An Attribute-Relation File Format (ARFF) file is created by pulling feature annotations from the CSV database, given a user selected feature set. The ARFF file is then vectorized using WEKA, and the vectorized ARFF file is split into testing and training folds. The training files then undergo info-gain filtering and the test files are normalized against them. The training-test file pairs are then classified using WEKA, generating a binary model that can be used later, and results containing the annotation guesses used to create new GATE XML files.

![Fig. 3.1. The GATE-WEKA Pipeline](image)

3.1 Importing GATE XML Files

In GATE, [Cunningham, et al., 2002], document are annotated with rationale, sentences, and tokens. Annotation means that a selection of text is declared to fall in a certain category. For example an assumption will be selected and annotated with the label “Assumption”. Figure 3.2 shows an example of this.
After documents are annotated in GATE, they can be exported as XML files. These XML files contain a section for the text of the document, with node labels (used in annotations) dispersed throughout the text, and sections for annotation sets, which contain annotation information, such as start and stop nodes and annotation types and IDs, as well as a number of features, each with their own types and strings.

The end goal for the data the pipeline reads in from the XML files is to obtain lists of sentence and token annotations, as well as all of the data and features they contain, and match tokens to sentences so sentence features may be obtained. The design decisions revolve around what parts of the XML to read and keep, and how the tokens and sentences are going to be stored, and their interaction.

3.2 Generating Features

Because of the nature of these experiments, and the nature of text mining in general, the robustness and extensibility of generating features and feature sets was one of the primary concerns for the pipeline. Creating hard-coded features that could be selected was not an option, as while it may have been sufficient for these experiments, there would be no ability for researchers to add their own features, severely limiting its extensibility. The algorithm that generated these features also should be fast.

In order to enable extensibility, no features were hard-coded, instead, a simple feature template language was developed to allow almost any feature to be created by a researcher and then parsed and used by the pipeline. Feature Templates should allow different attributes to be recorded when various conditions are met.

Because with a large document, the most computing time is taken up matching tokens to sentences, it is only done once. First, all of the feature templates are read into a list. Next, the sentence and token annotations are sorted. Features are then taken for each token in each sentence. Because both sentence and token annotations are sorted, each token is looked at.
only once, and each sentence is looked at only the number of times equal to the number of tokens it has.

3.3 Preparing and Running the Corpus

A corpus is a collection of writings, in this case, of bug reports. To prevent having to read in and re-find sentence features every time we want to run a new test, a CSV (comma separated volume) database is created to hold the sentences, their features, and their rationale presence.

After the Corpus Database is generated, the WEKA supported Attribute-Relation File Format (ARFF) files can be created. A string ARFF file is created by concatenating the feature strings, specified in the feature set variable, for each sentence, and then running WEKA's StringToWordVector program to get the vectorized and runnable ARFF file.

The newly vectorized ARFF file is then copied into training and test files, using the fold number of each sentence in the corpus database to determine into which test set the sentence would be included and which training set the sentence would be excluded. This is important so that we do not filter folds based on data that they do not have.

Each training set would then have a simple info-gain filter that would select the 1000 most relevant learning attributes from the training set and create a new ARFF file with only the 1000 most relevant training attributes included. The test files would then have to have their attributes trimmed to match those of their corresponding training files for classification. A filter would also be run on the full corpus ARFF file for later model generation.

A user specified classifier is then run on all of the training and test sets. The output of each run is then combined into one large results file using the corpus database to find the original sentence order by the fold numbers, with the group statistics presented at the bottom.

The same user specified classifier is then run on the full corpus ARFF file, the model of which is then saved.

The predictions generated by the large results file are then compared to the corpus database to get the sentence numbers of the positive predictions. The new generated rationale annotations are then added to the XML file.

Chapter 4 GATE-WEKA Pipeline Implementation

The implementation of the GATE-WEKA Pipeline was an integral part of running the experiments, a good implementation would allow for the flexibility required of research applications, allowing changes without major coding overhauls or re-designs, a good implementation would also have a substantial impact on the running time of the experiments, given that the genetic algorithm would require hundreds if not thousands of runs through the pipeline, a small speed-up or slow-down would have a magnified impact. The GATE-WEKA pipeline starts with exported GATE XML files and automatically runs WEKA tests as specified by the user and can generate new GATE XML files based on WEKA results.

4.1 Importing GATE XML Files

Because there already is a natural data structure for annotations and features given in the XML, it made sense to create Annotation objects, each with start and end nodes, IDs, and types, and Feature objects, each with types and values. It was also useful to create a method to see if two Annotation objects overlap, allowing for checking if a specific token is contained in a specific sentence. It was also considered that token Annotation objects might to point to the sentence that they are in, and for sentence Annotation objects to point to the
tokens that they contain. However, this added complexity was unnecessary given the method chosen for generating features.

4.2 Generating Features

The combination of the extensibility allowed by the feature templates and the relative speed of the algorithm that uses them allows feature generation to be both robust and timely.

4.2.1 Feature Templates

Feature Templates were defined as single line text documents with the following format:

\[ \text{name} \{\text{feature to be recorded}\} : \{\text{condition}\} \]

For example, to get the text of interjections, the feature template would be:

interjections \{string\} : \{category=UH\}

This says that the “interjections” feature is the value of the string feature of the token if that token has a category feature with a value of “UH”. Conditions can be AND'ed together with '!', for example, to get only prepositions and subordinating conjunctions that are also words:

prepsAndSubConj \{string\} : \{category=IN\&kind=word\}

Conditions can be OR'ed together with '|', for example, to get all of the adverbs:

adverbs \{string\} : \{category=R{B|RBR|RBS}\}

's and '|'s can be used together, but '|'s will always have a higher order of precedence. Conditions can be existential (equivalent to . * in regular expressions) by not specifying anything after the '='. For example, all of the parts of speech:

pos \{category\} : \{category=\}

Features to be recorded can be concatenated with '+', for example, to get all of the unigrams with their parts of speech as prefixes:

pos_unigram \{category+string\} : \{category=\}

N-grams are features that are concatenated together to create a longer unique feature, for example, “for all” is then 2-gram for “for” and “all”. They can be constructed by placing '(n)' at the start of the name, for example, bigrams:

\( (2)\text{bigram} \{\text{string}\} : \{\text{string}=\} \)

Finally, the number of times a condition is satisfied in a sentence can be counted by adding 'C_' at the start of the name and 'COUNT' as the feature to be recorded, for example, sentence length:

\( C_{\text{sentLength}} \{\text{COUNT}\} : \{\text{category=\} \}

4.2.2 Algorithm

First, all of the feature templates parsed are read into a list, because of the simple nature of the feature templates, parsing a feature template takes in linear time based on the number of terms in the template. (1)

\[ F \times \overline{F_{\text{size}}} \]

Where \( F \) is the number of feature templates and \( \overline{F_{\text{size}}} \) is the average number of terms in a feature template. Next, the sentence annotations are sorted in ascending order according to
their start node taking $S \log S$ time, where $S$ is the number of sentence annotations. Token annotations are then also sorted in ascending order according to their start nodes, taking $T \log T$ time, where $T$ is the number of token annotations. For each sentence, each token in that sentence is found. Because both sentence and token annotations are sorted, each token is looked at only once, and each sentence is looked at only the number of times equal to the number of tokens it has, so that the number of sentence lookups and token lookups are equal. So in total, it takes the amount of time specified in (2).

$$O(F \times \frac{\overline{F_{size}}}{\overline{F_{size}}} + S \log S + T \log T)$$

$$O(F \times \frac{\overline{F_{size}}}{\overline{F_{size}}} + 2(T \log T))$$

$$O(T \log T)$$

(2)

4.3 Generating Corpus Database

After all of the sentences have pointers to their tokens, the features can be read. In each sentence, each token is fed to the feature templates, and the output is recorded as a string of outputs for each feature for each sentence and takes the amount of time specified in (3).

$$O(T \times F \times \overline{F_{size}})$$

(3)

Where $T$ is the number of token annotations, $F$ is the number of feature templates, and $\overline{F_{size}}$ is the average number of terms in a feature template. These feature strings are then used to create the closest sentences. Sentences are then randomly assigned to folds, the default being 10 folds of as close to 1/10 of the sentences as possible. This output is then saved in a CSV (comma separated volume) database. This database contains a table of sentences. Each row contains the name of the XML file that the sentence came from, the sentence ID, the output of feature generation for each feature (including closest sentences as separate features), the type of rationale that is or is not present, and the fold number. Some example annotations taken from sentence 6330 from 24641.html are in figure 4.1.

unigrams = Issue 24641 - chromium - [ compatibility ] IPv 6 addresses mess up proxy autoconfig scripts - Project Hosting on Google Code

pos = NN CD : NN : NN NN NN NNP CD NNS NN RB NN NN NNS : NNP NNP IN NNP NNP
domain = Issue up on

C_sentLength = 44

(2)nouns = compatibility compatibility] ]mess messproxy proxyHosting HostingGoogle GoogleCode

nnp = IPv Project Hosting Google Code

Fig. 4.1. Example annotations
4.4 Generating WEKA Supported ARFF Files and Folds

To create WEKA supported ARFF files from the database. A string ARFF file is created by first setting the name as the feature set, then by concatenating the feature strings in the columns matched by the feature set, for each sentence, and then running WEKA's StringToWordVector program to get the vectorized and runnable ARFF file. See appendices B.3 and B.4 for examples.

The newly vectorized ARFF file is then copied into training and test files, copying over everything above the "@data" line exactly and then using the fold number of each sentence in the corpus database to determine into which test set the sentence would be included and which training set the sentence would be excluded.

4.5 Running Filters on Folds

Each training set then goes through simplified information-gain filtering that selects the 1000 most relevant learning attributes from the training set and create a new ARFF file, with only the 1000 most relevant training attributes included. In initial tests, where WEKA’s info-gain filtering was used, it became obvious that filtering was taking up much more time than classification, and was increasing the length of time necessary to run experiments. Information gain filtering, or info-gain, is a simple filter that tries to measure the information entropy of attributes based on the number of times it shows up in rationale and non-rationale containing sentences. (Technically, information gain measures the change in information entropy between the lack of, or knowledge of, the presence of an attribute) Information entropy can be estimated by the absolute value of the difference of the number of times an attribute is included in a rationale sentence and the number of times an attribute is included in a non-rationale sentence divided by the total number of occurrences of that attribute, shown in (4).

\[
\frac{|A_{rat} - A_{non-rat}|}{A_{total}} \quad (4)
\]

Where \(A_{rat}\) is the number of times an attribute is found in a rationale containing sentence, \(A_{non-rat}\) is the number of times an attribute is found in a non-rationale containing sentence, and \(A_{total}\) is the number of times an attribute is found in any sentence. This means that attributes that show up much more in either rationale or non-rationale sentences have a lower entropy. This value does not change for a specific word in a specific feature when the feature sets change, it was suggested that all of the possible features be pre-filtered for each fold, producing a large up-front time cost for filtering, but significantly reducing the over-all time taken filtering when running multiple tests (such as running a genetic algorithm).

For each feature and each fold, a list of the attributes and their info-gain score were generated and then sorted. Each list was then saved as a text file, where each line contained an attribute and its info-gain score where the best scoring attribute was recorded at the beginning of the document and the worst scoring attribute was recorded at the end. When the training files were ready to be filtered, the first lines of all of the info-gain files would be read in for each relevant feature for the corresponding fold, and the attributes with the best info-gain scores would be read in line by line until all of the attributes were read in or until 1000 attributes were read in, whichever came first.

The training files would then have to map the new attributes to the old attributes, and replacing their values in each of the sentence instances in the ARFF file. The test files would
then have to have their attributes trimmed to match those of their corresponding training files for classification.

4.6 Running Classifiers and Saving Results

A user specified classifier is then run on all of the training and test sets using WEKA. The output of each run is then combined into one large results file using the corpus database to find the original sentence order by the fold numbers, with the recall, precision, F-measure, true positives, false positives, true negatives, and false negatives presented at the bottom.

The same user specified classifier is then run on the full corpus ARFF file, the model of which is then saved in a WEKA binary format.

The predictions generated by the large results file are then compared to the corpus database to get the sentence numbers of the positive predictions. The old XML files are then read in and copied over, excepting the rationale annotations. The new generated rationale annotations are then added to the copied over XML file.

Chapter 5 GAFFS Design

The design of GAFFS was important as it effects the validity of the results, as well as the speed of which those results can be obtained and the adaptability of experiments. With no currently available existing genetic algorithm optimizing feature sets using a GATE-WEKA pipeline, non-obvious decisions about the design had to be carefully considered.

Figure 5.1 shows GAFFS. GAFFS operates like the pipeline with notable exceptions. First, the CSV database is not only used to generate ARFF Files, but also to generate Info-gain files. These files are used during the external info-gain process to save time when filtering. This was suggested by Dr. Michael Zmuda after the infogain implementation used in the pipeline took too long to be practical. Feature Sets are determined by chromosomes, which are initially set randomly. The feature sets and CSV database create ARFF files like the pipeline, and are vectorized by WEKA, and split into training and test folds like the pipeline as well. The training folds are filtered using the external info-gain filtering, and the test files are normalized like the pipeline. Classification is also the same as in the pipeline, but no model is produced. If classification was run for the final generation, the results are sorted by the highest f-measure and GAFFS quits, otherwise, the results are sorted by the highest f-measure and the fitness is evaluated, based on the f-measure and size of the feature set. The finesses are used to choose mates for the chromosomes, and crossover and mutation occur, after which, the cycle starts over again.
5.1 Using the Pipeline

The GATE-WEKA pipeline discussed in chapters 3 and 4 was modified to become the genetic algorithm. Chromosomes were created after the corpus database was created and were used to generate feature sets used for testing and training. The large results document generated by each chromosome was then used as the basis of the fitness function, which would then generate new chromosomes for more tests to be run. The number of chromosomes and generations are specified beforehand.

5.2 The Genetic Algorithm

Chromosomes consist of the binary inclusion of all of the possible features. They are initially generated randomly with a uniform distribution.

After each chromosome is given a fitness (described in Chapter 7), roulette wheel selection is used, where each chromosome is given a slice of the wheel proportional to its fitness, modified by the Boltzmann Selection process. [de la Maza and Tidor, 1993]

A new generation of chromosomes is created by copying the old chromosomes, and applying combination, switching the copied parent at each cross-over point, and then mutating (flipping) a small number of genes (feature binary inclusion).

Chapter 6 GAFFS Implementation

The implementation of GAFFS was important as it effects speed of the results can be obtained and the adaptability of experiments.
6.1 Using the Pipeline

The GATE-WEKA pipeline was modified to become GAFFS. The genetic algorithm starts after the sentence database and info-gain files are created with the creation of chromosomes, theses chromosomes contain the information necessary for filtering and classification of the folds. The large results document generated by the classification was then used as the basis of the fitness function, which would then generate new chromosomes for more tests to be run. The number of chromosomes and generations were specified beforehand.

6.2 The Genetic Algorithm

Chromosomes are data structures that hold information about binary inclusion of all possible features, creating feature sets. They are initially generated randomly, with a uniform distribution. After the chromosomes are used in ARFF generation, they are saved as binary files, and loaded back in at the end of classification, for combination and mutation. An example is shown in figure 6.1.

Fig. 6.1. Chromosomes Example

After each chromosome is loaded in after classification, each chromosome is given a fitness based off a combination of how many of the total possible features were used and what the chromosomes’ F-measure was. Then, Boltzmann Section is used to select pairs of parents. This means that roulette wheel selection is used, where each chromosome is given a slice of the wheel proportional to its fitness, but the proportion is also modified so that the beginning generations allow less fit individuals a greater chance of producing offspring, preserving some initial variance.

For each pair of parents selected, two new chromosomes are created by copying the old chromosomes, and applying combination, switching the copied parent at each cross-over point, and then mutating (flipping) a small number of genes (feature binary inclusion). Figure 6.2 gives an example.
Chapter 7 GAFFS Experiments

Experiments were designed to answer the two research questions:

1. What feature sets and classifiers are chosen by a feature set size weighted genetic algorithm when extracting design rationale from text documents?
2. What feature sets and classifiers are chosen by a feature set size weighted genetic algorithm when extracting design rationale subsets from text documents?

The two research questions can be answered by running a genetic algorithm that optimizes feature sets for text mining five times, changing the set of design rationale it learns over each time, first learning over the design rationale superset (alternatives, answers, arguments, assumptions, decisions, procedures, questions, and requirements) and then, the argumentation subset (alternatives, arguments, assumptions, decisions, and requirements), then the arguments-all subset (arguments, assumptions, and requirements), then the alternatives (only alternatives), and then decisions (only decisions).

There are two stages during a WEKA test that take the most amount of time, filtering and classification. Because information gain filtering was used, all of the individual features could be filtered beforehand, and the resulting attribute-value pairs could be used in for generating filtered combinations of those features. In previous experimentation, Naive Bayes Multimodal had the best result/time trade-off, so it was used as the classifier. The fitness function penalized chromosomes that use large feature-sets, trying to find the best feature-set combinations that exclude features that have little positive impact on precision and recall.
7.1 Experiment Design

For experiments involving genetic algorithms, several decisions need to be made about the structure and parameters of the genetic algorithms, such as population size, number of generations, combination and mutation rate, and the fitness function.

7.1.1 Fitness Functions

The fitness function determines how “fit” a chromosome is, that is, how likely that chromosome will produce offspring for the next generation. The fitness functions for both the feature-set size weighted and non-feature-set size weighted experiments use the F-measure generated by 10-fold cross validation by running each chromosome in WEKA. The F-measure was multiplied by one minus the fraction of the number of features in the feature set and the number of possible features in that set. Fitness is calculated by (5).

\[
fitness = F_{\text{measure}} \times (1 - \frac{\text{Set_{num}}}{\text{Set_{possible}}})
\]

(5)

Where \( \text{Set_{num}} \) is the number of features in the feature set and \( \text{Set_{possible}} \) is the number of possible features in that set. The resulting fitness would be returned.

The fitness function is then transformed according to Boltzmann Selection before using roulette wheel selection to get the next generation. Boltzmann Selection is used to preserve diversity in early generations, but to force convergence in later generations. The post-Boltzmann Selection fitness is equal to the fitness divided by the quotient of the generation number and the total number of generations, all over the average of the finesses divided by the quotients of the generation number and the total number of generations. Boltzmann Selection is shown in (6).

\[
fitness_{\text{Boltzmann Selection}} = \frac{\frac{fitness}{G_{\text{current}}}}{\frac{fitness}{G_{\text{total}}}}
\]

(6)

Where \( G_{\text{current}} \) is the current generation number and \( G_{\text{total}} \) is the total number of generations. [de la Maza and Tidor, 1993]

7.1.2 Population Size and Number of Generations

To make sure that all of the features had a chance to be in a set and have an impact to that set, the population size was set to 100.

There are usually two ways to decide the number of generations a genetic algorithm will have, iterate until there is little to no improvement of the best fitness, or choose the number of generations beforehand. [Mitchell, 1996] Because Boltzmann selection is being used, the number of generations is needed beforehand. Two things dictate how many generations should be run, the number of generations it takes for little to no increase in the fitness between generations and the time available for experiments. In preliminary trials,
small populations (20-30) with a small number of generations (15-20) reliably produce F-measures between 0.7 and 0.8, higher than the inter-annotator mean and median (0.583 and 0.67 respectively), close to the inter-annotator maximum (0.84), and better than the best feature set/combination pairs explored in our previous work (0.677). Inter-annotator agreement was calculated by having two researchers annotate the design rationale in a document and then comparing their annotations to obtain a human baseline for recall and precision. [Rogers, et al., 2013] Given this, optimality should not suffer dramatically if a relatively low number of generations is used.

With an estimated time for each chromosome to return a fitness function of 20 minutes and the population size of 100, each generation should take no more than 2000 minutes, or 32 hours and 40 minutes to run. With 20 generations, each experiment should take no more than 28 days to run.

### 7.1.3 Combination and Mutation Rates

Because of the small number of generations, the combination and mutation rates should be higher than average, in order to promote greater variance. Combination describes the process of taking two parent chromosomes and diving their genes between two child chromosomes. Every gene has a chance of being selected as a crossover point candidate, referred to here as the crossover candidate rate. This is different than some other genetic algorithms that have a set number of crossover candidates for each chromosome. [Mitchell, 1996] Doing it this way removes location bias from combination. Every crossover point candidate has a chance of being selected as a crossover point, referred to here as the crossover rate. After the combination of the two parent chromosomes, switching at the crossover points, each gene on each child chromosome is then subject to random mutation.

The crossover candidate rate was chosen to be $1/7$ (0.14), the crossover rate was chosen to be 0.7 and the mutation rate was chosen to be 0.01. These are higher than average and produced good results in initial testing.

### 7.1.4 Testing for Validity

Testing the validity of the results of the experiments is vital if the results are to be useful outside of the context of the experiments themselves. 10-fold cross-validation is run on each chromosome to achieve the fitness for that chromosome. The splitting of the folds happens before filtering, reducing over-fitting. As a whole, the corpus is split 30/70, with 70 percent of the corpus being used while using GAFFS. The best model obtained by GAFFS is then tested with the 30 percent of the corpus that was held back. The result of this test should reveal the amount of over-fitting.

### 7.2 Data Sets Used

The selection, annotation, and pre-processing of the data used in the experiments can have a significant impact on the results.

#### 7.2.1 Selection

Selecting what data to use can have a very strong impact on the applicability of the experiments run. Data that is artificial may not behave in a similar manner as real data, however, industry data that is allowable for publication is extremely hard to come by. Using documents that contain very little rationale may be prone to over-fitting, however, using
documents that are almost all rationale may be prone to under-fitting and the usefulness of the extracted rationale may not be much greater than the documents themselves. With these considerations, the data set used consisted of 200 Chrome bug reports extracted randomly from a subset of the data provided for the Mining Software Repositories 2011 mining challenge (http://2011.msrconf.org/msr-challenge.html).

The subset was created by taking all of the 49987 Chrome bug reports webpages and restricting the size of the bug reports to be larger than 20KB. This meant excluding 5246 bug report webpages that were 12KB and smaller, representing error and log-in pages, not actual bug report content, and excluding 24989 bug reports that were 20KB and smaller, representing bug reports averaging 3 to 5 sentences, deemed too short to be likely to have a substantial amount of rationale. This left the subset of 19752 bug reports that were greater than 20KB in size.

7.2.2 Annotation

Differences in annotating data can have a large impact on the usability of the models generated around them. Having a single researcher annotate all of a set of data is problematic as a single researcher will introduce their bias as what is and is not rationale onto all of the data, as well as a single researcher will be less likely to catch their mistakes. Multiple researchers annotating a set of data can overcome these problems, but can pose larger time constraints on the group, as annotations are reviewed and re-reviewed. Having researchers that have little experience with rationale annotate the data can lead to many mistakes, however, researchers with large amounts of experience with rationale are more difficult to find and may have tighter schedules. With this in mind, that data was annotated by four researchers, three with moderate experience with rationale and one with a large amount of experience with rationale.

After being imported into GATE (General Architecture for Text Engineering), documents were grouped into sets of ten, which were then assigned to two researchers, who annotated the documents independently of each other. After both researcher's annotations were complete, a third researcher would combine the annotations as they saw fit, meaning that at least two of the four researchers agreed on each annotation. [Rogers, et al., 2013] The rationale subsets consist of the following rationale: the argumentation subset (alternatives, arguments, assumptions, decisions, and requirements), the arguments-all subset (arguments, assumptions, and requirements) decisions (only decisions), and assumptions (only assumptions)

7.2.3 Pre-processing in GATE

Aside from manual annotation of the rationale, other features must be annotated in order to provide learning attributes for the classifiers. In order for the features used by the pipeline to work, Sentences and Tokens must be defined. Tokens were first extracted using the plug-ins provided by GATE, next a custom sentence splitter was used to identify the sentences. The GATE sentence splitter was not used because we found that it did not split the sentences correctly. Tokens were then annotated with Penn-Treebank parts of speech [Marcus, et al., 1993] using a GATE plug-in. After the sentences and tokens were annotated and the tokens had the parts of speech, the corpus was exported as XML files.

A program was then run on the XML files to add whether or not a token was listed as one of the general or domain rationale terms, described in detail below. At this point the data is ready for both the pipeline and genetic algorithm that uses it.
7.3 Features Used

Features are the most important part of the machine learning process, with the right ones, classifiers can draw clear conclusions about the absence or presence of features and whether or not a sentence has rationale. Wrong features have little to no correlation with the presence of rationale, which can cause error in classification, while missing features can do the same. The features used were taken from earlier research. [Rogers, et al., 2013]

- N-grams are series of N adjacent words concatenated with each other. When N-grams are referred to in this document, they may alternatively be called terms like “3-grams” or “bigrams”, but the “-gram” suffix designates them as made up of actual words. N-grams can provide strong correlation information, but at the cost of an extremely large set of attributes to learn over.

- Parts of speech (POS) are the Penn-Treebank [Marcus, et al., 1993] parts of speech for a word. Like N-grams, they can be concatenated adjacently by adding a number suffix, “2-pos” for example. Parts of speech can provide some correlation information, but tends to be limited due to the extremely small set of attributes to learn over.

- N-grams for specific parts of speech are the words of a specific part of speech. Only verbs, for example. They can also be concatenated in the same manner as above. N-grams for specific parts of speech can provide strong correlation information, as N-grams, but also cut down the set of attributes to learn over, due to only having a subset of N-grams.

- General and domain terms are like N-grams for specific parts of speech, but instead of being a specific part of speech, the word must be on either the general rationale terms list or the domain rationale terms list. The general rationale terms list is a list of rationale indicating terms that can apply to many different types of projects. The domain rationale terms list is a list of rationale indicating terms that apply specificity to web services, in order to more accurately target the data used (Chrome bug reports). It has similar benefits to the N-grams for specific parts of speech, with two exceptions: First, because the lists are created with rationale containing sentences in mind, general and domain terms are more likely to be relevant. Second, because the lists are created before looking at the data, general and domain terms may overlook words that have strong correlation but are not on the list, limiting discovery. General terms were taken from [Burge, 2005] and domain terms were taken from [AuditMyPC, 2010].

- Sentence length is the number of words in a sentence. While obviously limited, it can be used to find sentences that are unlikely to contain rationale, such as those that are very short, (1-4 words).

- N-closest sentences allows for context to be shared between sentences. When any of the above features uses closest sentences, that sentence’s attributes as well as N adjacent sentence’s attributes are weighted (to emphasize nearer sentences) and added as a feature. This way a sentence can indicate the presence or absence of rationale around it.
7.4 Classifier

Naïve Bayes Multinomial was used as the classifier for the experiments. This is because it was consistently one of the fastest and best performing classifiers in earlier experiments. Naïve Bayes Multinomial is a modified Naïve Bayes classifier that uses the count of present attributes, not just their presence as a branching decision. [McCallum and Nigam, 1998]

7.5 Expected Outcomes

The experiments are expected to provide interesting and relatively small feature sets that still perform well in classification. Binary rationale feature sets will probably be similar to argumentation subset feature sets. Argumentation subset feature sets will probably be similar to both binary rationale and arguments-all subset feature sets. Decision and alternative feature sets will probably be similar to one another, but different from the arguments-all subset feature sets.

Chapter 8 Results and Evaluation

After the experiments were run, the results were collected and then evaluated. The systems used in the experiments, the GATE-WEKA pipeline and GAFFS, were also evaluated for usability and extensibility.

8.1 GAFFS Experiment Results

When a GAFFS experiment ends, it produces a text file that contains all of the feature sets of the last generation, with their F-measure. They are also sorted by F-measure so that the best performing individual is the first on the list.

8.1.1 Results from Design Rationale Superset GAFFS

The GAFFS experiment that ran on binary design rationale produced some interesting results. There was a five way tie for the highest F-measure, as five very similar individuals had an F-measure of 0.926. This extremely good result is concerning, as over fitting may be to blame.

8.1.1.1 Feature Sets Produced

The five best feature sets all had the same F-measure, 0.926. These five feature sets are very similar, with only 9% of the features differing between any two sets. The full feature sets can be found in appendix C.1. Looking at Table 8.1, the row labelled “Binary Rationale” contains the average number of features that fall into each category. The row under it, labelled “Percent of Total” contains the percentage of the total features that were selected that fell into that category. These percentages will be used later to compare the results for the different rationale subsets. With only 61 features, 11% of the total, the genetic algorithm performed its task to generating a relatively small feature set. The different categories of features can be found in appendix C.6.
Table 8.1 Binary Rationale Feature Set

<table>
<thead>
<tr>
<th></th>
<th>Number of Features</th>
<th>Closest Sentence Features</th>
<th>N-gram Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Binary Rationale</td>
<td>61</td>
<td>38</td>
<td>17</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>11</td>
<td>63</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Part of Speech Features</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specific</td>
<td>General</td>
</tr>
<tr>
<td>Binary Rationale</td>
<td>47</td>
<td>8</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>76</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Verb Roots</th>
<th>Parts of Speech</th>
<th>Bag of Words</th>
<th>Sentence Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Rationale</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

8.1.2 Tests for Validity

A training ARFF file was made using the 140 Chrome bug reports that were used to optimize feature sets by GAFFS. The feature set used was the top performing feature set for binary rationale. The 60 Chrome bug reports that were not used by GAFFS were turned into the test ARFF. Info gain filtering was performed on the training ARFF. The Naïve Bayes Multinomial classifier was used for validation. With a precision of 0.722 and a recall of 0.843, the resulting F-measure was 0.778. When compared to the experimental F-measure, 0.926, this is much less. This means that the feature sets for binary rationale was over fit. However, 0.778 is still a very good result compared to human chosen feature sets.

8.1.2 Results from Design Rationale Subsets GAFFS

The GAFFS experiments that ran on the design rationale subsets (the argumentation subset, the arguments-all subset, decisions, and alternatives) produced interesting results. In previous experiments that did not use a genetic algorithm, the F-measure for decisions would be worse than binary rationale, but better than argumentation subset, which, in turn, would be better than alternatives. The F-measure of the arguments-all subset would be worse than all of the others. In the GAFFS results, the arguments-all subset did better than alternatives.

8.1.2.1 Feature Sets Produced

For argumentation subset, the feature sets that did the best were all very similar, but the one that did the best had an F-measure of 0.456. The ten best performing argumentation subset feature sets can be found in appendix C.2. Looking at table 2, the row labelled “Argumentation Subset” contains the number of features that fell into each category for the argumentation subset results. The row labelled “Percent of Total” immediately below it contains the percentage of the total features that were selected that fell into that category for the argumentation subset results.

For the arguments-all subset, the feature sets that did the best were also all very similar, but the one that did the best had an F-measure of 0.519. The ten best performing arguments-all subset feature sets can be found in appendix C.3. Looking at table 2, the row labelled “Arguments-All” contains the number of features that fell into each category for the arguments-all subset results. The row labelled “Percent of Total” immediately below it
contains the percentage of the total features that were selected that fell into that category for the arguments-all subset results.

For the alternatives subset, the feature sets that did the best were similar, but the one that did the best had an F-measure of 0.255. The ten best performing alternatives feature sets can be found in appendix C.4. Looking at table 2, the row labelled “Alternatives” contains the number of features that fell into each category for the alternatives results. The row labelled “Percent of Total” immediately below it contains the percentage of the total features that were selected that fell into that category for the alternatives results.

For the decisions subset, the feature sets that did the best were very similar, but the one that did the best had an F-measure of 0.765. The ten best performing decisions feature sets can be found in appendix C.5. Looking at table 8.2, the row labelled “Decisions” contains the number of features that fell into each category for the decisions results. The row labelled “Percent of Total” immediately below it contains the percentage of the total features that were selected that fell into that category for the decisions results.

<table>
<thead>
<tr>
<th>Table 8.2 Rationale Subset Feature Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argumentation Subset</td>
</tr>
<tr>
<td>Percent of Total</td>
</tr>
<tr>
<td>Arguments-All</td>
</tr>
<tr>
<td>Percent of Total</td>
</tr>
<tr>
<td>Alternatives</td>
</tr>
<tr>
<td>Percent of Total</td>
</tr>
<tr>
<td>Decisions</td>
</tr>
<tr>
<td>Percent of Total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part of Speech Features</th>
<th>Specific</th>
<th>General</th>
<th>General</th>
<th>Domain</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argumentation Subset</td>
<td>102</td>
<td>13</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>74</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Arguments-All</td>
<td>66</td>
<td>11</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>73</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Alternatives</td>
<td>84</td>
<td>13</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>59</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Decisions</td>
<td>68</td>
<td>11</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>74</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Table 8.2 Rationale Subset Feature Sets Continued

<table>
<thead>
<tr>
<th>Argumentation Subset</th>
<th>Verb Roots</th>
<th>Parts of Speech</th>
<th>Bag of Words</th>
<th>Sentence Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percent of Total</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Arguments-All</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Alternatives</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Decisions</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

8.1.2.2 Tests for Validity

For each rationale subset, a training ARFF file was made using the 140 Chrome bug reports that were used to optimize feature sets by GAFFS. The feature set used was the top performing feature set for binary rationale. The 60 Chrome bug reports that were not used by GAFFS were turned into the test ARFF. Info gain filtering was performed on the training ARFF. The Naive Bayes Multinomial classifier was used for validation.

With a precision of 0.456 and a recall of 0.683, the resulting F-measure for the argumentation subset was 0.547. When compared to the experimental F-measure, 0.456, this is somewhat greater. This means that the feature sets for the argumentation subset may be valid. However, the difference of 0.091 is a little concerning.

With a precision of 0.599 and a recall of 0.465, the resulting F-measure for arguments-all was 0.523. When compared to the experimental F-measure, 0.519, this is very close. This means that the feature sets for arguments-all is probably valid.

With a precision of 0.437 and a recall of 0.163, the resulting F-measure for arguments-all was 0.237. When compared to the experimental F-measure, 0.255, this is very close. This means that the feature sets for arguments-all is probably valid.

With a precision of 0.822 and a recall of 0.695, the resulting F-measure for arguments-all was 0.753. When compared to the experimental F-measure, 0.765, this is very close. This means that the feature sets for arguments-all is probably valid.

8.1.3 Comparison

Before comparison, the percent of certain types of features are of the total possible features should be given. Of all the possible features, there were 512 distinct features, listed in appendix C.6. Looking at table 8.3, the row labelled “Total Features” contains the number of features that fell into each category. The row labelled “Percent of Total” immediately below it contains the percentage of the total features that fell into that category.
Table 8.3 Total Feature Set

<table>
<thead>
<tr>
<th></th>
<th>Number of Features</th>
<th>Closest Sentence Features</th>
<th>N-gram Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Total Features</td>
<td>512</td>
<td>261</td>
<td>96</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>100</td>
<td>50</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Part of Speech Features</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specific</td>
<td>General</td>
</tr>
<tr>
<td>Total Features</td>
<td>360</td>
<td>60</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>75</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Verb Roots</th>
<th>Parts of Speech</th>
<th>Bag of Words</th>
<th>Sentence Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Features</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 8.4 contains the percentage of the total features that fell into each category for each argumentation subset. The row labelled “Total Features” corresponds to the total possible features. The row labelled “Binary Rationale” corresponds to the binary rationale features. The row labelled “Argumentation Subset” corresponds to the argumentation subset features. The row labelled “Alternatives” corresponds to the alternatives subset features. The row labelled “Decisions” corresponds to the decision subset features.

The binary rationale and all of the subsets used only a small portion of the total possible features, between 11 and 28 percent. This was to be expected, as the genetic algorithm penalized large feature sets. The binary rationale feature set was the only one that differed significantly from the baseline when considering N-Closest Sentences features. The binary rationale and the argumentation subset feature sets used significantly more than average 1-grams, and then rest used slightly more than average. 2, 3, and 4-grams were generally close to average, tending slightly above average. 5-grams were generally worse than average, with the binary rationale feature set was significantly worse than average.

Only the alternatives feature set differed significantly from average from the specific parts of speech features, of which it had much less. When considering the general parts of speech features, both the argumentation subset and alternatives had less than average. The binary rationale feature set had much less than average domain only term feature sets, otherwise, term feature sets were chosen close to proportionally when compared to the superset of all possible features. Verb roots features were more common than average, while parts of speech features were as common, or slightly less common then average. The argumentation subset and alternatives feature set contained close to average numbers of bag of word and sentence length features, while the others contained none at all.

27
Table 8.4 Feature Sets by Percent

<table>
<thead>
<tr>
<th></th>
<th>Number of Features</th>
<th>Closest Sentence Features</th>
<th>N-gram Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Total Features</td>
<td>100</td>
<td>50</td>
<td>20</td>
</tr>
<tr>
<td>Binary Rationale</td>
<td>11</td>
<td>63</td>
<td>28</td>
</tr>
<tr>
<td>Argumentation Subset</td>
<td>26</td>
<td>49</td>
<td>38</td>
</tr>
<tr>
<td>Arguments-All</td>
<td>17</td>
<td>51</td>
<td>23</td>
</tr>
<tr>
<td>Alternatives</td>
<td>28</td>
<td>50</td>
<td>23</td>
</tr>
<tr>
<td>Decisions</td>
<td>18</td>
<td>51</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Part of Speech Features</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specific</td>
<td>General</td>
</tr>
<tr>
<td>Total Features</td>
<td>75</td>
<td>12</td>
</tr>
<tr>
<td>Binary Rationale</td>
<td>76</td>
<td>13</td>
</tr>
<tr>
<td>Argumentation Subset</td>
<td>74</td>
<td>9</td>
</tr>
<tr>
<td>Arguments-All</td>
<td>73</td>
<td>12</td>
</tr>
<tr>
<td>Alternatives</td>
<td>59</td>
<td>9</td>
</tr>
<tr>
<td>Decisions</td>
<td>74</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Verb Roots</th>
<th>Parts of Speech</th>
<th>Bag of Words</th>
<th>Sentence Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Features</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Binary Rationale</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Argumentation Subset</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Arguments-All</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Alternatives</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Decisions</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8.5 compares the recall, “R”, precision, “P”, and f-measure, “F”, between the results from the GAFFS experiment, “GAFFS”, the results from the GAFFS experiment validation tests, “GAFFS (Val)”, the results from [Rogers, et al., 2013], “Working Paper”, and the inter-annotator agreement, also from [Rogers, et al., 2013]. The feature set for GAFFS can be found in appendix C1. The VADCC2_PSn feature set is equivalent to:

\(\{2\}vb+\{2\}av+\{2\}nnp+\{2\}dt+\{2\}cc+\{2\}pos|\{2\}C\_sentenceLength\)

The 1WGPSn feature set is equivalent to:

\(unigrams+pos|C\_sentenceLength\)

When compared to both the working paper and inter-annotator agreement, the GAFFS produced much better results. Even though the validity of the results are called into question, the lower number generated by the validity test was still greater than both the working paper and the inter-annotator agreement.
Table 8.5 Binary Rationale Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Feature Set</th>
<th>Classifier</th>
<th>R</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAFFS</td>
<td>Best Binary Rationale (C.1)</td>
<td>Naïve Bayes Multinomial</td>
<td>1</td>
<td>.862</td>
<td>.926</td>
</tr>
<tr>
<td>GAFFS (Val)</td>
<td>VADCC2 PSn</td>
<td>Naïve Bayes Multinomial</td>
<td>.722</td>
<td>.843</td>
<td>.778</td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2 PSn</td>
<td>Stacking</td>
<td>.878</td>
<td>.551</td>
<td>.677</td>
</tr>
<tr>
<td>Working Paper</td>
<td>1WGPSn</td>
<td>Random Forest</td>
<td>.424</td>
<td>.731</td>
<td>.537</td>
</tr>
<tr>
<td>Inter-annotator</td>
<td>Agreement</td>
<td></td>
<td></td>
<td></td>
<td>.69</td>
</tr>
</tbody>
</table>

Table 8.6 compares the recall, “R”, precision, “P”, and f-measure, “F”, between the results from the GAFFS experiment, “GAFFS”, the results from the GAFFS experiment validation tests, “GAFFS (Val)”, the results from [Rogers, et al., 2013], “Working Paper”, and the inter-annotator agreement, also from [Rogers, et al., 2013]. The feature set for GAFFS can be found in appendix C2. The 5PG feature set is equivalent to:

\[(5) pos\]

The V feature set is equivalent to:

verbs

When compared to the working paper and the inter-annotator agreement, the GAFFS produced slightly worse results. However, the validity results produced better f-measure results than the working paper and inter-annotator agreement. Overall, very little was gained by adding many more features when compared to the working paper.

Table 8.6 Argumentation Subset Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Feature Set</th>
<th>Classifier</th>
<th>R</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAFFS</td>
<td>Best Argumentation Subset (C.2)</td>
<td>Naïve Bayes Multinomial</td>
<td>.437</td>
<td>.473</td>
<td>.456</td>
</tr>
<tr>
<td>GAFFS (Val)</td>
<td>5PG</td>
<td>Naïve Bayes Multinomial</td>
<td>.456</td>
<td>.683</td>
<td>.547</td>
</tr>
<tr>
<td>Working Paper</td>
<td>5PG</td>
<td>Bayes Net</td>
<td>.812</td>
<td>.365</td>
<td>.504</td>
</tr>
<tr>
<td>Working Paper</td>
<td>5PG</td>
<td>Random Forest</td>
<td>.272</td>
<td>.706</td>
<td>.393</td>
</tr>
<tr>
<td>Working Paper</td>
<td>V</td>
<td>Stacking</td>
<td>.734</td>
<td>.395</td>
<td>.514</td>
</tr>
<tr>
<td>Inter-annotator</td>
<td>Agreement</td>
<td></td>
<td></td>
<td></td>
<td>.495</td>
</tr>
</tbody>
</table>

Table 8.7 compares the recall, “R”, precision, “P”, and f-measure, “F”, between the results from the GAFFS experiment, “GAFFS”, the results from the GAFFS experiment validation tests, “GAFFS (Val)”, the results from [Rogers, et al., 2013], “Working Paper”, and the inter-annotator agreement, also from [Rogers, et al., 2013]. The feature set for GAFFS can be found in appendix C3.

When compared to the working paper and the inter-annotator agreement, the GAFFS produced much better results, which were backed up by very consistent validity tests. Identification of the arguments-all subset produced the largest f-measure jump, going from a best of .375 (the inter-annotator agreement) to .523.
Table 8.7 Arguments-All Subset Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Feature Set</th>
<th>Classifier</th>
<th>R</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAFFS</td>
<td>Best Arguments-All Subset (C.3)</td>
<td>Naïve Bayes Multinomial</td>
<td>.549</td>
<td>.492</td>
<td>.519</td>
</tr>
<tr>
<td>GAFFS (Val)</td>
<td>Naïve Bayes Multinomial</td>
<td>.599</td>
<td>.465</td>
<td>.523</td>
<td></td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>Random Forest</td>
<td>.607</td>
<td>.054</td>
<td>.099</td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>Bayes Net</td>
<td>.208</td>
<td>.823</td>
<td>.332</td>
</tr>
<tr>
<td>Working Paper</td>
<td>V</td>
<td>Stacking</td>
<td>.282</td>
<td>.548</td>
<td>.373</td>
</tr>
<tr>
<td>Inter-annotator Agreement</td>
<td></td>
<td></td>
<td>.375</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.8 compares the recall, “R”, precision, “P”, and f-measure, “F”, between the results from the GAFFS experiment, “GAFFS”, the results from the GAFFS experiment validation tests, “GAFFS (Val)”, the results from [Rogers, et al., 2013], “Working Paper”, and the inter-annotator agreement, also from [Rogers, et al., 2013]. The feature set for GAFFS can be found in appendix C4.

When compared to the working paper and the inter-annotator agreement, the GAFFS produced worse results, with the validity results being the worst. It’s possible that the GAFFS run was an outlier, given the extremely low percentage of specific parts of speech features, or that Naïve Bayes Multinomial was not the best classifier for alternatives.

Table 8.8 Alternatives Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Feature Set</th>
<th>Classifier</th>
<th>R</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAFFS</td>
<td>Best Alternatives (C.4)</td>
<td>Naïve Bayes Multinomial</td>
<td>.536</td>
<td>.167</td>
<td>.255</td>
</tr>
<tr>
<td>GAFFS (Val)</td>
<td>Naïve Bayes Multinomial</td>
<td>.437</td>
<td>.163</td>
<td>.237</td>
<td></td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>Random Forest</td>
<td>.794</td>
<td>.142</td>
<td>.241</td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>Bayes Net</td>
<td>.131</td>
<td>.81</td>
<td>.226</td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>SGD</td>
<td>.429</td>
<td>.31</td>
<td>.36</td>
</tr>
<tr>
<td>Inter-annotator Agreement</td>
<td></td>
<td></td>
<td>.515</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.9 compares the recall, “R”, precision, “P”, and f-measure, “F”, between the results from the GAFFS experiment, “GAFFS”, the results from the GAFFS experiment validation tests, “GAFFS (Val)”, the results from [Rogers, et al., 2013], “Working Paper”, and the inter-annotator agreement, also from [Rogers, et al., 2013]. The feature set for GAFFS can be found in appendix C5. When compared to the working paper, the GAFFS produced similar results, with the validity results confirming.

Table 8.9 Decisions Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Feature Set</th>
<th>Classifier</th>
<th>R</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAFFS</td>
<td>Best Decisions (C.5)</td>
<td>Naïve Bayes Multinomial</td>
<td>.853</td>
<td>.693</td>
<td>.765</td>
</tr>
<tr>
<td>GAFFS (Val)</td>
<td>Naïve Bayes Multinomial</td>
<td>.822</td>
<td>.695</td>
<td>.753</td>
<td></td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>Random Forest</td>
<td>.889</td>
<td>.719</td>
<td>.795</td>
</tr>
<tr>
<td>Working Paper</td>
<td>VADCC2_PSn</td>
<td>Bayes Net</td>
<td>.514</td>
<td>.842</td>
<td>.638</td>
</tr>
<tr>
<td>Inter-annotator Agreement</td>
<td></td>
<td></td>
<td>.32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 8.1 shows the f-measure of the chromosome with the highest fitness in each generation for the binary rationale and each of the rationale subsets. Both alternatives and decisions are basically flat, which means that the genetic algorithm was not very useful in improving their f-measures.

**Fig. 8.1. Results by Generation**

### 8.1.4 Relationship between Results and Hypotheses

The experiments were expected to provide interesting and relatively small feature sets that still performed well in classification, and they generally did. The feature sets, being 11 to 28 percent of the total features were relatively small. They were also interesting, differing from the baseline percentages for a number of different features. Only alternatives did much worse than previous work in classification. [Rogers, et al., 2013]

Binary rationale feature sets were expected to be similar to the argumentation subset feature sets, however they differed significantly when it came to the number of, n-closest sentences, bag of word, and sentence length features. The argumentation subset feature sets were expected to be similar to both binary rationale and arguments-all subset feature sets, but differed to both when it came to the number of, bag of word, and sentence length features. Decision and alternative feature sets were expected be similar to one another, but different from the arguments-all subset feature sets. Decisions and the arguments-all subset feature sets tended to be very similar, and the alternative and the argumentation subset feature sets also tended to be somewhat similar.

- Overall general and domain term features overall were as common, or more common than average. However, general only term features were much more common than domain only features. This could either indicate that the domain terms were not
selected carefully enough, or that the general terms are more commonly found in rationale than domain terms.

- Overall, sentence length had little to no impact on the results. In fact, the two most underperforming feature sets when compared to previous results, the argumentation subset and alternatives, were the only feature sets that contained sentence length features. However, the number of possible sentence length features was very small to start with, so it may be better to explore the importance of sentence length features within a smaller feature set.

- Except for binary rationale, where the percentage of closest sentence features was much higher than average, closest sentence features seemed to not have a major impact classification.

- Generally, 1-grams were observed at a higher percentage than 2, 3, and 4-grams, which were observed at a higher percentage than 5-grams. The exceptions were the arguments-all subset and decisions, where 4-grams were selected the least.

8.2 Usability

The usability of the two systems created, the GATE-WEKA pipeline and GAFFS, is important, as it determines how easily experiments can be carried out and how much the GATE-WEKA pipeline and or GAFFS is used outside of the context of these experiments.

8.2.1 Pipeline Usability

The usability of the pipeline is very important, as it will be used in other experiments conducted by the research group, as well as possibly other researchers working on different problems. This means that it has to be easy to use, robust, and adaptable.

The GATE-WEKA pipeline is extendible in a couple of ways. First, neither the filtering, nor the classification, nor the feature set is hard coded. Variables can be set to change where the pipeline looks for features, classifiers, filters, and GATE xml files. Variables can also be set to determine how many folds the pipeline will create, where the temp folder will be created and named, the annotations that are considered valid for running, the WEKA directory, the feature set that will be run, and where the generated xml files will be output. The template files also allow for extensibility by allowing many different custom features to be generated.

The GATE-WEKA pipeline is limited by the fact that it currently only runs on unix systems. Also, it currently only runs on the command line, which may make it more difficult to use.

8.2.2 GAFFS Usability

The usability of GAFFS is important as the ability to use it easily affect the ability to run experiments. Because GAFFS is not expected to be used by others like the GATE-WEKA pipeline, it does not have to have as good usability or extensibility.

GAFFS is extendible in all of the ways that the GATE-WEKA pipeline it uses is, except that the filter and classifier are hard coded. The number of individuals in a generation and the number of generations are controlled by variables. The mutation and crossover rates are variables, but the fitness function is hard coded.
GAFFS has some important limitations. First, it has all of the limitations of the GATE-WEKA pipeline. It also has its filter, classifier, and fitness function hard coded.

8.3 Summary of Results

After the experiments were run, the different design rationale subsets, (binary rationale, the argumentation subset, the arguments-all subset, alternatives, and decisions) the best feature sets obtained by GAFFS were tested for validity and compared to the best results found in [Rogers, et al., 2013]. All of the subsets were relatively small and differed in percentages of the types of features they were composed of. Binary rationale results did better than the working paper, although the validity test showed inconsistencies. The argumentation subset results performed similarly to the working paper. The arguments-all subset did much better than the working paper. The alternatives results did worse than the working paper, and decisions performed similarly to the working paper.

Chapter 9 Future Research

While the experiments ran showed that interesting and relatively small feature sets that also perform well can be produced by a genetic algorithm, and that some certain types of features can generally drive performance more than others, there are many questions yet to be answered.

9.1 Expanded Experiments

The results produced by GAFFS was varied based on what rationale subset was used. While the other experiments performed similarly when subjected to validation, binary rationale and the argumentation subset both had validation tests that performed very differently. The binary rationale validation test did worse than the experimental results, and the argumentation subset validation test did better. There were also cases where rationale subsets did worse than observed in [Rogers, et al., 2013], such as the alternatives subset.

Taken together, this may suggest that some of the experiments that were run may have produced outlier, or at least non-average results. Because GAFFS was run once for each rationale subset, there were no other experimental results to see, if, in fact, the results were outliers. More GAFFS experiments should be run on each of the rationale subsets, to be able to either strengthen or weaken the validity of the conclusions reached.

Additional experiments should also be conducted on different types of rationale containing documents. Because the experiments were only conducted on bug reports, the conclusions can only most widely be applied to identifying rationale in bug reports.

Multi-stage rationale experiments should also be conducted. This would mean that rationale would first be identified as binary rationale or not, and then a second pass through the sentences identified as binary rationale would identify the specific type of rationale or the subset that that rationale belongs to. This could improve performance of currently underperforming rationale subsets, such as alternatives.

The experiments could also be run using different types of filters or classifiers, to see if more optimal results or more interesting feature sets can be produced. The main problem that this could promote would be over fitting.

9.2 Using the Pipeline

The pipeline was critical when conducting GAFFS experiments, but its functionality
is still untested for other experiments and lacks a proper user interface. More, differing, experiments should be run with the pipeline, to make sure that it is, in fact, as useful as it was planned to be. The pipeline should also be expanded to include a user interface that would allow easier customization and use. It should also be expanded to allow for greater customization and use, and be kept an active project, so that is remains compatible with the current versions of both GATE [Cunningham, et al., 2002] and WEKA [Hall, et al., 2009].

Chapter 10 Conclusions

The two questions I set out to answer were:

1. What feature sets and classifiers are chosen by a feature set size weighted genetic algorithm when extracting design rationale from text documents?
2. What feature sets and classifiers are chosen by a feature set size weighted genetic algorithm when extracting design rationale subsets from text documents?

Both of the questions were answered after conducting experiments with GAFFS. An in-depth analysis of the results can be found in chapter 8, and the top ten performing feature sets for each rationale subset can be found in appendix C. The feature sets produced were interesting, relatively small, and generally performed as good or better than other feature sets used on the same data. [Rogers, et al., 2013]

The validity tests for the experiments ran on binary rationale and the argumentation subset were interesting, as the binary rationale validation tests did much worse than the experimental results, but still did better than previously selected feature sets, and the argumentation subset validation tests did better than the experimental results.

Both the binary rationale and the arguments-all subset did much better than found in [Rogers, et al., 2013], with the arguments-all subset increasing the best f-measure from 0.375 to 0.523. Alternatives, on the other hand, produced worse results. The interesting results could indicate that not enough tests were run and that some of the results may be outliers. It could also indicate that there is an error somewhere in either the implementation of the GATE-WEKA pipeline, or in GAFFS.

The experiments modelled well the effect that n-grams features had on performance, but some features, such as bag of words and sentence length were represented by such a small number of features, conclusions drawn about them could be invalid, as they may not have had a chance to make large performance impacts.
Bibliography


Hall, Frank, Holmes, Pfahringer, Reutemann, Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.


Rogers, Qaio, Gung, Mathur, Burge, (2013). Using Text Mining to Extract Rationale from Existing Documentation, working paper


Georgia State University, Atlanta, GA, USA. Advisor(s) Anu G. Bourgeois. AAI3293841.


**Appendices**

**Appendix A Lists of Rationale Words**

**A.1 List of Domain Rationale Words** [AuditMyPC, 2010]

<table>
<thead>
<tr>
<th>Abuse</th>
<th>Composite</th>
<th>External</th>
<th>Load</th>
<th>Predictable</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Compromise</td>
<td>Facilitates</td>
<td>Loadable</td>
<td>Prevention</td>
<td>Standardization</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Computer</td>
<td>Failure</td>
<td>Local</td>
<td>Prevents</td>
<td>Standards</td>
</tr>
<tr>
<td>Account</td>
<td>Computing</td>
<td>False</td>
<td>Location</td>
<td>Privacy</td>
<td>Star</td>
</tr>
<tr>
<td>Ack</td>
<td>Condition</td>
<td>Fast</td>
<td>Log</td>
<td>Private</td>
<td>State</td>
</tr>
<tr>
<td>Acls</td>
<td>Conference</td>
<td>Fault</td>
<td>Logic</td>
<td>Privilege</td>
<td>Static</td>
</tr>
<tr>
<td>Action</td>
<td>Confidentiality</td>
<td>Faults</td>
<td>Lookup</td>
<td>Proactive</td>
<td>Stay</td>
</tr>
<tr>
<td>Actions</td>
<td>Configuration</td>
<td>Features</td>
<td>Loop</td>
<td>Probe</td>
<td>Stealth</td>
</tr>
<tr>
<td>Active</td>
<td>Confirmation</td>
<td>Feedback</td>
<td>Loss</td>
<td>Processing</td>
<td>Storage</td>
</tr>
<tr>
<td>Activex</td>
<td>Conflicts</td>
<td>File</td>
<td>Mac</td>
<td>Products</td>
<td>Store</td>
</tr>
<tr>
<td>Activity</td>
<td>Connections</td>
<td>Files</td>
<td>Machine</td>
<td>Profile</td>
<td>Straight</td>
</tr>
<tr>
<td>Actors</td>
<td>Consistency</td>
<td>Filter</td>
<td>Macro</td>
<td>Program</td>
<td>Stream</td>
</tr>
<tr>
<td>Ad</td>
<td>Consistent</td>
<td>Filtering</td>
<td>Mail</td>
<td>Programs</td>
<td>String</td>
</tr>
<tr>
<td>Address</td>
<td>Content</td>
<td>Finger</td>
<td>Malicious</td>
<td>Projects</td>
<td>Strong</td>
</tr>
<tr>
<td>Addressing</td>
<td>Contention</td>
<td>Firewall</td>
<td>Malware</td>
<td>Promiscuous</td>
<td>Sub</td>
</tr>
<tr>
<td>Administrative Continuity</td>
<td>Firewalls</td>
<td>Management</td>
<td>Property</td>
<td>Supplying</td>
<td></td>
</tr>
<tr>
<td>Administrator</td>
<td>Contract</td>
<td>Flooding</td>
<td>Mandatory</td>
<td>Proprietary</td>
<td>Supports</td>
</tr>
<tr>
<td>Advanced</td>
<td>Contrast</td>
<td>Floor</td>
<td>Manipulation</td>
<td>Protection</td>
<td>Sweep</td>
</tr>
<tr>
<td>Adware</td>
<td>Control</td>
<td>Flow</td>
<td>Mapped</td>
<td>Protocol</td>
<td>Switch</td>
</tr>
<tr>
<td>Aes</td>
<td>Controls</td>
<td>Force</td>
<td>Mapping</td>
<td>Proxy</td>
<td>Switched</td>
</tr>
<tr>
<td>Agency</td>
<td>Convention</td>
<td>Form</td>
<td>Markup</td>
<td>Public</td>
<td>Symmetric</td>
</tr>
<tr>
<td>Agent</td>
<td>Cookie</td>
<td>Format</td>
<td>Master</td>
<td>Race</td>
<td>Syn</td>
</tr>
<tr>
<td>Alert</td>
<td>Corruption</td>
<td>Forward</td>
<td>Md</td>
<td>Re</td>
<td>System</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Cost</td>
<td>Forwarding</td>
<td>Measures</td>
<td>Read</td>
<td>Systems</td>
</tr>
<tr>
<td>Allows</td>
<td>Crack</td>
<td>Fragmentation</td>
<td>Mechanism</td>
<td>Real</td>
<td>Tables</td>
</tr>
<tr>
<td>Amount</td>
<td>Cracking</td>
<td>Frames</td>
<td>Memory</td>
<td>Reconfigure</td>
<td>Tamper</td>
</tr>
<tr>
<td>Analysis</td>
<td>Crash</td>
<td>Fraud</td>
<td>Message</td>
<td>Record</td>
<td>Task</td>
</tr>
<tr>
<td>Analyzing</td>
<td>Crimeware</td>
<td>From</td>
<td>Metaphor</td>
<td>Recovery</td>
<td>Tcp</td>
</tr>
<tr>
<td>Anti</td>
<td>Criminals</td>
<td>Ftp</td>
<td>Method</td>
<td>Red</td>
<td>Technology</td>
</tr>
<tr>
<td>Antivirus</td>
<td>Criteria</td>
<td>Full</td>
<td>Metric</td>
<td>Redirect</td>
<td>Telnet</td>
</tr>
<tr>
<td>Api</td>
<td>Cross</td>
<td>Fully</td>
<td>Microsoft</td>
<td>Reflection</td>
<td>Template</td>
</tr>
<tr>
<td>Applet</td>
<td>Cryptographic</td>
<td>Function</td>
<td>Mode</td>
<td>Registry</td>
<td>Terminate</td>
</tr>
<tr>
<td>Application</td>
<td>Cut</td>
<td>Functionality</td>
<td>Model</td>
<td>Regression</td>
<td>Test</td>
</tr>
<tr>
<td>Architecture</td>
<td>Cyber</td>
<td>Functions</td>
<td>Modification</td>
<td>Reload</td>
<td>Testing</td>
</tr>
<tr>
<td>Ascci</td>
<td>Daemon</td>
<td>Gateway</td>
<td>Modify</td>
<td>Rename</td>
<td>Text</td>
</tr>
<tr>
<td>Asp</td>
<td>Dancing</td>
<td>Gnu</td>
<td>Modules</td>
<td>Repair</td>
<td>Threading</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Dark</td>
<td>Hash</td>
<td>Monitoring</td>
<td>Replication</td>
<td>Threat</td>
</tr>
<tr>
<td>Attack</td>
<td>Data</td>
<td>Hacker</td>
<td>Monitors</td>
<td>Reply</td>
<td>Threats</td>
</tr>
<tr>
<td>Attacks</td>
<td>Datagram</td>
<td>Hacking</td>
<td>Mp</td>
<td>Reporting</td>
<td>Tiger</td>
</tr>
<tr>
<td>attributes</td>
<td>day</td>
<td>handles</td>
<td>ms</td>
<td>repudiation</td>
<td>tls</td>
</tr>
<tr>
<td>------------</td>
<td>-----</td>
<td>---------</td>
<td>----</td>
<td>-------------</td>
<td>-----</td>
</tr>
<tr>
<td>audit</td>
<td>deadlock</td>
<td>handling</td>
<td>multi</td>
<td>request</td>
<td>tm</td>
</tr>
<tr>
<td>auditing</td>
<td>death</td>
<td>handshake</td>
<td>multiple</td>
<td>requests</td>
<td>token</td>
</tr>
<tr>
<td>authenticate</td>
<td>defacement</td>
<td>hardware</td>
<td>name</td>
<td>reset</td>
<td>tom</td>
</tr>
<tr>
<td>authentication default</td>
<td>hash</td>
<td>naming</td>
<td>resident</td>
<td>tool</td>
<td></td>
</tr>
<tr>
<td>authenticity</td>
<td>defence</td>
<td>header</td>
<td>nat</td>
<td>resource</td>
<td>toolbar</td>
</tr>
<tr>
<td>authorization</td>
<td>defence</td>
<td>heuristic</td>
<td>national</td>
<td>responds</td>
<td>toolkit</td>
</tr>
<tr>
<td>automated</td>
<td>defined</td>
<td>hiding</td>
<td>natural</td>
<td>response</td>
<td>tp</td>
</tr>
<tr>
<td>availability</td>
<td>degradation</td>
<td>hijack</td>
<td>net</td>
<td>restart</td>
<td>trace</td>
</tr>
<tr>
<td>averse</td>
<td>delay</td>
<td>hijacking</td>
<td>network</td>
<td>retry</td>
<td>tracking</td>
</tr>
<tr>
<td>avoids</td>
<td>demand</td>
<td>hole</td>
<td>networks</td>
<td>rfc</td>
<td>transactions</td>
</tr>
<tr>
<td>background</td>
<td>denial</td>
<td>horse</td>
<td>newsgroup</td>
<td>rich</td>
<td>transfer</td>
</tr>
<tr>
<td>backup</td>
<td>dependencies</td>
<td>host</td>
<td>ntfs</td>
<td>ring</td>
<td>translation</td>
</tr>
<tr>
<td>bag</td>
<td>dependent</td>
<td>https</td>
<td>null</td>
<td>rip</td>
<td>transmission</td>
</tr>
<tr>
<td>bandwidth</td>
<td>depth</td>
<td>hub</td>
<td>octet</td>
<td>risk</td>
<td>transport</td>
</tr>
<tr>
<td>banner</td>
<td>des</td>
<td>human</td>
<td>office</td>
<td>rogue</td>
<td>tree</td>
</tr>
<tr>
<td>basic detection</td>
<td>hyperlink</td>
<td>officer</td>
<td>room</td>
<td>triggered</td>
<td></td>
</tr>
<tr>
<td>batch</td>
<td>detects</td>
<td>id</td>
<td>offset</td>
<td>root</td>
<td>triggers</td>
</tr>
<tr>
<td>bcp</td>
<td>devices</td>
<td>identifier</td>
<td>onunload</td>
<td>rootkit</td>
<td>trojan</td>
</tr>
<tr>
<td>bind</td>
<td>dialer</td>
<td>identity</td>
<td>open</td>
<td>router</td>
<td>trojans</td>
</tr>
<tr>
<td>bios</td>
<td>dictionary</td>
<td>ids</td>
<td>operating</td>
<td>routing</td>
<td>trust</td>
</tr>
<tr>
<td>bit</td>
<td>digest</td>
<td>ietf</td>
<td>operational</td>
<td>rpc</td>
<td>trusted</td>
</tr>
<tr>
<td>block</td>
<td>digital</td>
<td>imap</td>
<td>operations</td>
<td>rsa</td>
<td>tty</td>
</tr>
<tr>
<td>blocker</td>
<td>direct</td>
<td>impact</td>
<td>orange</td>
<td>rule</td>
<td>tunneling</td>
</tr>
<tr>
<td>blocks</td>
<td>directory</td>
<td>incident</td>
<td>os</td>
<td>rules</td>
<td>udp</td>
</tr>
<tr>
<td>blog</td>
<td>disclosure</td>
<td>infection</td>
<td>overflow</td>
<td>run</td>
<td>unc</td>
</tr>
<tr>
<td>book</td>
<td>disk</td>
<td>information</td>
<td>overlap</td>
<td>safety</td>
<td>undo</td>
</tr>
<tr>
<td>boot</td>
<td>display</td>
<td>infrastructure</td>
<td>overload</td>
<td>sandbox</td>
<td>uniform</td>
</tr>
<tr>
<td>border</td>
<td>distributed</td>
<td>injection</td>
<td>overwrites</td>
<td>scan</td>
<td>universal</td>
</tr>
<tr>
<td>bot</td>
<td>distribution</td>
<td>input</td>
<td>overwriting</td>
<td>scanner</td>
<td>university</td>
</tr>
<tr>
<td>breach</td>
<td>dns</td>
<td>insecurity</td>
<td>owner</td>
<td>scanning</td>
<td>unix</td>
</tr>
<tr>
<td>bridge</td>
<td>doc</td>
<td>inspection</td>
<td>packet</td>
<td>screen</td>
<td>unprotected</td>
</tr>
<tr>
<td>british</td>
<td>documentation institute</td>
<td>page</td>
<td>script</td>
<td>uri</td>
<td></td>
</tr>
<tr>
<td>broadcast</td>
<td>domain</td>
<td>integration</td>
<td>paging</td>
<td>scripting</td>
<td>url</td>
</tr>
<tr>
<td>browser</td>
<td>domains</td>
<td>integrity</td>
<td>pair</td>
<td>scripts</td>
<td>usability</td>
</tr>
<tr>
<td>brute</td>
<td>door</td>
<td>intelligence</td>
<td>parameters</td>
<td>secure</td>
<td>use</td>
</tr>
<tr>
<td>buffer</td>
<td>dos</td>
<td>interaction</td>
<td>partitions</td>
<td>security</td>
<td>user</td>
</tr>
<tr>
<td>bug</td>
<td>drive</td>
<td>interface</td>
<td>passive</td>
<td>send</td>
<td>users</td>
</tr>
<tr>
<td>building</td>
<td>driving</td>
<td>interfaces</td>
<td>password</td>
<td>sensitive</td>
<td>uses</td>
</tr>
<tr>
<td>business</td>
<td>dropper</td>
<td>internal</td>
<td>passwords</td>
<td>separation</td>
<td>utilizes</td>
</tr>
<tr>
<td>byte</td>
<td>dsa</td>
<td>international</td>
<td>patch</td>
<td>sequences</td>
<td>validation</td>
</tr>
<tr>
<td>cac</td>
<td>dss</td>
<td>internet</td>
<td>patching</td>
<td>server</td>
<td>value</td>
</tr>
<tr>
<td>cache</td>
<td>dynamic</td>
<td>interoperability</td>
<td>path</td>
<td>service</td>
<td>values</td>
</tr>
<tr>
<td>caching</td>
<td>ease</td>
<td>interpretation</td>
<td>pattern</td>
<td>session</td>
<td>variable</td>
</tr>
<tr>
<td>call</td>
<td>eavesdropping</td>
<td>interrupt</td>
<td>payload</td>
<td>set</td>
<td>variant</td>
</tr>
<tr>
<td>capacity</td>
<td>echo</td>
<td>intranet</td>
<td>payloads</td>
<td>sha</td>
<td>vbs</td>
</tr>
<tr>
<td>capture</td>
<td>efficient</td>
<td>intrusion</td>
<td>peer</td>
<td>shadow</td>
<td>vector</td>
</tr>
<tr>
<td>care</td>
<td>electronic</td>
<td>investigation</td>
<td>pem</td>
<td>share</td>
<td>verification</td>
</tr>
<tr>
<td>cast</td>
<td>email</td>
<td>ip</td>
<td>perceived</td>
<td>shared</td>
<td>version</td>
</tr>
<tr>
<td>cell</td>
<td>encrypted</td>
<td>ips</td>
<td>performance</td>
<td>shell</td>
<td>violation</td>
</tr>
</tbody>
</table>
A.2 List of General Rationale Words [Burge, 2005]

<table>
<thead>
<tr>
<th>Abuse</th>
<th>Completeness</th>
<th>Fixed</th>
<th>Meeting</th>
<th>Replaceable</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Condition</td>
<td>Flexibility</td>
<td>Method</td>
<td>Reporting</td>
<td>Test</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Conflicts</td>
<td>For</td>
<td>Metric</td>
<td>Require</td>
<td>Testing</td>
</tr>
<tr>
<td>Action</td>
<td>Connections</td>
<td>Form</td>
<td>Model</td>
<td>Requirements</td>
<td>Thing</td>
</tr>
<tr>
<td>Actions</td>
<td>Content</td>
<td>Forward</td>
<td>Modification</td>
<td>Requires</td>
<td>Through</td>
</tr>
<tr>
<td>Active</td>
<td>Control</td>
<td>From</td>
<td>Modify</td>
<td>Research</td>
<td>Time</td>
</tr>
<tr>
<td>Activity</td>
<td>Cost</td>
<td>Good</td>
<td>Naming</td>
<td>Resolution</td>
<td>Times</td>
</tr>
<tr>
<td>Additional</td>
<td>Criteria</td>
<td>Graceful</td>
<td>Negative</td>
<td>Resource</td>
<td>Tiny</td>
</tr>
<tr>
<td>Address</td>
<td>Dark</td>
<td>Guidance</td>
<td>Non</td>
<td>Reuse</td>
<td>Tm</td>
</tr>
<tr>
<td>Addressing</td>
<td>Default</td>
<td>Half</td>
<td>Not</td>
<td>Reverse</td>
<td>Tool</td>
</tr>
<tr>
<td>Advanced</td>
<td>Defined</td>
<td>High</td>
<td>On</td>
<td>Risk</td>
<td>Top</td>
</tr>
<tr>
<td>Allows</td>
<td>Delay</td>
<td>Human</td>
<td>One</td>
<td>Role</td>
<td>Triple</td>
</tr>
<tr>
<td>Amount</td>
<td>Denial</td>
<td>Idea</td>
<td>Open</td>
<td>Rule</td>
<td>Uniform</td>
</tr>
<tr>
<td>An</td>
<td>Design</td>
<td>In</td>
<td>Our</td>
<td>Rules</td>
<td>Universal</td>
</tr>
<tr>
<td>Analysis</td>
<td>Different</td>
<td>Incident</td>
<td>Owner</td>
<td>Secondary</td>
<td>Up</td>
</tr>
<tr>
<td>Analyzing</td>
<td>Disaster</td>
<td>Increase</td>
<td>Pair</td>
<td>Section</td>
<td>Usability</td>
</tr>
<tr>
<td>Appropriate</td>
<td>Documentation</td>
<td>Increased</td>
<td>Perceived</td>
<td>See</td>
<td>Use</td>
</tr>
<tr>
<td>Area</td>
<td>Driven</td>
<td>Increases</td>
<td>Performance</td>
<td>Self</td>
<td>User</td>
</tr>
<tr>
<td>Assessment</td>
<td>Due</td>
<td>Information</td>
<td>Performs</td>
<td>Simple</td>
<td>Users</td>
</tr>
<tr>
<td>Assignment</td>
<td>Dynamic</td>
<td>Infrastructure</td>
<td>Physical</td>
<td>Sizes</td>
<td>Uses</td>
</tr>
</tbody>
</table>
Appendix B Example Files

B.1 Example XML File TextWithNodes

```xml
<B.1 Example XML File TextWithNodes>

<Node id="0" /> Issue <Node id="5" /> <Node id="6" /> 32225 <Node id="11" /> <Node id="12" /> - <Node id="13" /> <Node id="14" /> chromium <Node id="22" /> <Node id="23" /> - <Node id="24" /> <Node id="25" /> Shockwave <Node id="34" /> <Node id="35" /> does <Node id="39" /> n't <Node id="42" /> <Node id="43" /> respond <Node id="50" />

B.2 Example XML File Annotations

<Annotation Id="336" Type="Token" StartNode="25" EndNode="34">
  <Feature>
    <Name className="java.lang.String">length</Name>
    <Value className="java.lang.String">9</Value>
  </Feature>

  <Feature>
    <Name className="java.lang.String">orth</Name>
    <Value className="java.lang.String">upperInitial</Value>
  </Feature>

  <Feature>
    <Name className="java.lang.String">category</Name>
    <Value className="java.lang.String">NNP</Value>
  </Feature>

  <Feature>
    <Name className="java.lang.String">kind</Name>
    <Value className="java.lang.String">word</Value>
  </Feature>

  <Feature>
    <Name className="java.lang.String">string</Name>
  </Feature>
</Annotation>

</B.1 Example XML File TextWithNodes>
```
B.3 Example ARFF File

@relation 'text_files_in_F:\\html-txt'

@attribute filename string
@attribute contents string

@data
1.txt,'on code open source projects on code on code is easy open source on code on code storage storage source code code easy high code base an issue easy projects care in account stay in access code service privacy policy '  
10069.txt,'issue on code in be due network an browser web issue open for advanced issue person issue be back list bug in comment not in dynamic on sandbox from line line comment test is not be is in comment privacy '  
10113.txt,'issue on code in be due network an browser web issue open for advanced issue person issue be back list bug in comment not in accessibility is in accessibility is not for border not block offset point offset point block test is speed test from line line comment bug be is in comment privacy '  
11131.txt,'issue toolbar browser on code in be due network an browser web issue open for advanced issue toolbar browser person issue be back list issue bug in comment browser toolbar overlap on is way one way is in comment comment for comment comment for toolbar browser in comment privacy '  
11285.txt,'issue be on code in be due network an browser web issue open for advanced issue be issue be back list fixed bug in comment open on on for be internet control see comment comment comment for fixed in comment privacy '  
11418.txt,'issue is not in on code in be due network an browser web issue open for advanced issue is not in issue be back list bug in comment version is be is not additional information is javascript code detects through browser not call comment for on comment for comment for in comment privacy '  
40781.txt,'issue browser when on code in an browser web issue open for advanced issue browser when issue be back list bug in comment version is os architecture in in windows access access
open one browser back up allows is be browser be additional information for first on not fixed in comment stack trace comment be comment comment on bug open stack in comment privacy

B.4 Example Vectorized ARFF File

@relation 'text_files_in_F:\html-txt-
weka.filters.unsupervised.attribute.StringToWordVector-R1,2-
W1000-prune-rate-1.0-N0-S-
stemmerweka.core.stemmers.NullStemmer-M1-
stopwordsC:\Users\Lin\workspace\TxtToARFF\stp.txt-
tokenizerweka.core.tokenizers.AlphabeticTokenizer'

@attribute access numeric
@attribute accessibility numeric
@attribute account numeric
@attribute action numeric

@attribute wide numeric
@attribute wild numeric
@attribute windows numeric
@attribute wireless numeric
@attribute zip numeric
@attribute class {T,F}

@Data

{0 1, 2 1, 14 1, 24 1, 42 1, 54 1, 90 1, 134 1, 141 1, 152 1, 153 1, 202 1, 204 1, 225 1, 233 1, 239 1, 283 1, 293 1, 303 1, 304 1, 360 ?}
{9 1, 14 1, 22 1, 26 1, 35 1, 37 1, 54 1, 56 1, 86 1, 87 1, 115 1, 122 1, 141 1, 152 1, 153 1, 166 1, 169 1, 197 1, 199 1, 202 1, 204 1, 218 1, 233 1, 272 1, 317 1, 353 1, 360 ?}
{1 1, 9 1, 14 1, 22 1, 26 1, 30 1, 33 1, 35 1, 37 1, 54 1, 56 1, 86 1, 115 1, 122 1, 141 1, 152 1, 153 1, 166 1, 169 1, 197 1, 199 1, 201 1, 202 1, 204 1, 218 1, 223 1, 231 1, 295 1, 317 1, 353 1, 360 ?}
{9 1, 14 1, 22 1, 26 1, 35 1, 37 1, 54 1, 56 1, 86 1, 115 1, 141 1, 152 1, 153 1, 169 1, 197 1, 202 1, 204 1, 218 1, 233 1, 317 1, 353 1, 360 ?}
{9 1, 14 1, 22 1, 26 1, 35 1, 37 1, 54 1, 56 1, 67 1, 86 1, 114 1, 115 1, 141 1, 149 1, 153 1, 169 1, 197 1, 202 1, 204 1, 323 1, 328 1, 353 1, 360 ?}
{6 1, 9 1, 14 1, 22 1, 26 1, 35 1, 37 1, 40 1, 54 1, 56 1, 78 1, 86 1, 115 1, 141 1, 142 1, 152 1, 153 1, 169 1, 197 1, 199 1, 202 1, 204 1, 233 1, 321 1, 347 1, 353 1, 360 ?}
{0 1, 6 1, 9 1, 12 1, 14 1, 19 1, 22 1, 26 1, 35 1, 37 1, 54 1, 56 1, 113 1, 114 1, 115 1, 141 1, 142 1, 152 1, 153 1, 169 1, 199 1, 202 1, 203 1, 204 1, 207 1, 233 1, 298 1, 327 1, 334 1, 347 1, 353 1, 354 1, 357 1, 360 ?}
Appendix C Feature Sets

C.1 Binary Rationale 10 Best Feature Sets

The binary rationale ten best feature sets were the ten feature sets from the last generation from the binary rationale experiment with the best f-measures.

{2}(4)jjs+{2}dt+{2}(4)pronouns+{2}conjunctions+{2}(2)2nnp+{2}(4)fw+{2}(3)nnps+{2}adverbs+{2}wtdt+{2}(4)poss+{2}vbz+{2}vbz+{2}in+{2}(5)ls+{2}(3)noun+{2}adverbs+{2}vbd+{2}vbp+{2}(4)vbn+{4}nnp+{2}(3)nnp+rbs+{5}jjs+{2}(5)vbd+{2}(3)to+{2}ex+{2}(4)ontology+roots+{2}(2)cc+{2}(3)vbn+{4}conjunctions+{3}in+{2}(5)md+{2}(3)jj+{2}(2)rp+{2}fw+{2}(2)to+{2}pdt+{2}nnp+{2}(3)md+{2}(5)vbz+{2}(2)vbn+{2}vb+{2}(2)vbg+{2}(2)vbp+{2}(2)sym+{4}general+{2}(3)noun+{2}vbd+{2}(2)(5)vbz+{2}(2)vbg+{2}(2)vb+{2}(2)sym+{4}general+{2}nnps+{2}pronouns+{3}vbz+{2}(2)roots+{2}(5)wtdt+{2}(5)pos, 0.9256924178258624

{2}(4)jjs+{2}dt+{2}(4)pronouns+{2}conjunctions+{2}(2)2nnp+{2}(4)fw+{2}(3)nnps+{2}adverbs+{2}wtdt+{2}(4)poss+{2}vbz+{2}vbz+{2}in+{2}(5)ls+{2}(3)noun+{2}adverbs+{2}vbd+{2}vbp+{2}(4)vbn+{4}nnp+{2}(4)wrb+{2}(3)nnp+rbs+{5}jjs+{2}(5)vbd+{2}(3)to+{2}ex+{2}(4)ontology+roots+{2}(2)cc+{4}conjunctions+{3}in+{2}(5)md+{2}(3)jj+{2}(2)rp+{2}fw+{2}(2)to+{2}pdt+{2}nnp+{2}(3)md+{2}(5)vbz+{2}(2)vbn+{2}vb+{2}(2)vbg+{2}(2)vbp+{2}(2)sym+{4}general+{2}(3)noun+{2}vbd+{2}(2)(5)vbz+{2}(2)vbg+{2}(2)vb+{2}(2)sym+{4}general+{2}nnps+{2}pronouns+{3}vbz+{2}(2)roots+{2}(5)pos, 0.9256924178258624

{2}(4)jjs+{2}dt+{2}(4)pronouns+{2}conjunctions+{2}(2)2nnp+{2}(4)fw+{2}(3)nnps+{2}adverbs+{2}wtdt+{2}(4)poss+{2}vbz+{2}vbz+{2}in+{2}(5)ls+{2}(3)noun+{2}adverbs+{2}vbd+{2}vbp+{2}(4)vbn+{4}nnp+{2}(2)rp+{2}(2)fw+{2}(2)to+{2}pdt+{2}nnp+{2}(2)vbn+{2}vb+{2}(2)vbg+{2}(2)vbp+{2}(2)sym+{4}general+{2}(2)2noun+{2}pdt+{2}(2)md+{2}(4)cc+{2}(2)vbg+{2}(2)vbd+{2}(2)vbn+{2}(2)vbp+{2}(2)sym+{4}general+{2}nnps+{2}pronouns+{3}vbz+{2}(2)roots+{2}(5)pos, 0.9256924178258624
C.2 Argumentation Subset 10 Best Feature Sets

The argumentation subset ten best feature sets were the ten feature sets from the last generation from the argumentation subset experiment with the best f-measures.

\[
\{2\} \{5\} \text{general}+\{4\} \text{in}+\{2\} \{5\} \text{ls}+\{2\} \{3\} \text{nouns}+\{3\} \text{adverbs+vbd}+\{2\} \text{vb} \text{d}+\text{vbp}+\{2\} \{4\} \text{vbn}+\{4\} \text{nnp}+\{2\} \{3\} \text{nnp}+\text{rbs}+\{5\} \text{jj}+\{2\} \{5\} \text{vbdb}+\{2\} \{3\} \text{to}+\{2\} \text{ex}+\{2\} \{4\} \text{ontology}+\text{roots}+\{2\} \{2\} \text{cc}+\{4\} \text{conjunctions}+\{3\} \text{in}+\{2\} \{5\} \text{md}+\{2\} \{3\} \text{jj}+\{2\} \{4\} \text{rp}+\{3\} \text{jj}+\{2\} \{2\} \text{fw}+\{2\} \{2\} \text{to}+\{2\} \text{pdt}+\text{nnp}+\{2\} \{3\} \text{md}+\{2\} \{5\} \text{vbp}+\{2\} \{2\} \text{vb}+\{2\} \{3\} \text{vbg}+\{2\} \{2\} \text{sym}+\{4\} \text{general}+\{2\} \{2\} \text{nouns}+\{4\} \text{pdt}+\{2\} \text{md}+\{4\} \text{cc}+\{2\} \{2\} \text{vbg}+\{2\} \{3\} \text{general}+\{2\} \text{nnps}+\{2\} \text{pronouns}+\{3\} \text{vbb}+\{2\} \text{roots}+\{2\} \{5\} \text{pos}, 0.82909090909090909
\]

{2\} \{4\} \text{jj}+\{2\} \text{dt}+\{2\} \{4\} \text{pronouns}+\{2\} \text{conjunctions}+\{2\} \{2\} \text{nnp}+\{2\} \{4\} \text{fw}+\{2\} \{3\} \text{nnps}+\{2\} \text{adverbs+wtd}+\{2\} \{4\} \text{poss}+\text{vbz}+\{2\} \{4\} \text{in}+\{2\} \{5\} \text{ls}+\{2\} \{3\} \text{nouns}+\{3\} \text{adverbs}+\{2\} \text{vbd}+\text{vbp}+\{2\} \{4\} \text{vbn}+\{4\} \text{nnp}+\{2\} \{3\} \text{nnp}+\text{rbs}+\{5\} \text{jj}+\{2\} \{5\} \text{vbdb}+\{2\} \{3\} \text{to}+\{2\} \text{ex}+\{2\} \{4\} \text{ontology}+\text{roots}+\{2\} \{2\} \text{cc}+\{4\} \text{conjunctions}+\{3\} \text{in}+\{2\} \{5\} \text{md}+\{2\} \{3\} \text{jj}+\{2\} \{4\} \text{rp}+\{2\} \{2\} \text{to}+\{2\} \text{pdt}+\text{nnp}+\{2\} \{3\} \text{md}+\{2\} \{5\} \text{vbp}+\{2\} \{2\} \text{vb}+\{2\} \{2\} \text{sym}+\{4\} \text{general}+\{2\} \{2\} \text{nouns}+\{2\} \{2\} \text{nouns}+\{2\} \{3\} \text{general}+\{2\} \text{nnps}+\{2\} \text{pronouns}+\{3\} \text{vbz}+\{2\} \text{roots}+\{2\} \{5\} \text{pos}, 0.7771428571428571

\[
\{2\} \{4\} \text{cd}+\{5\} \text{rbs}+\{2\} \text{dt}+\{5\} \text{cc}+\{2\} \text{ls}+\{2\} \{3\} \text{adjectives+jjr}+\{2\} \{2\} \text{nn}+\{2\} \{4\} \text{poss}+\{2\} \{4\} \text{md}+\{2\} \{2\} \text{wp}+\{2\} \{5\} \text{prp}+\{5\} \text{vbg}+\{2\} \{5\} \text{rbs}+\{2\} \{5\} \text{unigrams}+\{2\} \{5\} \text{cd}+\{2\} \{2\} \text{wp}+\{2\} \{5\} \text{prp}+\{5\} \text{vbg}+\{2\} \{5\} \text{unigrams}+\{2\} \{2\} \text{wp}+\{2\} \{5\} \text{rbs}+\{2\} \{5\} \text{unigrams}+\{2\} \{5\} \text{cd}+\{2\} \{2\} \text{wp}+\{2\} \{5\} \text{unigrams}+\{2\} \{3\} \text{ontology}+\text{roots}+\{2\} \{2\} \text{cc}+\{4\} \text{conjunctions}+\{3\} \text{ex}+\{2\} \{4\} \text{ontology}+\text{adverbs}+\{3\} \text{prp}+\{2\} \{3\} \text{prp}+\{2\} \{2\} \text{to}+\{2\} \text{pdt}+\text{nnp}+\{2\} \{3\} \text{md}+\{2\} \{5\} \text{vbp}+\{2\} \{2\} \text{vb}+\{2\} \{2\} \text{sym}+\{4\} \text{general}+\{2\} \{2\} \text{nouns}+\{4\} \text{pdt}+\{2\} \text{md}+\{4\} \text{cc}+\{2\} \{2\} \text{vbg}+\{2\} \{3\} \text{general}+\{2\} \text{nnps}+\{2\} \text{pronouns}+\{3\} \text{vbz}+\{2\} \text{roots}+\{2\} \{5\} \text{pos}+\{5\} \text{pdt}, 0.45569294178258624
\]
bs+[5])jjs+{2})5jjs+ontology+{3}domain+{3}unigrams+{3}to+[4]w
0.4529946457477275

0.4527500000000004

0.4527500000000004
C.3 Arguments  All 10 Best Feature Sets

The arguments-all 10 best feature sets were the ten feature sets from the last generation from the arguments-all subset experiment with the best f-measures.
The alternatives ten best feature sets were the ten feature sets from the last generation from the alternatives subset experiment with the best f-measures.

{2} (2) nn+(4) cd+(5) rbs+(2) dt+(5) cc+(2) ls+(2) (3) adjectives+jjr+{2} (2) nn+(2) poss+(2) (4) md+(2) (2) wp+(2) (5) prp+(4) prp+(5) vbg+(2) r p+(5) unigrams+(2) (2) cd+(2) nn+(2) (5) wp$+nns+(4) f w+(3) ontology+{2} (3) ontology+adverbs+(3) prp+(2) r p+fw+vbz+(2) vbg+(4) to+(2) rb+(5) general+(4) in+(2) r p+(5) ls+(3) rbr+(2) (3) rbr+(3) rbs+(3) poss+(2) (3) adverbs+(3) c c+(2) (3) cc+vb+(2) (2) nns+(2) (3) roots+(2) ex+(2) (2) ex+(2) in+(2) r p$+(2) (4) nouns+(4) verbs+pos+(2) wp$+(2) (5) ontology+(3) nn+(2) rbs+(5) j j+(2) jjs+ontology+(3) domain+(3) unigrams+(3) to+(4) wdt+(2) (4) wdt+(2) (4) nn+(4) sym+(5) wp+(2) (4) unigrams+(2) (2) adverbs+(3) wp+(2) roots+(4) ex+(2) (4) ex+(2) wbr+(3) vbn+(4) conjunctions+(2) (5) m d+(2) domain+(3) j j+(2) (3) j j+(4) rp+(5) vbn+(2) wp+(2) (2) prp+(2) vb+{2} (3) ex+(2) (5) nn+(2) (2) nn+(2) (4) adjectives+{2} (2) jjr+nnp+(2} (2) pronouns+(4) dt+(5) r b+(2) (5) rbr+(2) (5) sym+(2) (2) v b+(2) unigrams+(4) j j+(2) (4) pr p+(2) (2) v b+(2) (2) sym+v b+(2) (3) ls+(4) general+(2) (4) general+(4) v b g+(2) poss+(2) (2) poss+(2) (4) dt+(2) md+(2) ls+(2) (4) v b p+(2) general+(5) verbs+(2) (2) pos+(2) nn+(2) pronouns+(3) v bz+(2) roots+(2) (2) roots+(2) domain+(5) wdt+(2) nouns+(2) rbs+(2) adverbs+(2) r b+(2) (3) pronouns+(4) r bs+(2) (4) r bs+(5) pos+(3) r b+(2) (3) r b+(3) verbs|{2} C_sentLength, 0.3123043712898003

(4) cd+(5) r bs+(2) dt+(5) cc+(2) ls+(2) (3) adjectives+jjr+(2) nn+(2)
C.5 Decisions 10 Best Feature Sets

The decisions ten best feature sets were the ten feature sets from the last generation from the decisions subset experiment with the best f-measures.
cd+pronouns+{2} (5) jjr+{2} (3) vbz+ (2) roots+{2} (2) roots+adjective s+(3) wrb+ (5) roots+{2} (4) rbs+ (5) dt+ (3) rb+ (3) verbs+{2} (3) verbs, 0.7355176933158584

(4) cd+ (4) jjs+ (5) cc+ {2} (2) ls+ (2) vbz+ rb+ {2} rb+ {2} jjr+ {2} poss+ {2} (4) pronouns+ {4} prp+ {2} (4) domain+ (5) cd+ (5) in+ (3) jjs+ (5) wp$+ nns+ {2} nns+ (3) prp+ (3) wdt+ {2} uh+ {2} (4) poss+ vbz+ {2} (5) general+ {2} (4) in+ {2} (5) vb+ {2} (3) rbr+ {2} (3) rbs+ to+ {2} general+ vb+ {2} (2) vb+ {2} md+ (3) ls+ (4) general+ {2} nouns+ {2} (2) noun+ {2} poss+ {2} (2) poss+ {2} md+ (2) (2) md+ rbr+ {2} (2) cd+ pronouns+ {2} (5) jjr+ {2} (3) vbz+ {2} (2) roots+ adjectives+ (3) wrb+ {2} (5) roots+ {2} (4) rbs+ {2} (5) dt+ (3) verbs+ {2} (3) verbs, 0.72416666666666666

C.6 Full Feature Set List

Only half of the features are shown here, the other half are the same features with {2} appended to the front. (The {2} represents 2-closest sentences.)

Penn-Treebank features: [Marcus, et al., 1993]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>cd</td>
</tr>
<tr>
<td>jj</td>
<td>jjr</td>
</tr>
<tr>
<td>nnp</td>
<td>nnps</td>
</tr>
<tr>
<td>prp</td>
<td>rb</td>
</tr>
<tr>
<td>to</td>
<td>uh</td>
</tr>
<tr>
<td>vbp</td>
<td>vbz</td>
</tr>
<tr>
<td>(2) cc</td>
<td>(2) cd</td>
</tr>
<tr>
<td>(2) jj</td>
<td>(2) jjr</td>
</tr>
<tr>
<td>(2) nnp</td>
<td>(2) nnps</td>
</tr>
<tr>
<td>(2) prp</td>
<td>(2) rb</td>
</tr>
<tr>
<td>(2) to</td>
<td>(2) uh</td>
</tr>
<tr>
<td>(2) vbp</td>
<td>(2) vbz</td>
</tr>
<tr>
<td>(3) cc</td>
<td>(3) cd</td>
</tr>
</tbody>
</table>

59
General parts of speech features:

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjectives</td>
<td>jj</td>
</tr>
<tr>
<td>Adverbs</td>
<td>jjr</td>
</tr>
<tr>
<td>Conjunctions</td>
<td>jjs</td>
</tr>
<tr>
<td>Nouns</td>
<td>ls</td>
</tr>
<tr>
<td>Pronouns</td>
<td>md</td>
</tr>
<tr>
<td>Verbs</td>
<td>nn</td>
</tr>
<tr>
<td>Adjectives</td>
<td>jj</td>
</tr>
<tr>
<td>Adverbs</td>
<td>jjr</td>
</tr>
<tr>
<td>Conjunctions</td>
<td>jjs</td>
</tr>
<tr>
<td>Nouns</td>
<td>ls</td>
</tr>
<tr>
<td>Pronouns</td>
<td>md</td>
</tr>
<tr>
<td>Verbs</td>
<td>nn</td>
</tr>
</tbody>
</table>

Rationale terms: [AuditMyPC, 2010] and [Burge, 2005]

general domain ontology (2)general (2)domain (2)ontology
(3)general (3)domain (3)ontology (4)general (4)domain (4)domain
(5)general (5)domain (5)ontology

Verb roots: [Miller, 1995]

roots (2)roots (3)roots (4)roots (5)roots

Parts of Speech: [Marcus, et al., 1993]
pos (2)pos (3)pos (4)pos (5)pos

Bag of Words:
unigrams (2)unigrams (3)unigrams (4)unigrams (5)unigrams

Sentence Length:
C_sentLength

**Appendix D Pipeline Instructions**

The following pipeline instructions are taken directly from Miami University’s Rationale Extraction from Existing Documents research group wiki.

All right, so you want to use the pipeline, but not have to read a book to do it. Don't worry, I've got you covered. Follow the instructions below, and you should be getting results in no time.
1) Checkout a copy of the svn repository [https://svn.csi.org/burgeje/WEKA/Pipeline/](https://svn.csi.org/burgeje/WEKA/Pipeline/). (If not working try: [https://svn.csi.miamioh.edu/burgeje/WEKA/Pipeline](https://svn.csi.miamioh.edu/burgeje/WEKA/Pipeline))

2) Open pipeline.sh in you favourite text editor. (Ok, it doesn't need to be your favourite, but why would you use anything else?)

3) In the Set variables section
   a) Set validAnnoStr equal to the attributes signifying existence of what you want to classify.
   b) Set featSet equal to the feature set you would like to use for classification. Specifics can be found [here](https://svn.csi.org/burgeje/WEKA/Pipeline/). (Unless you're doing something fancy, this means feature or feature1+feature2 etc.)
   c) Set sentWeight equal to the highest sentence weight used in featSet. If you don't know what that means, set it to 1.
   d) Set wekaPath equal to the full path to the weka.jar file you're using. If you haven't downloaded Weka yet, do it! You're going to need it.

4) Save and close pipeline.sh.

5) Open the GATEXMLs directory.

6) If there's anything in there, delete it, and put the XML files that you want to classify (generated by GATE) in here, and go back to the Pipeline directory.
   a) **IMPORTANT** - If you have more then 10 XML files go back to pipeline.sh and add the following line between
   ```
   echo $(java SentencesFileGenerator $FSetsDir $featSet $validAnnoStr) > temp/$index.tmp &
   ```
   ```
   if (( $index % 10 == 0)); then wait; fi
   ```
   This will prevent fifty-bajillion processes trying to run at the same time.

7) Open the Classifiers directory.

8) If there's anything in there, don't delete it yet.
   a) Open them up, look at them, and try to figure out what's going on. Or you could try this [page](https://svn.csi.org/burgeje/WEKA/Pipeline/).
   b) Create a new file that will run the classifier you need, or edit a file already there.
   c) Delete all of the other files in the directory and go back to the Pipeline directory.

9) Open up the FeatureSets directory.

10) If there's anything in there, don't delete it yet.
    a) Open them up, look at them, and try to figure out what's going on, look at this [page](https://svn.csi.org/burgeje/WEKA/Pipeline/) for help.
    b) Create new files for features you want, or edit files already there. Make sure all of the features you specified in the featSet variable are here!
    c) Delete all whatever features you want to and you don't need. Extra features here may increase running time, but it'll run correctly.
    d) Go back to the Pipeline directory.

11) Open up the command line terminal.

12) Change the directory to the Pipeline directory.

13) Run pipeline.sh (You might need to run ./pipeline.sh)

14) Wait for it to run. It will take a while.
   a) First, the XML files will be processed, 10 at a time. The shell will tell you which files are being processed. This takes a while
   b) Second, the sentence files will be joined, and you will be notified
   c) Third, the ARFF files will be created, and you will be notified
   d) Forth, the ARFF files will be filtered and reconciled, and you will be notified
e) Fifth, the tests will be run. You will see the text from the file you made in the Classifier directory, as well as possible "Discretising Data Set" notifications.

f) Finally, when everything is done, you'll get the message "Finished Running Tests"

15) After it has finished running,
   a) Open up the temp directory.
   b) Each of the "results.txt" files will have line by line results of the classification of each of the folds. Currently no calculations are presented.