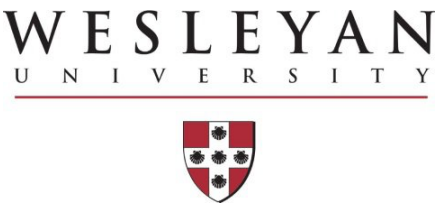

MINING DESIGN RATIONALE FROM SOFTWARE DOCUMENTATION USING NATURE INSPIRED METAHEURISTICS

By

Miriam Lester

Faculty Advisor: Dr. Janet Burge

A Dissertation submitted to the Faculty of Wesleyan University in partial fulfillment of the requirements for the degree of Master of Arts in Computer Science

Middletown, Connecticut **May 2017**

[*a leaf falls in front of one of the worker ants in the food line*]
Worker Ant #1: I'm lost! Where's the line? It just went away. What do I do? What do I do?
Worker Ant #2: Help!
Worker Ant #3: We'll be stuck here forever!
Mr. Soil: Do not panic, do not panic. We are trained professionals. Now, stay calm. We are going around the leaf.
Worker Ant #1: Around the leaf. I-I-I don't think we can do that.
Mr. Soil: Oh, nonsense. This is nothing compared to the twig of '93.
– A Bug's Life

Acknowledgements

First and foremost I would like to give a huge thank you to my wonderful advisor and mentor, Janet Burge. Thank you for agreeing to advise me remotely from Colorado, dealing with time differences, and putting up with many dropped Skype calls. I would not have gotten through this process without your patience, guidance, and support. Getting to work on this research has been so rewarding. I am so appreciative that you brought me on board and had enough confidence in me to continue spending time helping me grow as a researcher, even though you were not at Wesleyan! Thank you also to Connor Justice for helping me to get started with this research, and having so much enthusiasm for the project that you continued to find ways to stay involved.

I also have a deep gratitude for my undergraduate advisor, James Lipton, who let me into his class when I announced on the first day of freshman year that I needed a spot in his introductory computer science class because I was going to graduate in three years. Never mind that I had never taken a computer science class before, and I obviously had no idea what it would take to graduate in three years, he vouched for me without hesitation. Thank goodness the excuse worked, because as it turns out I wouldn't have reached this point otherwise. Thank you for teaching me so much about computer science, and for taking the time to read my thesis. Thank you also to Daniel Krizanc for agreeing to be on my panel, and for coming all the way back to Wesleyan for my defense. I am extremely appreciative of the whole Wesleyan Computer Science department and all of the professors who helped me along the way.

Thank you to the American Association of University Women, for honoring me with the Selected Professions Fellowship. The funding really helped me to be able to pursue this Masters.

Finally, thank you to my family for the overwhelming the support throughout my whole life. I was so fortunate to grow up in a family that valued education so highly, and who made becoming a scientist look like the coolest job ever. When I was little, one of the first math concepts I learned from my family was “What is the square-root of a negative pancake? An imaginary pancake!” Later, my grandpa showed me what fun it was to program computers to play pranks on people, and I never could have guessed how much his love of computers would wear off on me. You’ve inspired my lifelong love of learning, and for that I am so thankful! Thanks to Aaron for being the best brother and friend anyone could ask for! Thanks to my parents Marc and Amy for pretending they understand metaheuristics. This thesis is dedicated to them.

Abstract

Design rationale refers to the explicit documentation of the reasons design decisions are made, the alternative design options considered, and the justification supporting these decisions. If properly recorded during the software development life cycle, design rationale documentation can be extremely useful to software engineers. However, because of the costs associated with maintaining extra documentation, it is often omitted. Fortunately, design rationale is naturally embedded in other documents that are created in the general course of software development - such as bug reports, project specifications, emails, or meeting transcripts. I will introduce the text mining problem of automatic extraction of design rationale from such software documentation, and will describe the machine learning based approach to capturing the rationale. I will discuss the feature selection step of the machine learning method, and give background on the genetic algorithm and ant colony optimization metaheuristic chosen in this research for the feature selection procedure. I will show that by applying these text mining techniques to classification of design rationale in software documentation, we are able to achieve higher F-measure than our estimate of what would be achievable manually.

Table of Contents

List of Tables

Chapter 1 Introduction

Design rationale refers to the explicit documentation of the reasons design decisions are made, the alternative design options considered, and the justification supporting these decisions [1]. If properly recorded during the software development life cycle, design rationale documentation can be extremely useful to software engineers. However, because of the costs associated with maintaining extra documentation, it is often omitted. Fortunately, design rationale is naturally embedded in other documents that are created in the general course of software development [2]. This chapter will outline the problem of automatic extraction of design rationale from such software documentation. We will describe the machine learning based approach to capturing the rationale. We will then introduce a feature selection step of the machine learning method, and give background on the genetic algorithm and ant colony optimization algorithm chosen in this research for the feature selection procedure. Finally, we will outline the key questions guiding this research project.

1.1 Problem Description and Motivation

Software engineers use design rationale to provide a comprehensive record of the reasoning behind the decisions made during the software design process. For example, the rationale might include all of the alternative designs considered in the design process, the questions and answers discussed to come to a conclusion about which design would work best, and the ultimate decision of which design to go with.

It also could elucidate a specific assumption that was presumed by an engineer to support the decision of choosing one software design pattern over another, and the requirements of the project that informed the design decision. Rationale contains valuable information for someone joining a software development team after significant code has already been implemented. For example, design rationale helps an engineer continue to develop a project the way the original designers had intended and allows the final product to be compared to project specifications to verify that all development requirements of the project have been satisfied. However, most companies do not currently incorporate the recording of design rationale into their documentation practices as it is expensive and time-intensive [3].

Even when design rationale is not explicitly recorded, companies often incidentally record key information in documents such as bug reports, project specifications, emails, or meeting transcripts [2]. Design rationale could theoretically be manually extracted from these text documents, but in practical terms, the time and energy required would be prohibitive—likely more than the costs involved in explicitly recording the design rationale in the first place. Thus, tools that could automatically extract relevant design rationale out of existing software documentation would be extremely valuable. This thesis focuses on the problem of automatically capturing design rationale from existing text documentation, with the aim of making design rationale documentation available to software teams in a cost-effective manner. This thesis will compare the extraction of a few different types, or subclasses, of design rationale including alternatives, arguments, assumptions,

decisions, requirements, questions, answers, and procedures. We focus on the *argumentation* subset of rationale subclasses – where alternatives, arguments, assumptions, decisions, and requirements are considered valid rationale. We also look at *arguments-all* subset of rationale subclasses – where requirements, arguments, and assumptions are considered valid rationale, and the *binary rationale* classification – where all rationale subclasses are considered valid rationale.

1.2 Approach

The problem of automatically extracting design rationale from software documentation can be tackled using supervised learning. Generally, a supervised learning approach to machine learning takes labeled data, builds a model of the data, and uses the model to predict classification for unlabeled data. One example of such a machine learning method is the spam filter in email. As more emails are marked with the categories of "spam" and "not spam" in an inbox, a classifier refines the way it recognizes patterns of emails that are marked spam. When the classifier encounters new, unlabeled mail, it can guess the spam label to assign the email based on the patterns it has previously determined. Once the spam filter has been trained, it can handle the marking of spam emails automatically. In this way, only a small fraction of the emails must initially be labeled as spam, and the automatic spam filter classifies the rest.

We can apply this principle to any text mining problem. First, construct a corpus of labeled sentences. Then extract features from each sentence to create a *feature set* for each sentence: either use the bag-of-words model, where each word in

the sentence is a feature for that sentence, or extract other information from the structure, grammar, and phrases in the sentences. For example, if we extract the parts of speech of each term in the software documentation, then we can use all nouns in a sentences as features of the sentence. We could also use the presence of a noun-verb phrase in the sentence as a feature. Next, split labeled feature sets representing each sentence into training and test sets. Then use the training set to build a classifier model. Finally, use the model to predict labels of the other test sentences. By evaluating metrics about how many of the predicted labels were correct, we can deduce how suitable the model is.

Metrics recall, precision, F-measure, and accuracy are most commonly used to determine how well a model generated with supervised learning can predict the expected labels. These performance measures are given by the following equations:

Precision = True positives out of total positive predictions

Recall = True positives out total true instances

F-measure = Harmonic average of precision and recall = (2*P*R)/(P+R)

Accuracy = True predictions out of total predictions

To gain stronger confidence that the model created is in fact representative and sufficiently general, k-fold cross validation is used [4]. With this technique, the data are split into k parts and the evaluation is repeated k times, with each part being used once as a test set and the respective other k-1 parts being used as a training set. The result is the average of the relevant statistic evaluated for each of the k models. This method is useful in that it makes it more likely the results will not be biased by a

specific split of training/test data, especially when the quantity of data is limited. This also is beneficial in preventing overfitting, which is the problem of a model being generated that too closely fits to the particular instances in the training set, and is not general enough to constitute a strong predictive model for new unlabeled instances. This might occur if the statistical model is fitted to the random error within the training set or is overly complex, instead of capturing the strongest deciding features of the pattern.

To implement automatic capture of design rationale from software documents, we train a classifier with a set of annotated software documents – ones labeled to denote which sentences are rationale. Once the classifier creates a model to determine if an instance is rationale, the classifier can predict rationale classification of non-annotated documents. This paper continues previous work on automatically extracting design rationale from software development documentation [5] through the application of these statistical machine learning techniques which classify textual features in documentation.

1.3 Feature Selection

The *feature space* refers to all possible combinations of the features available, where each feature is considered one dimension of the space. One of the greatest challenges in classifying text with machine learning techniques is that there can be thousands of textual features to analyze [6]. There are a potentially unlimited range of possible text features – for example, they might include nouns, verbs, unigrams, bigrams, occurrence of the word “computer”, whether the sentence ends in a vowel,

etc. In this research, we also distinguish between categories of features and instances of features. If we are classifying a sentence "I chose to use HTML." A feature category could be "verbs," and the feature instances would then be the specific verbs in the sentence, in this case "chose" and "use." Or the category could be more domain specific, like acronyms, and the instance in this sentence would be "HTML." For reference of size, this study considers 723 unique text feature categories, with hundreds of thousands of feature instances.

In such a highly dimensional feature space, features might be irrelevant, redundant, noisy, or generally detrimental to the performance of the classifier [7]. Therefore, *feature selection,* also known as *feature reduction,* is an important step in machine learning. The purpose of feature selection is to identify the most relevant, essential features that can classify the data without hurting the classifier accuracy. In many cases, selecting the optimal feature subset impacts the classifier efficiency and accuracy significantly. It is too computationally expensive to conduct a *brute-force* or *exhaustive search* (systematically trying every possible combination of features) over the feature space to find the optimal subset of features; thus many feature selection techniques have been developed to search for favorable feature subsets [8].

Most commonly, feature sets are selected with the statistical techniques of information gain, mutual information, document frequency, chi squared, and term strength [7]. More recently, nature inspired optimization algorithms such as the genetic algorithm and the ant colony optimization algorithm have become increasingly popular. These algorithms, though more computationally complex, have

the advantage of utilizing knowledge from previous iterations and introducing randomness to more broadly search the feature space to find a more optimal feature set [7].

1.4 Metaheuristics for Combinatorial Optimization Problems

A combinatorial optimization problem involves searching for some element in a finite set that maximizes or minimizes a constraint. In mathematical terms, an instance of a combinatorial optimization problem is a triple (S, f, Ω) such that S is a set of all possible solutions, f is an objective function which evaluates to a discrete value f(s) for each possible solution $s \in S$, and Ω is a set of constraints of the problem. A globally maximum feasible solution to (S, f, Ω) would be a solution s' that (1) satisfies constraints Ω and (2) $f(s') \geq f(s)$ for all s that also satisfy Ω [9]. Combinatorial optimization problems have significant real-world relevancy, including but not limited to such applications as packet routing on the internet, job scheduling, cutting-stock to minimize waste, the knapsack packing problem, the traveling salesperson problem, and finding the minimum spanning tree on a graph. The feature selection step for improving the classification model in automatic design rationale classification is an instance of a problem of selecting an optimal object (namely a subset of features) from a finite set of objects (the set of all subsets of features). Thus, this feature subset selection problem for the classification task can be stated as a combinatorial optimization problem: "Given the original set, F, of *n* features, find

subset S, consisting of m features ($m < n$, $S \subset F$), such that the classification accuracy is maximized" [10].

Unfortunately, many combinatorial optimization problems (including this feature reduction problem) are NP-hard, meaning it is widely believed there is no known polynomial time solution to this class of problems. As the instance size grows, the size of the set of candidate solutions, and therefore the amount of time it takes to find the optimal solution, grows exponentially. Since an exhaustive search to find the solution to combinatorial optimization problems is often infeasible, the general method to approach these problems is to sacrifice optimality for a reduction in runtime. Near-optimal solutions to instances of combinatorial optimization problems can be found with approximation algorithms also called heuristics. Heuristic algorithms use information about the problem and approximation methods to compute a sub-optimal solution in a practical amount of time. The greedy construction heuristic approximation algorithm iteratively adds components until a solution is considered complete [9]. The iterative improvement method searches for a better solution within the neighborhood of the current solution, replacing the current solution with the better solution until no improving solution is found [9]. These two methods tend to be unable to find high quality optima because they get stuck in local areas [9].

A class of algorithm that is better at broadly covering the search space is called the *metaheuristic* [9]. A metaheuristic is an algorithm that is used to guide a heuristic search towards a more optimum solution in the search space. Dorigo, the

computer scientist who developed the ant colony optimization algorithm [11], noted, "The use of metaheuristics has significantly increased the ability of finding very high quality solutions to hard, practically relevant combinatorial optimization problems in a reasonable amount of time" [9]. The algorithmic concepts outlined by the metaheuristic can apply broadly to many types of problems, making a metaheuristic approach very appealing.

1.4.1 Ant Colony Optimization

Everyone from young children to senior biologists has been fascinated by the peculiar behavior of ants and other swarming insects. Although individual actors cannot achieve much on their own, together in a hive or colony these seemingly simple organisms can attain impressive social organization and solve challenging, sophisticated problems. One example of this complex problem solving is the incredible ability of a colony of ants to solve a graph theory problem called "the minimum cost path problem" [9]. In 1990, Deneubourg et al. showed that ants use pheromone trails to find the shortest path between their nest and a food source [12]. He explores how this collective pattern can emerge from what is referred to as an *autocatalytic* or positive feedback process between foragers. This process of exploration and self-organization is moderated by *stigmergy* – the process by which simple agents indirectly communicate through their environment [9]. In this case, the ants build and reinforce the shortest path without planning, control or sight by laying

down pheromone chemicals that can be followed by other identically genetically programmed agents.

Further, in what became known as the double bridge experiments [13], it was shown that the ants could navigate their environment to collectively avoid obstacles, and reliably find the shortest path between their nest and food. The experiment laid out two possible paths for the ants to follow to get from their nest to a food source, with one path being twice as long as the other. While initially the foragers chose equally between the long and short paths to the food, after only a few minutes most of the ants were taking the shorter path because they followed the pheromones other ants deposited to create a positive feedback loop. This impressive observation inspired Marco Dorigo to show that this natural process could be modeled by the computer simulation of an ant colony. This method became known as the ant colony optimization (ACO) metaheuristic [11]. Dorigo combined the properties observed in real ants with a pheromone evaporation rule to increase the performance of the search procedure in an algorithm called the Simple-ACO. This was used to approximate the solution to the minimum cost paths problem [9]. Consequently, many different variations of the ACO were developed, including a version of the ACO for subset problems [14]. Thus ACO can be applied to our feature subset selection problem, by having each ant construct a subset of features – choosing features based on pheromone trails applied directly to the feature components instead of connections between components, and considering some measure of heuristic desirability of each feature.

1.4.2 Genetic Algorithm

A little over a hundred years after Darwin published his theory of evolution and survival of the fittest in *Origin of Species*, the Genetic Algorithm based on these natural processes was formalized by John Holland [15], who realized it could be possible to exploit the adaptive mechanics of natural systems in artificial software systems. Holland wrote, "GA's offer robust procedures that can exploit massively parallel architectures and, applied to classifier systems, they provide a new route towards an understanding of intelligence and adaptation." [16] The codification of the idea of imitating sexually reproducing organisms by the process of mating and mutating possible solutions to optimization problems [15], was a breakthrough in the evolutionary algorithmic field, motivating an enormous growth in the popularity of this branch of artificial intelligence.

The Genetic Algorithm (GA) is a population based search method, inspired by the principles of natural selection and survival of the fittest. Individuals representing candidate solutions to the optimization problem are maintained throughout multiple iterations called generations. During each generation, individuals are evaluated for fitness – the quality of a solution to the combinatorial optimization problem. Next, the "survival of the fittest" is mimicked by the highest scoring individuals being retained as "parents" for the next generation, while the rest of the chromosomes are not preserved. To evolve the population, pairs of parental chromosomes are selected and biologically imitating crossover and mutation operators are applied to produce chromosome "offspring." The next generation then begins with the new population of

both previous parents and created offspring chromosomes. Generations are run until a stopping criteria is met, at which time the best chromosome(s) are returned as potentially optimal solutions to the problem [17]. The GA can be applied to the feature selection problem by having chromosomes represent candidate subsets of features, and using the F-measure of classifying sentences with only features from the selected subset as measure of fitness.

1.5 Research Questions

This study focuses on answering four main questions:

1. **What parameters are best for selecting the optimal feature set with the ant colony optimization algorithm?** In the ant colony optimization algorithm for feature selection there are many parameters that can modify the search of the feature space, affect speed of convergence, or shift the balance of local feature importance and pheromone trail intensity. These values are specific to the dataset being analyzed, so this study shows which initializations work best for finding the optimal feature set to classify rationale for design rationale capture.
2. **What textual feature categories do the ant colony optimization algorithm and genetic algorithm select?** The optimal feature set gives the best classification with the fewest features possible to both reduce the computational complexity and increase the accuracy of the classifier. Specific feature categories will be chosen to result in the strongest classifying power that can be used to capture rationale of future unannotated documentation.

3. **How does the ant colony optimization selected feature set compare to the genetic algorithm selected feature set for classification?** This study will compare the average F-measure, precision, and recall of classification in ACO and GA selected feature sets, to determine which methodology is best for selecting an optimal feature set for rationale extraction.
4. **How generalizable are models created for a specific dataset?** This research will compare models created from analysis of two separate datasets, to see if the features chosen to optimize extraction on one dataset are effective in capturing rationale from the other dataset.

1.6 Outline

Two datasets were used so generalizability of feature selection technique could be assessed. The first dataset was Google Chrome Bug Reports (BR), which was made up of 200 Chrome Bug reports from the data available for the Mining Software Repositories Challenge (http://2011.msrconf.org/msr-challenge.html) [5]. More detailed information about how the dataset was compiled can be found in Rogers' thesis [5]. The second dataset was the Studying Professional Software Design (SPSD) transcripts, which consists of transcripts of teams designing traffic simulators. Further information about how the SPSD dataset was constructed can be found in Mathur's thesis [18].

These datasets are annotated with rationale from the following categories [19]:

- Requirements: statement indicating obligation for designer to do something.
- Decisions: statements indicating issue that was settled.

- Alternatives: different actions for resolving the issue described by a decision.
- Arguments: reasons supporting or refuting an alternative.
- Assumptions: supposition contributing to reason for making a decision.
- Questions: questions asked.
- Answers: answers to questions asked.
- Procedures: steps described for answering a question or making a decision.

Rogers employed a Genetic Algorithm for Feature Selection (GAFFS) to search for the optimal feature set to classify design rationale in a software documentation textual corpus [5]. This study will apply an original ACO algorithm and an updated GA algorithm to the software design rationale text classification problem on the same corpus dataset as Rogers, and compare the classification results of the text with feature sets selected by the updated GA and ACO respectively. The ACO and new GA achieve better F-measures for classification with less computational effort and complexity than GAFFS.

The following chapters are organized as follows: Chapter 2 lays foundation in text mining and relevant current research, Chapter 3 describes the ant colony optimization metaheuristic and genetic algorithm as well as details of their respective implementations, Chapter 4 discusses the system design, Chapter 5 describes the experimental design, Chapter 6 indicates the results of the experiments, Chapter 7 describes extensions for future work, and Chapter 8 concludes.

Chapter 2 Background: Text Mining and Related Work

In this chapter is a detailed explanation of text mining, feature selection methods, and current approaches for applying ACO and GA to feature selection problems in text mining.

2.1 Knowledge Discovery from Text

The explosion of the internet has resulted in an incredible amount of available text data. From news articles to restaurant reviews, almost any textual corpus can be found online. This wealth of information can never be fully processed, analyzed, or utilized manually. In light of this, linguists and computer scientists have begun to work together to mine this information through *natural language processing* (NLP). NLP algorithms are used to analyze patterns and extract relevant information from text written for a human audience. A branch of this field, *knowledge discovery from text*s, is a process that uncovers patterns in text document corpuses and evaluates how beneficial this knowledge is [20]. With this combination of natural language processing and machine learning, many different problems can be undertaken.

Text mining – combining machine learning and NLP – is a relatively new and quickly expanding field. It was not until September 2016, 10 years after the launch of the tool Google Translate, that the research team at Google announced they finally could outperform the "conventional phrase-based translation system" with a new model built on machine learning based NLP [21]. What had previously been state-of-the-art technology became replaced with a much better system that could now

translate sentences in their entirety, and seemingly capture the nuances and art of native language. The machines obviously could not *understand* the language they were producing, but could clearly learn complex patterns of natural language, and extract information that is valuable to people all over the world.

Yet another challenging NLP task is the problem of sentiment analysis. While it is easy for a person to distinguish between a positive or negative Amazon review of a product they are interested in purchasing, it is more challenging for a computer to interpret such a distinction. You could construct a model which considers any review containing the word "good," to be a positive review, but then the simple model would fail trying to distinguish that a review saying, "this product was not good" had negative tone. Returning to our discussion of supervised machine learning from section 1.2, we can understand that to build a highly predictive model we need many positive and negative examples of labeled data-points to learn over, and we must extract from each data-point the most strongly distinguishing features possible.

Researchers have considered many different techniques to represent and extract meaning from natural language texts. Atkinson-Abutridy et al. [20] used a two-step process in which they first extract a hypothesis representation of each document in a technical and scientific natural language corpus. Then, they used a GA to select the best hypotheses to represent all the data. The model they built was noteworthy because it did not rely on any outside lexical resources like WordNet or specific concept resources like thesauri. Instead, it utilized only information from text corpus and from data computed directly from these primary documents.

2.2 Text Mining Rationale

Identifying design rationale in software documentation is an even more nuanced problem than sentiment classification, and it is often harder to intuit which features in the text might be highly relevant. Since we can indiscriminately extract hundreds of thousands of features from each sentence in the software documentation corpus, the step of determining the specific features that overall have the greatest distinguishing power becomes even more significant. Thus, we focus heavily on the feature selection step. Another feature that makes design rationale extraction a unique text mining problem is the fact that it can be structured. For example, each design decision could come from a choice between multiple alternatives, and the decision could be based on requirement, assumption, and procedure [5]. Throughout this study, I will be looking at capturing the rationale through binary classification rather than preserving or reconstructing the structure.

Mao and Xiao [22] tried to identify rationale in online deliberation text, namely a Wikipedia article for deletion discussion forums. They looked closely at argument rationale, and wanted to specifically identify direct imperative arguments. Their system had F-measure of .7874 on the Wikipedia data.

Moens, et al. [23] worked on the automatic identification of arguments in legal texts. They trained a classifier on a set of annotated legal arguments, and evaluated the results with the multi-nominal naïve Bayes classifier and maximum entropy model. They also utilized a list of keywords that were indicative of argumentation for feature extraction.

Liang, et al. [24] also try to extract design rationale from existing design documents using text mining. For their experiments they utilized a corpus of patent documents, and a layered approach to capture artifacts, summarize issues, and create solution-reason pairs. To identify solution-reason pairs in the text, they separately identify all candidate reasons and solutions in the text. They used a ranking algorithm and shared information between candidate reasons and solutions to achieved an F-measure of .561. They worked towards creating an interface to interact with the automatically captured design rationale, which is how we hope our research would be applied in the future.

Lopez, et al. [2] worked on recovering design rationale from existing software documents, as well as representing the rationale and integrating the rationale with a software tool. Although the latter two aims would be related to future goals of this research, the first is directly related to the experiments in this thesis. They use an ontology based extraction of software design rationale, where the ontologies relates concepts of software architectures and thesauri of relevant terms. Thus they can implement rationale-specific text mining tools. They tested their tool with documents for designs of a security clearing system. With their tool they achieved an F-measure of .5 for recovering rationale. This was stronger than the manual recovery F-measure of .42, and took less time.

Rogers [5] created a genetic algorithm for feature selection, which he applied to the automatic capture of design rationale in software documentation problem. His research laid the foundation for the work in this thesis. The GA proved to be

successful in finding strong feature subsets for classification of design rationale. Because of a bug later found in his work, the F-measure he calculated with respect to classification in the Bug Reports dataset cannot be directly compared. However we do use the same Bug Reports dataset for experiments in this thesis.

Mathur [18] built upon Rogers' work, trying to improve the classification results by focusing on combatting the class imbalance present in the data. The Chrome Bug Report dataset had only 10.9% rationale, and thus was imbalanced. To deal with this, he added new features and the SMOTEBoost algorithm, and found improved F-measure only for binary rationale classification. Mathur introduced the Studying Professional Software Design dataset, which we use in this thesis.

2.3 Feature Subset Selection

There are many methods for selecting the strongest features such that the noisy and irrelevant features do not hurt the classifier performance. Generally, we are trying to pick out the features that show the greatest power in differentiating the classes of data. In this way, the patterns that distinguish each class will be more discernible. A search through the entire feature space would take time equal to:

$$\sum_{s=0}^{n}\binom{n}{s}=\binom{n}{0}+\binom{n}{1}+\ldots+\binom{n}{n}=2^{n}. \quad (2.1)$$

Where n is the number of features and s is the size of the current feature subset [25]. Equation 2.1 shows that an exhaustive search would take exponential time.

Other approaches can decrease the time complexity; however, optimality is not guaranteed. The two main methods of selecting best features are the wrapper and

filter methods. The filter approach includes any method that is done separately from a machine learning algorithm. These algorithms tend to be computationally efficient, and usually are done as preprocessing by calculating statistics about how strongly a feature can distinguish data classification [25]. The wrapper approach uses machine learning classifiers within the evaluation procedure. The search is conducted with each candidate solution's suitability being measured by how well it can be classified with a machine learning algorithm. Wrappers yield better results but are more computationally expensive [25].

There are many statistical methods to evaluate the value of features. Yang and Pedersen [26] made a comparative study of the major corpus statistics – document frequency, information gain (IG), mutual information, CHI-squared, and term strength effectiveness – for dimensionality reduction for text categorization on the Reuters dataset. Yang et al. found experimentally that the CHI-squared and information gain statistics were most effective for feature subset selection, and that term strength was also effective for cases when computational efficiency outweighed a small decrease in efficacy. For this reason, I chose to implement CHI-squared as a heuristic value within my ACO algorithm. In my case, the CHI-squared value is precomputed for each feature before the ACO routine to improve efficiency, and thus it is considered a filter method.

There are four main greedy methods for creating a feature subset [25]:

1) Forward Selection: start with an empty set and greedily add features one by one

2) Backwards Elimination: start with the set of all features and greedily remove features one by one
3) Forward Stepwise Selection: start with an empty set and greedily either add or remove features one by one
4) Backwards Stepwise Elimination: start with the set of all features and greedily either add or remove features one by one.

These types of greedy construction heuristics usually find poor solutions, as they have no backtracking and therefore are more likely to get stuck in inferior local-optima [9].

Metaheuristics can more broadly search the feature space and have an advantage over these greedy heuristics [9]. Common metaheuristic approaches to feature selection include the ACO [11], the GA [15], simulated annealing [27], and particle swarm optimization [28] – though this thesis will implement and compare only ACO and GA. The next three subsections will present current approaches to using ACO, GA, and metaheuristic hybrids.

2.4 Ant Colony Optimization

Many variations of ACO algorithms, which are currently being investigated in feature selection for classification problems, are having success over traditional methods. Al-Ani [10] used ant colony optimization search procedure to select feature sets for speech segment and image texture classification. ACO performance was compared to that of the GA and was found to have slightly higher average classification accuracy of 84.2% versus the GA's 83.5% accuracy. He also found that with only 20 features, ACO could achieve similar classification accuracy to the

results attained with a set of all 50 features. These results cannot be directly compared to our own because Al-Ani measured accuracy, while we measured F-measure of classification. We would need to know statistics about what percent of these data instances were positive instances to be able to draw more conclusions. We can identify ACO used in this study by its method of only using the k best subsets to update pheromone trails and start the construction of feature subsets of the next iteration. I incorporated this variation of Rank-Based Ant System [9] in my implementation of ACO for feature subset selection.

Another study which illustrated the potential of the ACO algorithm was done by Aghdam, Ghasem-Aghaee, Basiri [7]. They used the ACO algorithm for feature set selection as a new way to reduce the dimensionality of search space for a text categorization problem. They compared the performance of an ACO based approach to GA, IG, and CHI on the Reuters-21578 benchmark dataset. In the experiments, ACO out-performed all other methods. They calculated a Macro-F1 score equally weighting all classes. Their experiments resulted in Macro-F1 scores of 78.4 with ACO, 76.3 with GA, 70.9 with CHI, and 69.8 with IG. They also calculated Micro-F1 scores of 89.1 with ACO, 86.4 with GA, 82.2 with CHI, and 80.9 with IG, where the Micro-F1 average weights the performance of common classes highest. Thus ACO outperformed all other methods in both metrics.

Saraç and Özel [6] explored web page classification, which is another important aspect of the field of NLP, with the ACO. In earlier trials, feature extraction was only implemented as bag-of-words, producing relatively small

numbers of features. They broke from this convention and created a very highly dimensional feature space by looking at pairs of "tagged terms" (eg. <url><term>). This allowed them to try ACO on a much larger scale set of possible features. They ran an experiment using tagged terms on 5 different datasets. For every dataset, using ACO selected features increased the experimental resulting F-measure. Over all the datasets they had an average F-measure of .952 with ACO, and an average F-measure of .684 without feature selection. The paper also implemented ACO with an ant feature subset construction method that chooses all the features in groups instead of one by one to reduce unnecessary computation, and I have integrated this into my ACO implementation to improve efficiency.

2.5 Genetic Algorithm

The GA has been used to tackle the challenge of trying to capture domain specific concepts from text. This is highly relevant to our problem, as we are attempting to identify rationale in software documentation where there is a high occurrence of technical and domain specific terms. Ozyurt [29] introduced genetic algorithm based feature selection technique to improve the performance of classification of biomedical literature. He uses a supervised learning approach to label predicate argument structures in biomedical text, where domain specific corpora were necessary to develop strong classification models.

Mukherjee, et al. [30] use GA to select relevant attributes to classify emails. The attributes are sets of domain specific concepts present in the emails. Each chromosome solution is a vector of topics in the document, and parents for each

generation are selected with “truncation selection” – randomly from half of the best chromosomes.

2.6 Hybrid Algorithms

Many researchers have had success with combining the best features from multiple algorithms, known as hybrid algorithms. Zaiyadi and Baharudin [31] presented a new approach to feature subset selection by hybridizing ACO with information gain. They explore this novel method of reducing dimensionality of feature space for text document categorization. In this composite, information gain is used as the ant’s heuristic desirability measure. Similarly, Ali and Shahzad [32] proposed a filter based feature selection approach, combining ACO with Symmetric Uncertainty (variation of Information Gain). In this ACO-SU algorithm, the fitness evaluation is a function weighting the length of the chosen subset and the sum of the SU for all the features chosen in the subset. I have taken inspiration from these two in my implementation, using the similar information gain statistic of CHI squared as a measure of heuristic value.

Other researchers have tried combining ACO and GA algorithms. Basiri and Nemati [25] introduced an algorithm that hybridized the ACO with the GA for feature selection. This algorithm has the advantage of heuristic information being determined by classifier performance and length of selected feature subset, so that no a priori knowledge of features is needed. They hybridize the algorithm by applying selection, mutation, and crossover after a population is selected by ants, then exchanging the worse individuals with the better solutions found with the GA. They found this hybrid

algorithm does have an advantage over ACO alone. Also, Roeva, Fidanova, and Atanassova [33] presented a novel combination of ACO and GA, by using the ACO to generate an initial population for the GA. This allows the GA to start with a population nearer to an optimal solution. The result is that they reduce computational time by using much smaller populations than in the ACO or GA separately. Perhaps this could be a method to attempt in further research to see if it improves feature selection for classifying rationale.

Along these same lines, Jiang, et al. [34] employ a GATS algorithm which creates a hybrid of GA with the taboo search algorithm towards feature subset selection to improve text categorization. They focus on maintaining the advantages of each algorithm in the hybrid by including taboo search memory function into the GA's evolution based search routine. Since the ACO also has a concept of memory, an ACO and GA hybridization might benefit in the same way.

Chapter 3 Feature Subset Selection Algorithms: ACO & GA

This chapter discusses the ACO and GA metaheuristic approach to approximating solutions to combinatorial optimization problems. It also details our specific ACO and GA implementations used to search for near-optimal solutions to the feature subset selection problem for automatically capturing design rationale from software documentation.

3.1 Ant Colony Optimization Metaheuristic

The ACO metaheuristic is broadly applicable, and can be used to approximate any combinatorial optimization problem with an appropriate heuristic [9]. Implementing the ACO requires defining the following critical components [7], [25], [35], [36]:

1) Graph Representation of the Problem: A graph with nodes and edges between nodes representing the discrete search space of the problem [25]. It must be able to represent a solution to the problem.
2) Feasible Solution Construction Constraint: A mechanism to make sure only feasible solutions are built [25].
3) Heuristic Desirability: A "measure of goodness" [36] of adding any component to a partially constructed solution.
4) Pheromone Update Rule: A procedure for updating pheromone levels – namely a strategy for increasing pheromone concentration for previously successful

solutions, and an evaporation rule to globally decrease pheromone [36]. This constitutes the autocatalytic feedback process.

5) Probabilistic Transition Rule: A solution construction rule that dictates the probability of an ant next moving to a new node in the graph [25].

Here we illustrate the representation of the ACO metaheuristic for a general combinatorial optimization problem. A combinatorial optimization problem is mapped to a problem that is characterized by a "finite set $C = \{c_1, c_2, \ldots, c_n\}$ of components, where n is the number of components. ... Artificial ants build solutions by performing randomized walks on completely connected graph $G_c = (C, L)$ whose nodes are the components C, and the set L fully connects the components of C. The graph G_c is called *construction graph* and elements of L are called *connections*" [9]. The specific problem's constraints are implemented in the way in which artificial ants construct solutions, such that an ant is only allowed to add a component to the current solution if the resulting solution is feasible [9]. Each component $c_i \in C$ has an associated pheromone trail τ_i and a heuristic value η_i. Pheromones constitute a global, long-term memory of good solutions found throughout the search procedure, and are updated by each ant. Note that pheromone trails evaporate over time to avoid converging to sub-optimal local extrema. Moreover, the heuristic value represents theoretical information about the problem deduced from a source other than the ants (like the cost of adding a component to the solution).

Each simulated ant moves by applying a probabilistic transition rule, which is a function of the locally available pheromone trails, heuristic values, and problem

constraints. Once the artificial ants have built a solution, they can update the pheromones of the components along the path they took based on the quality of that solution. Dorigo explains the logic underlying his algorithm by saying, "It is important to note that ants act concurrently and independently and that although each ant is complex enough to find a (probably poor) solution to the problem under consideration, good-quality solutions can only emerge as the result of the collective interactions among the ants" [9]. The ACO is a unique metaheuristic in that it is a constructive, population-based metaheuristic, and it incorporates a persisting global memory of search performance [9].

There are many examples of the ACO algorithm's application to combinatorial optimization problems. The earliest experiments applying ACO attempted to solve ordering problems such as the traveling salesperson problem [37]. The problem is represented with a completely connected graph – where each city is a node in the graph and each arc between cities is an edge in the graph. Because it is completely connected, each city is connected to every other city. Artificial ants would then construct solutions guided by the heuristic measure of Euclidean distance between cities – with one solution being a permutation of the set of all cities. This means each ant visits all nodes in the graph, and the order in which the ant visited the nodes would be considered a tour. Additionally, ants would deposit pheromones on edges between cities to indicate which paths between cities had previous success.

The ACO can also be used to solve subset problems [14]. In our specific problem, we are looking for subsets of features instead of a tour of cities. Thus, nodes

in the graph represent features instead of cities, and a solution is not the order in which the ants visit all the cities but instead a set of the features the ant chooses. In this case, ants construct a solution by applying a probabilistic transition rule, adding available features to their current subset until a stopping criteria is met. For this class of problems, pheromone trails are associated with components rather than connections, since the ordering of items chosen is irrelevant. Interestingly, the TSP can be seen as an instance of a subset problem, where ants are choosing a subset of all available arcs between cities [9].

ACO has been applied to many different types of subset problems [38] including the Multiple Knapsack problem [14], Set Covering problem [39], and Maximum Clique problem [40]. ACO has been further extended to a wide range of machine learning problems, including learning the structure of a Bayesian Network [41] and learning rules in a Fuzzy System [42].

3.2 Ant Colony Optimization Implementation

Figure 3-1 shows the pseudocode of the ACO based feature subset selection algorithm run for my experiments, and is based on an algorithm outlined by Al-Ani in [10]. Figure 3-2 is a diagram of the implementation of this ACO feature subset selection algorithm.

Initialize:

I = # iterations
NA = # ants
M = feature subset size
T_i = initial pheromone level
ρ = evaporation rate
K = # best ants' subsets used to update pheromone
P = # remaining features to be selected each iteration
USM (Updated Selection Measurement) parameters
η = relative weight of pheromone trails
κ = relative weight of local importance
Subsets = For each ant: randomly assign $S_{ant} = \{f_1, f_2 ..., f_M\} \subset \{\text{all features}\}$

For iteration in range(I):

For each ant: FS_{ant} = Generate feature set from S_{ant} and training sentences
For each ant ***In Parallel***: calculate Fitness(FS_{ant})
Sort subsets by highest fitness
Update best subset found so far if necessary
For each of K best ants:
Update pheromone trails of paths they constructed
K-Set = {union of all features in K best ants' subsets}
For each ant:
randomly assign m-p feats from K-Set: $S_{ant} = \{f_1, f_2, ..., f_{m-p}\} \subseteq$ K-Set
For each ant:
Choose p best features that maximize USM(feature, S_{ant})
$S_{ant} = S_{ant} \cup \{f_{m-p+1}, \ldots, f_m\}$

Return Best Subset

Figure 3-1: Pseudocode of ACO

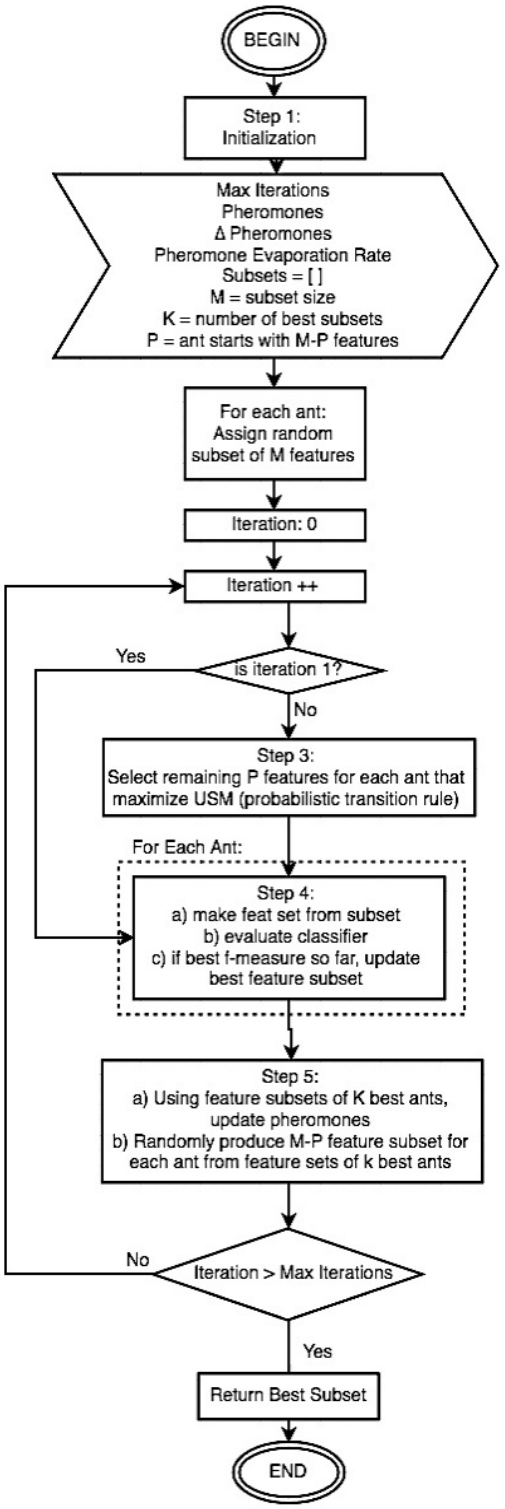

Figure 3-2: Diagram of ACO Algorithm

3.2.1 Graph Representation of the Problem

Nodes represent feature categories, and edges between them represent choice of the feature category (Figure 3-3). The search is represented by ants walking along the graph, until a stopping criteria is met. Nodes are fully connected to allow any node to be added to the ant's constructed subset.

3.2.2 Feasible Solution Construction Constraint

In this case, the stopping criteria is that a pre-specified number of nodes have been chosen, with each one chosen by satisfying the probabilistic transition rule. Figure 3-3 illustrates an ant starting at f_1. As all the features are connected, the ant could choose to move to any feature next. In this example, the pre-specified size of a feature subset is four. As the ant has not met the stopping criteria, since it has not constructed a subset of four features, it uses the probabilistic transition rule to move to feature f_2. Similarly, it then moves to f_3 and f_4. At this point it has met the stopping criteria, and returns this feature subset $\{f_1, f_2, f_3, f_4\} \subset \{\text{all features}\}$ as a possible optimal feature subset. Note, because a candidate solution in this problem is a subset, which does not define an order between its components, the probabilistic transition rule is not influenced by the partial solution constructed so far or the most recent component chosen by the ant [14].

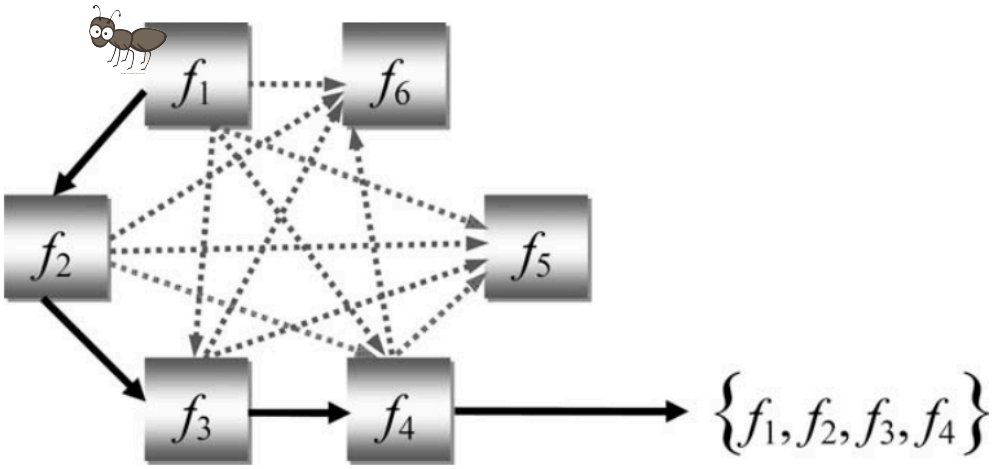

Figure 3-3: ACO Solution Construction [7]

3.2.3 Heuristic Desirability

As is the case with ACO applied to subset problems, heuristic value is associated with a feature instead of the edge between two features [14]. I have decided to define Local Importance (LI) as the heuristic measure which is calculated as the highest CHI-squared statistic of all terms in the category. This is a filter function in that it is static and calculated before ACO runs.

CHI-squared statistic: Note that the implementation used in our algorithm comes from the Python nltk library, where the source code states that the statistic is calculated from the "*Pearson's chi-square as in Manning and Schutze 5.3.3."* The statistic is defined from this source.

For two words, x and y, make a 2x2 table of frequencies of bigram occurrences in the corpus.

x + y	(not x) + y
x + (not y)	(not x) + (not y)

(e.g. for words x = “hello” and y = “world”, then x + y = “hello world”,

not x + y = “small world”, etc.)

“The X^2 statistic sums the differences between observed and expected values in all squares of the table, scaled by the magnitude of the expected values as follows:

$$X^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

where i ranges over rows of the table, j ranges over columns, O_{ij} is the observed value for cell (i,j) and E_{ij} is the expected value.” [43]

3.2.4 Pheromone Update Rule

As is the case with ACO applied to subset problems, the pheromone value is associated with a feature instead of the edge between two features. Pheromone update rule for an ant is shown, where pheromone of feature i is being updated from T_i -> T_i’, with ρ representing pheromone evaporation rate. My ACO implementation follows the standard method of selecting the k best ants and updating the paths they took [36], to reinforce the success of these best paths.

For each feature i in k best ants' features sets:

Equation 3-1, adapted from [10]

$$\Delta \mathrm{Ti} = \frac{\mathrm{fitness(ant)} - \mathrm{worst_{fitness}}}{\max_{j=1:k}(\mathrm{fitness(j)} - \mathrm{worst_{fitness}})} \quad (3.1)$$

Equation 3-2, [10]

$$\mathrm{Ti'} = (1-\rho) * \mathrm{Ti} + \Delta \mathrm{Ti} \quad (3.2)$$

Fitness Function: Utilizes a Naïve Bayes classifier [44] to evaluate the suitability of the ant's subset. It is a wrapper method as it incorporates a machine learning algorithm.

Input: FS_{ant} = feature set of all terms from categories in S_{ant} for all training sentences
Output: F-measure of classifying FS_{ant} with Naïve Bayes Classifier (Average of 10-fold cross validation)

3.2.5 Probabilistic Transition Rule

In my implementation, each ant chooses p features that maximize updated selection measurement (USM).

Equation 3-3: Probablistic Transition Rule, [10]

$$USM_i^{S_j} = \begin{cases} \dfrac{(\tau_i)^{\eta}\left(LI_i^{S_j}\right)^{\kappa}}{\sum_{g \notin S_j}(\tau_g)^{\eta}\left(LI_g^{S_j}\right)^{\kappa}} & \text{if } i \notin S_j \\ 0 & \text{Otherwise} \end{cases} \quad (3.3)$$

USM of feature i with respect to S_j (the subset of ant j):
T_i = pheromone level of feature i (see 3.2.4)
LI_i = Local importance of feature i (see 3.2.3)
η = relative weight of pheromone intensity
κ = relative weight of local importance

3.2.6 Runtime Analysis

The most time intensive step of each iteration in the ACO procedure was generating the feature set for each ant's subset and then calculating the fitness of these feature sets with 10 fold cross evaluation by Naïve Bayes classifier.

- I = Number of iterations
- A = Number of ants
- S = Number of sentences (# train or test differ by constant factor)
- C = Number of categories in each ant's subset
- F = Average number of features in a category (Avg. over all the sentences)
- W_{nltk} = Work to train the NLTK classifier with a feature set,
  or work to classify a feature set with the NLTK classifier
- # Folds = Number of folds for cross validation in evaluating the classifier.
  Included for clarity, but technically a constant factor.

The work of this process can be described by:

$$W = I \times A \times (W_{generate\_feature\_set} + W_{calculate\_fitness\_of\_feature\_set})$$
$$= I \times A \times S \times (C \times F + \#\,Folds \times W_{nltk})$$

Luckily, this step was also the most parallelizable. With no memory limitations and with number of processors equal to number of ants, both of these time intensive steps could be run completely in parallel for every ant.

In the ideal case the span of the process would be described by:

$$S = I \times (W_{generate\_feature\_set} + W_{calculate\_fitness\_of\_feature\_set})$$
$$= I \times S \times (C \times F + \#\,Folds \times W_{nltk})$$

Unfortunately, because of memory limitations, we could not run the feature subset creation step in parallel. However, if we assumed number of cores available is equal to number of ants, the span of the process is described by:

$$S = I \times (A \times W_{generate\_feature\_set} + W_{calculate\_fitness\_of\_feature\_set})$$
$$= I \times S \times (A \times C \times F + \#\,Folds \times W_{nltk})$$

On the Wesleyan Cluster, we only had 40 cores available, but used 100 ants, so the process was not perfectly parallelized. However, it was still a constant factor

difference between number of cores and ants so the span equation is still descriptive. Additionally, due to memory constraints, there was an additional step of reading each feature set in and out from a file. Although there likely was some overhead using the Python pickle library for this step, for this analysis I assumed that that process was order of S because each feature set file had one entry for each sentence. With this calculation, the term for reading file in and out dropped because it was lower order.

3.3 Genetic Algorithm Metaheuristic

Here we illustrate the representation of the GA metaheuristic for a combinatorial optimization problem. A combinatorial optimization problem is mapped to a problem that is characterized by a finite set $C = \{c_1, c_2, \ldots, c_n\}$ of components, where n is the number of components. A population of P chromosomes is defined where each chromosome is characterized by of a subset of $m < n$ randomly chosen components. This population will change over each iteration, but will always have a fixed number of P chromosomes. Genetic operators of crossover, selection, and mutation are defined to guide the reproduction of the population at each generation, building new candidate solutions within the problem constraints. A fitness function is also defined. It is the objective function that evaluates the quality of the solution. In a successful run of a genetic algorithm, the population of chromosomes converges so that each individual has a very similar genotype, and the chosen chromosome is a near optimal solution to the central optimization problem [45]. GAs are superior to other traditional search methods in that they explore more solutions to the optimization problem in parallel, and therefore are less likely to become trapped

in local suboptimal areas of the search space [20]. Another advantage of the GA is that it does not require any domain knowledge about the problem to optimize the classification procedure.

3.4 Genetic Algorithm Implementation

Figure 3-4 shows the implementation of this GA feature subset selection algorithm run for my experiments. Figure 3-5 diagrams this GA based feature subset selection algorithm.

```
Initialize:
        I = # iterations
        P = # chromosomes
        L = feature subset size
        Retain = % best chromosomes to be retained as parents for next generation
        Mutate = probability of parent chromosome to be mutated
Generate initial population:
        Individuals = randomly select L features for each chromosome c
        Population = [ (fitness(c),  c) for each chromosome c in individuals ]
Update best chromosome/fitness so far

For iteration in range(I):
        Evolve population:
        ⇒ Select best Retain% of population as Parents
        ⇒ For each chromosome in Parents:
                If random # < Mutate:
                    mutate chromosome and update fitness
        ⇒ While # of parents + children generated so far < P:
                generate child by crossover
        ⇒ For each child:
                generate feature set from child's features and training sentences
        ⇒ For each child *In Parallel*: calculate fitness from child's feature set
        ⇒ Population = Parents + Children (list concatenation)
        Update best chromosome/fitness so far

Return Best Chromosome
```

Figure 3-4: Pseudocode of GA

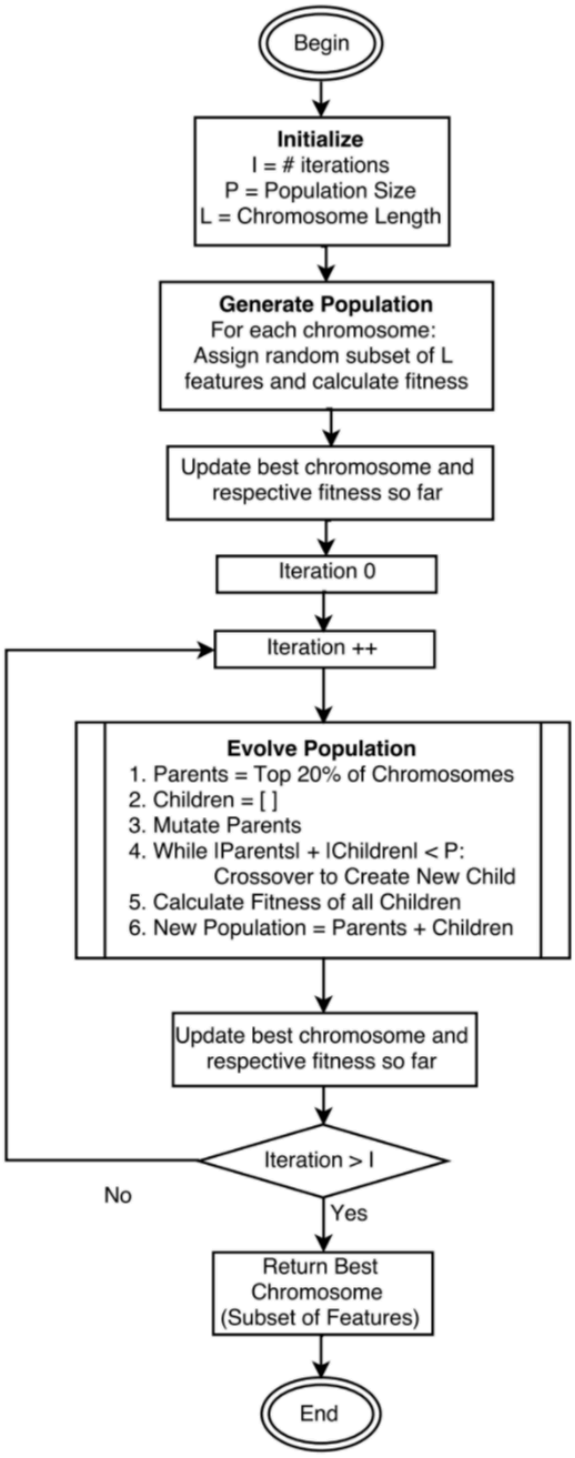

Figure 3-5: Diagram of GA Algorithm

3.4.1 Representation of Chromosomes

Each chromosome is a candidate solution to the optimization problem. Each gene in the chromosome represents one feature category. In the implementation, a chromosome is a fixed length list of feature categories representing a candidate feature subset (Figure 3-6, Example). If a specific feature category is a gene in a chromosome, then all the instances of features in that category will be considered relevant and included in the resultant classifier [17]. In this representation, the order of the genes does not matter because the chromosome is representing an unordered subset of feature categories. Further, the order has no impact when the chromosome is evaluated. The features of each chromosome are initially randomly selected, but then are the result of the selection, crossover, and mutation operators as the population evolve.

In the example chromosome in Figure 3-6, the features chosen for that subset are listed. Some features include n-grams (denoted by (n) prefix) and instances from adjacent sentences (denoted by {m} prefix). An n-gram is a sequence of n feature instances concatenated together to form a new feature instance. For example if "design", "rationale", and "extraction" were separate feature instances of some category, c1, in sentence, s1, then "designrationale" and "rationaleextraction" would be 2-grams, and "designrationaleextraction" would be a 3-gram. I used m-adjacent to refer to combining feature instances from adjacent sentences. If there is a sentence adjacent to s1 that has feature instances from c1 that are "machine" and "learning,"

the 2-adjacent sentences will include all feature instances “design,” “rationale,” “extraction,” “machine,” and “learning.” See Appendices A and B for all possible features and a table of feature abbreviations.

Chromosome Representation:

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{10}	F_{20}

=> {F1, F2, F3, … , F20} $\subseteq$ {All Features}

Example Chromosome:
[{3}(5)prp$, {2}vb, {2}(4)rb, (3)nnp, (2)ls, {2}(5)pdt, uh, (2)unigrams, (3)rbr, (2)roots, roots, (5)general, {3}(3)vbz, {2}sym, (3)unigrams, {3}cc, wp, {3}nn, {3}wp, {2}cc]

Figure 3-6: Chromosome Representation

3.4.2 Length of Chromosomes

Chromosome length is equal to the number of features in a candidate solution [17]. I have chosen to fix the length of each chromosome to 20 feature categories, so that a substantial number of feature categories are eliminated.

3.4.3 Size of Population

The size of the population is a pre-determined, fixed number of chromosomes that stays constant over every iteration. For our experiments, population size is set to 100 which is a normal value for this component [17].

3.4.4 Fitness Function

This fitness function is a wrapper method in that it utilizes the classifier to evaluate

the suitability of the chromosome [17]. Similar to ACO, we use the Naïve Bayes classifier [44].

Evaluate fitness of chromosome, c:

Input: FS_c = feature set of all terms from categories in c for all train sentences

Output: F-measure of classifying FS_c w/ Naïve Bayes (Avg. 10-fold cross validation)

3.4.5 Selection

We pick a small subset of parent chromosomes with the highest fitness (average Naïve Bayes F-measure over 10 fold cross validation) to "reproduce" in every next generation [46]. This is an example of *truncated selection* [30], where only the top chromosomes can be selected for reproduction.

3.4.6 Crossover

When we apply the genetic operator of crossover (Figure 3-7), we select two different parents randomly from the pool of chromosomes selected to repopulate the next generation. Then we randomly select half of the categories from parent A and randomly select half of the categories from parent B, and combine these categories together into a new set representing the "child" chromosome with any duplicates removed. Finally, if the length of the child is shorter than the chromosome length (because the same features were inherited from both parent A and B and the duplicate was removed) randomly add in features from the set of all features, until the pre-determined chromosome length has been reached. This constitutes a random

exchange of information to reproduce a chromosome with features from both parents. This method was chosen because the ordering of the features in the chromosome is not relevant [9]. Additionally, if the parents were too similar, that is an indication that the solution is converging and stagnating, so this method introduced more variation to continue searching the feature space.

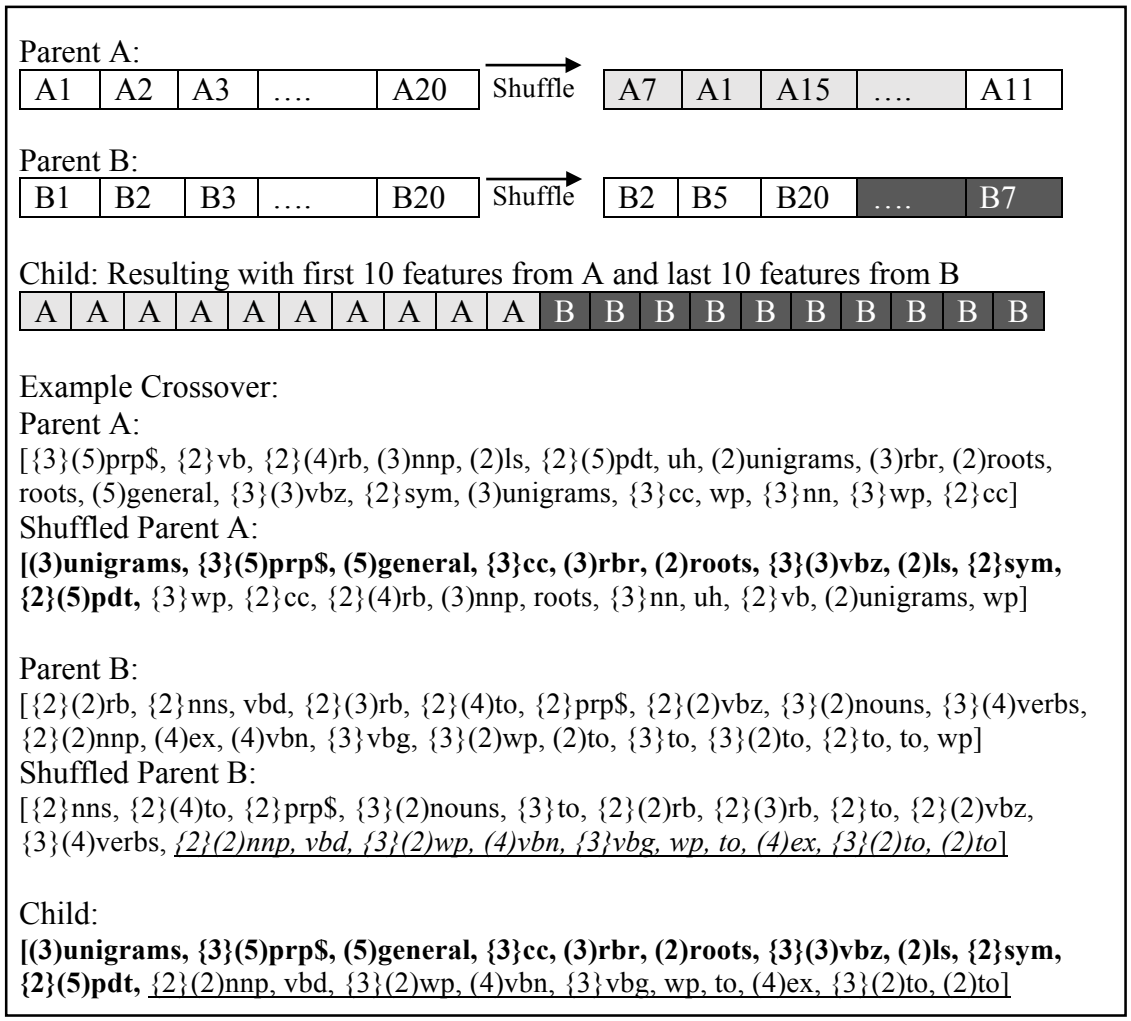

Figure 3-7: Crossover Operator

3.4.7 Mutation

If a parent chromosome is selected to be mutated (Figure 3-8), we change the feature at one random position to a randomly chosen feature [46]. The feature is randomly chosen from all features that are not already a gene in this parent chromosome. This simulates genetic variation and increases the area of the feature space searched.

Chromosome:

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{10}	F_{20}

^Mutate

Mutated Chromosome:

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F'	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{10}	F_{20}

F10 is replaced with F' $\in$ {All Features}\{F1, …, F20}
F' is chosen randomly from the available features.

Example Mutation:
Chromosome:
[{3}(5)prp$, {2}vb, {2}(4)rb, (3)nnp, (2)ls, {2}(5)pdt, uh, (2)unigrams, (3)rbr, **(2)roots,** roots, (5)general, {3}(3)vbz, {2}sym, (3)unigrams, {3}cc, wp, {3}nn, {3}wp, {2}cc]

Mutated Chromosome:
[{3}(5)prp$, {2}vb, {2}(4)rb, (3)nnp, (2)ls, {2}(5)pdt, uh, (2)unigrams, (3)rbr, **{3}to,** roots, (5)general, {3}(3)vbz, {2}sym, (3)unigrams, {3}cc, wp, {3}nn, {3}wp, {2}cc]

Figure 3-8: Mutation Operator

3.4.8 Runtime Analysis

Asymptotic runtime analysis of the GA is nearly identical to that of ACO (see section 3.2.6), with a few terminology substitutions. Iterations is equivalent to generations, the number of ants is equivalent to the population size, and the number of categories in each ant's subset is equivalent to chromosome length. Experiments were run on same computing cluster, so memory and processors limitations faced were identical. The main logic of the most intensive steps of the algorithms were the same. In the

GA, analogous to the ACO, for each generation we build the feature set for each chromosome and then calculate the fitness of these feature sets with 10 fold cross evaluation by Naïve Bayes classifier.

- G = Number of generations
- P = Population size
- S = Number of sentences (# train or test differ by constant factor)
- C = Chromosome length
- F = Average number of features in a category (Avg. over all the sentences)
- W_{nltk} = Work to train the NLTK classifier with a feature set,
  or work to classify a feature set with the NLTK classifier
- # Folds = Number of folds for cross validation in evaluating the classifier.
  Included for clarity, but technically a constant factor.

The work of this process can be described by:

$$W = G \times P \times (W_{generate\_feature\_set} + W_{calculate\_fitness\_of\_feature\_set})$$
$$= G \times P \times S \times (C \times F + \#\,Folds \times W_{nltk})$$

In the ideal case, with no memory limitations and number of processors equal to population size, the span of the process would be described by:

$$S = G \times (W_{generate\_feature\_set} + W_{calculate\_fitness\_of\_feature\_set})$$
$$= G \times S \times (C \times F + \#\,Folds \times W_{nltk})$$

Unfortunately, because of memory limitations, we could not run the feature subset creation step in parallel. In this case, the span of the process is described by:

$$S = G \times (P \times W_{generate\_feature\_set} + W_{calculate\_fitness\_of\_feature\_set})$$
$$= G \times S \times (P \times C \times F + \#\,Folds \times W_{nltk})$$

In real time, the GA experiments took longer than the ACO experiments to on average. This means it is likely that the GA implementation had more overhead from lower order operations of selection, crossover, and mutation than the ACO had for mechanics of constructing ant subsets. However, the times experiments took to run on the cluster were not consistent, so it is also possible that there were other factors pertaining to the cluster platform that could have influenced this observation.

Chapter 4 System Design

The system (Figure 4-1) is a pipeline that takes in the preprocessed sentences.csv file database as input, and outputs a classification model. Since it wraps around the feature subset selection algorithm, the same pipeline is used for both ACO (Figure 3-1) and GA (Figure 3-4) based subset selection algorithms. The system was built in Python, and utilizes the NLTK library for natural language processing and the multiprocessing library for parallel computing. It is designed to run on the Wesleyan supercomputing cluster to take advantage of multiple processors for efficient parallel computing in the feature subset selection procedure. Rogers implemented info gain filtering in his GATE-WEKA pipeline [5]. However, the Wesleyan cluster can handle computation with all the feature instances, and thus this filtering step is excluded.

4.1 The Sentences Database

The input to the pipeline system is the sentences.csv database. This database holds both the features extracted from each sentence and the rationale annotations for each sentence in one dataset. The database is a table, where each row is a sentence of the corpus. Each row consists of the name of the XML file that the sentence originated from, the sentence ID, and all of the extracted feature instances for each category for that sentence [5]. Features used come from earlier research. Parts of speech features are from the Penn-Treebank [47] and other features are from research by Burge [48].

4.1.1 Sentences.csv Header

The first line of the sentences.csv file stores information about which feature categories are extracted from each sentence and also which rationale subclasses are annotated for each sentence. A convention is used to identify features that are n-grams, and also ones that contain instances from m-adjacent sentences. If (n) precedes the feature category name, then that feature category contains n-grams of feature instances. If {m} proceeds the feature name, then the feature includes instances of that category from the m-adjacent sentences (see 4.3.1 for example of n-grams and m-adjacent). Our sentences.csv database is constructed to include up to 5-grams for each feature, and features from up to 3-adjacent sentences. We also include valid rationale annotations of questions, answers, alternatives, decisions, assumptions, procedures, arguments, and requirements.

4.1.2 Row of Sentences.csv

As denoted in the header, each row begins with an XML ID and a Sentence ID. Then for each category in the header, every feature instance in that sentence is recorded. Finally for each rationale subclass, there is a Boolean value signifying whether the annotation is present in the sentence.

4.1.3 Example Sentences.csv Excerpt

This example will illustrate the contents of a sentences.csv file created from the SPSD dataset.

Here is a partial sentences.csv header, where the first four feature categories are adjectives, and 8 rationale subclasses are annotated for each sentence:

xmlID`sentID`` adjectives ` {2}adjectives ` {3}adjectives ` (2)adjectives … ``alternative ` argument ` assumption ` decision ` requirement ` question ` answer ` procedure`` [See appendix A for complete list of feature categories in sentences.csv]

Consider a sentence from the SPSD dataset, with adjectives underlined:

S1= "Male 2: Should we define an initial simple intersection, an initial simple car"

The sentences before and after it are:

S0 = "Well, where is the first place to start"

S2 = "Male 1: You mean the actual data structures".

The sentences 2-adjacent have no adjective instances.

Here is a corresponding excerpt from the row for S1 in sentences.csv (with duplicate feature instances removed for clarity):

spsd_adobe_filtere.xml` 29384`` initial simple` first initial simple actual` first initial simple actual` initialsimple simpleinitial` ….`` false` false` false` false` true` false` false` false`

4.1.4 Sentences File Generation

A sentence file for each dataset was originally generated by Rogers [5]. The features of each sentence were extracted using the annotation tools in the General Architecture for Text Engineering (GATE) software. In GATE, sentences were also manually annotated with rationale by multiple students. After manual and automatic annotation in GATE, the files could be exported as XML. This information from the XML files was used to generate the sentences.csv. Similar sentences.csv files were used by Rogers, et al. [19]. I have refactored the code that extracts features from the XML into the database, and as a result was able to capture many more feature categories than in the experiments by Rogers, et al [19].

4.2 Parsing Sentences

Deciding what type of data structure to use to represent each sentence and corresponding feature set was one of the key design decisions made. All the feature instances listed in the sentences.csv file had to be encapsulated in the sentence data structure. We also had to link each feature instance to the category it came from to build the related feature set. The ability to efficiently construct a feature set from any subset of feature categories was also a requirement. Thus the feature instances of the sentence are organized in a dictionary structure (Figure 4-1A), with each (key, value) pair holding information about only an individual category. The key is the ID of the feature category, and the value is the set of all feature instances of that feature category for that sentence. With a dictionary structure, each category can be accessed

in constant time, allowing the feature set (Figure 4-1B) to be created from a simple union of instances from each category in the chosen subset of categories.

Example, for the sentence: “I decided to use HTML”
0 1 2 3 4
With feature categories = [unigrams, bigrams, verbs, adverbs, acronyms]

[A] Sentence dictionary representation:

```
{
    0: {“i”, “decided”, “to”, “use”, “html”},                    # unigrams
    1: {“idecided”, “decidedto”, “touse”, “usehtml”},            # bigrams
    2: {“decided”, “use”},                                       # verbs
    3: {}                                                        # adverbs
    4: {“html”}                                                  # acronyms
}
```

[B] Feature set for the selected category subset = [1,2] = [bigrams, verbs]

```
{
    “idecided”: True,          # bigram
    “decidedto”: True,         # bigram
     “touse”: True,            # bigram
     “usehtml”: True,          # bigram
     “decided”: True,          # verb
     “use”: True               # verb
}
```

Figure 4-1: Example Sentence Parsing.
[A] shows sentence dictionary representation. [B] shows feature set representation.

4.3 Pipeline Design

The following sub-sections describe what each of the components of Figure 4-2 represents.

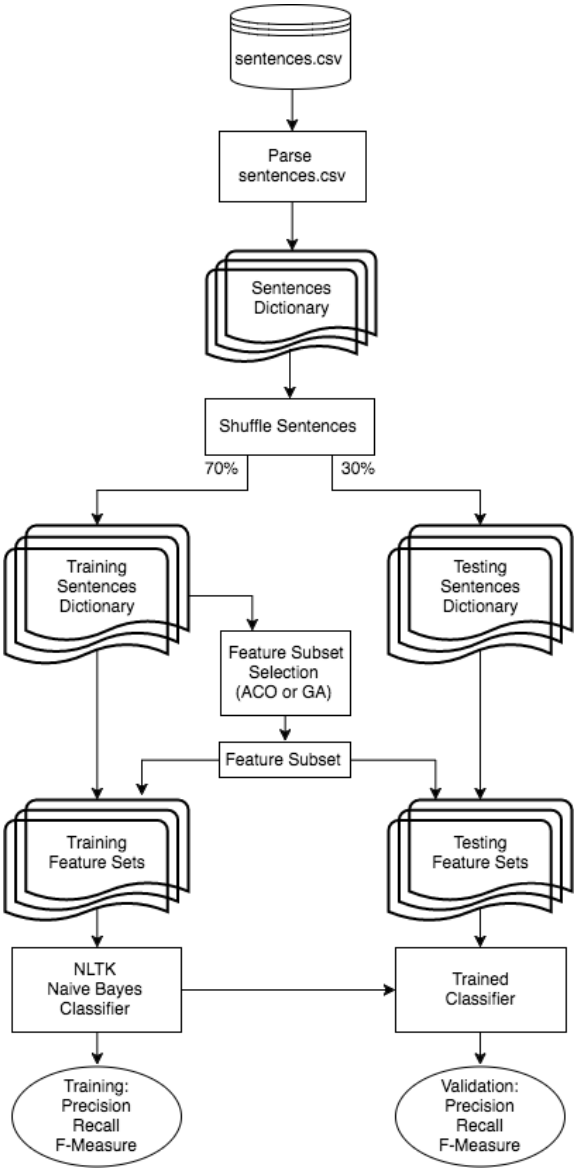

Figure 4-2: The Pipeline Design

4.3.1 Parse Sentences.csv

The rectangle marked “parse sentences.csv” represents where the sentences.csv database file is read into the program. Two arguments are supplied to help parse out the data from the database. The first argument denotes which subclasses of rationale are valid (i.e. given positive classification) and the second argument denotes what source dataset to use. The first line of the sentences.csv file is parsed into a list of feature categories, where the ID of a feature category is its index in the list. Each sentence is read into memory. For each sentence, all punctuation is removed. The sentence is then parsed into a dictionary where the key corresponds to the feature ID and the value is a set of all feature instances given for that category in that sentence (Section 4.3, Figure 4-1A). Finally, if the sentence is annotated with any of the valid rationale subclasses, it is given the classification label of “rationale;” otherwise, it is given the label of “nonrationale.”

4.3.2 Shuffle Sentences

The box marked “shuffle sentences” represents the step where the indices of the sentences are randomly shuffled.

4.3.3 Training Sentences Dictionaries and Testing Sentences Dictionaries

The shuffled indices are split 70/30 to distinguish the training sentences and the testing sentences respectively – keeping the same ratio of positive and negative instances in training and test sets [49]. Each parsed sentence dictionary is written to

the training or testing file corresponding to its index with the Python pickle library – which allows reading in/out arbitrary Python data structures. The list of indices of testing sentences are written to a file with the Python pickle library as well, so that the experiments can be re-run with the same training/test data split.

4.3.4 Feature Subset Selection and Feature Subset

Only the training sentences are input into the feature subset selection algorithm. These sentences are used to influence which features are chosen by the algorithm. The output to the feature subset selection step is the feature subset chosen at the iteration 40 of the feature selection routine. The feature subset is a list of n feature categories, $[C_1, \ldots, C_n]$, where n is the predetermined length of the feature subset – equal to 20 for my experiments.

4.3.5 Training Feature sets and Testing Feature Sets

Each training feature set is generated from a training sentence dictionary and the selected feature category subset. For each training sentence, the feature set is a new dictionary representing the union of all present feature instances in all chosen feature categories (Section 4.3, Figure 4-1B). We generate a feature set for each sentence in test sentences by the same method as for the training sentences.

4.3.6 NLTK Naïve Bayes Classifier: Build Classifier

Each sentence is represented as a tuple of its feature set and its binary rationale classification label. An NLTK Naïve Bayes classifier is then trained with the list of all (feature set, label) tuples of the training sentences. This classifier is the model of our data that can be used to generate labels for new sentences.

4.3.7 Trained Classifier: Evaluate Classifier

We classify the test feature sets with the trained Naïve Bayes classifier, and compare the predicted test class labels to actual test class labels.

4.3.8 Precision, Recall, F-Measure

By calculating how many labels the classifier assigns correctly, we can analyze the validity of the model we have generated. More specifically, we perform statistical analysis of how well the sentences were classified to determine precision, recall, and F-measure. Training statistics are calculated over 10-fold cross validation of the training sentences. Validation statistics are calculated from comparing labels of classified test sentences to the actual class labels.

Chapter 5 Experimental Design

Experiments were designed such that at runtime I could specify which dataset, valid subclass of rationale, and feature selection algorithm to use. I looked at every combination of dataset, rationale subclass, and feature selection algorithm. With 2 datasets, 2 feature selection algorithms, and 11 subclasses of rationale, there were a total of 2*2*11 = 44 different experiments. Because the experiments took less time, I was able to run multiple trials of each experiment. This allowed for the calculation of standard deviation between results of each experiment to learn about the consistency of the feature selection algorithms. For each dataset and valid rationale, I generated a single split of training/test sentences so that experiments with different feature subset selection algorithms would be consistent and comparable. In each split of testing/training sentences, we preserved the balance of positive and negatively classified sentences in training and test set [49] and [4]. The test sentences were set aside as a holdout set, such that the feature selection algorithm was completely blind to these data. Thus, they had no influence on the feature subset chosen by the algorithm. This allowed the predictive ability of the classifier trained with a candidate subset of features to be evaluated by classifying these test sentences [50]. The trial resulting in the feature set that had the highest predictive ability for classifying the identical held out test sentences was considered the best model.

5.1 Valid Rationale

In multi-class valid rationale experiments, we considered a group of rationale subclasses to be valid rationale. The sentences annotated with at least one of these chosen rationale classes are considered positive instances (rationale), and the rest of the sentences that are not annotated with any of these categories of rationale are considered negative instances (non-rationale). These include binary rationale, argumentation subset, and arguments-all. In individual class valid rationale experiments, we considered single subclasses of rationale to be valid rationale. If the sentence was annotated with the specified rationale, then it was considered a positive instance (rationale), otherwise it was considered a negative instance (non-rationale). These include alternative, answer, argument, assumption, decision, question, and requirement.

5.2 Feature Subset Selection Parameters

First, the feature subset selection algorithm parameters had to be set. In order for the GA to be easily compared with previous research, I used parameters as consistent with Rogers' parameters [5] as possible. I used the same population size Rogers [5]. Population size was set to 100. In Rogers experiments, the chromosome was a bit string with length equal to 512, where each mutation flipped a bit – either turning a feature on or off. The mutation rate was set to .01, so each chromosome would expect an average of 5 mutations [5]. In my experiments, each chromosome was represented by a list of 20 present features, and one mutation would be equivalent to two mutations in Rogers representation (turning one feature off and a

different feature on). Thus I set mutation rate to .1, so there would be an average of 2 mutations per chromosome, equivalent to 4 mutations in Rogers representation. Because the experiments were taking much less time than when Rogers ran them, I doubled the number of generations. For my experiments the number of generations was set to 40 to give the algorithm more time to converge towards an optimal solution. I also kept 20% of chromosomes as parents for the next generation. I pre-specified the length of each chromosome to be 20 features. My fitness function was an evaluation of F-measure calculated by 10-fold cross validation with a Naïve Bayes classifier from the Python NLTK library.

Certain ACO parameters correspond with GA parameters. To maintain consistency, the values of these parameters were matched. The number of ants, analogous to population size, was set to 100. 40 iterations were used, because of the equivalent 40 generations set for the GA. Further, the length of the feature subset constructed by each ant was set to 20 to be comparable to the chromosome length. The top k=20 ants were selected to update pheromone concentrations on the paths they constructed, parallel to the top 20% of chromosomes selected as parents. The rest of the ACO parameters were discovered experimentally. Pheromone evaporation rate was set to .1. For heuristic measure of a feature category, I used the maximum CHI-squared statistic of the features in that category. Heuristic measure was given more weight to pheromone trail intensity – with alpha = 1 and beta = 2. This was found experimentally to decrease the speed of convergence, so ants did not get stuck in poor local optima. To construct each ant's feature subset, 3/4 of features were

chosen from k best ants of the previous generation, and 1/4 of features were chosen to maximize the update selection measurement rule (balancing pheromone trails and heuristic value).

5.3 Classifier

Classifiers from the Python NLTK and Sklearn machine learning libraries were compared, in order to choose which classifier was best. The NLTK Naïve Bayes, NLTK Decision Tree, Sklearn Bernoulli Naïve Bayes, Sklearn SVM (both SVC and LinearSVC), and Sklearn MaxEntropy (both GIS and IIS) classifiers were tested. The NLTK Naïve Bayes Classifier best balanced performance and efficiency, and thus NLTK Naïve Bayes was used throughout these experiments. The Sklearn MaxEntropy with GIS performed second best.

5.4 Testing for Validity

A 70/30 split of training and test data was used for all experiments. For each dataset and each subclass of rationale, I shuffled the sentences and then used the same split of training and test data for ACO and GA. This enabled the experiments to all be consistent with one another to allow for straightforward comparison of all results. This meant that the classifier would be built for both GA and ACO experiments from the same set of training sentences, and would be validated by classifying the same set of new test instances. After the near-optimal feature subset was selected by the feature selection algorithm, the model was evaluated. 10-fold cross validation results of classifying the training sentences with the NLTK Naïve Bayes classifier were

evaluated first. Then the NLTK Naïve Bayes classifier that was trained with all the training sentences was used to classify the unseen test data. Precision, recall, and F-measure of training with cross validation classification and test data classification were calculated. If the training F-measure is much higher than the testing F-measure, that is a sign that overfitting is occurring – i.e. that the model is fitted too tightly to the specific idiosyncrasies of the training data to be able to generalize on unseen data.

5.5 Parallelization

I found that the most computationally intensive step in the feature subset selection procedure was evaluating the fitness of every member in the population at every iteration (the wrapper function). To speed up this step, I implemented parallel processing with the Python multiprocessing library. Taking advantage of the many cores of the Wesleyan supercomputing cluster, I used the Python multiprocessing library to create a pool of processes so that each subset's fitness could be evaluated with its own process in parallel.

Because of Python's implementation, the best way to parallelize the algorithm is with multi-processing and not multi-threading. Thus, there is not shared memory. When I first attempted to run the parallel processes, the entire sentences data was being copied into the memory of each parallel process. Because the sentences data is very large, memory constraints were an issue. Thus, I kept the sentences data written to a file instead of loaded into global memory, and only copied the specific feature set to be evaluated into the memory of the spawned process.

5.5.1 Parallel Algorithm

For either ants or chromosomes, the fitness of each subset is evaluated in parallel. We evaluate the subset using the Naïve Bayes classification algorithm. Note that in diagram 5-1, subsets[i] stands for the subset constructed by ant/chromosome i, FS_i stands for Feature Set for ant/chromosome i, and FM stands for F-measure.

```
# function to calculate fitness for each subset
def processSubset(i):
        read FSi into memory
        return 10-fold cross validated FM of FSi with nltk Naïve Bayes classifier

# sequential, b/c parallel processes can't all access sentences simultaneously
for each i, subset in enumerate(subsets):
        FSi = for each training sentence:
                union of all feature instances of all categories in subset
        write FSi to file

create a pool of processes
call processSubset(i) for each subset in parallel, and save results as they finish
processes are all finished, close pool
make the results available, join pool
delete the pools so memory from each process is freed
all subsets' fitness are parsed from the list of results and are enumerated
        to indicate which fitness corresponds to which subset
```

Figure 5-1: Parallel Python Pseudocode for ACO or GA

Chapter 6 Results

Experiments were run on the Wesleyan High Performance Compute Cluster, on compute nodes with dual ten core E5-2550 v3 (2.3 Ghz) in Supermicro 1U rack servers with a memory footprint of 32 GB each. By taking advantage of the 40 cores available for computation on the cluster, we were able to significantly speed up the time to run experiments. For reference, in Roger's 2013 thesis it was estimated that running GAFFS with Chrome Bug Reports (BR) dataset would take 28 days [5]. In 2017, an improvement was seen as GAFFS experiments took only a week to run on BR, by utilizing some parallelization [19]. For this thesis, experiments with GA feature selection on BR dataset took on average 2.2 hours. Experiments with ACO feature selection on BR dataset took only 1.3 hours on average. Experiments on the smaller SPSD dataset took .4 hours on average with GA or ACO. For each valid rationale category, 5 trials each of longer GA and ACO experiments for BR were run and 30 trials each for shorter GA and ACO experiments for SPSD were run. The results reported are the best of the trials for each experiment, based on predictive ability assessed by classifying hold-out test data [50]. The standard deviation of results of trials are also given.

Experiments using *all* feature categories were run as a baseline to compare to experiments with ACO or GA feature selection. It took an average of 13s to classify rationale in SPSD with all features for each valid rationale category. It took 104s to evaluate BR with all features for each category on average. Inter-annotator agreement was calculated by comparing annotations of two researchers, and represents an

estimate of F-measure achievable manually. Any overlapping annotations were considered a match [5].

Each experiment outputs a text file with the best feature subset at each iteration. Statistics are also printed out about training and test F-measure of each best subset per iteration. All training scores were the result of 10 fold cross validation over just the training sentences with the Naïve Bayes classifier, using the subset of features generated at the 40th iteration of the feature selection algorithm. All validation scores were based on training a Naïve Bayes classifier with the training sentences, and evaluating this classifier over all the test sentences – still using just the subset of features chosen at the 40th iteration of the feature selection algorithm.

6.1 Dataset 1: Google Chrome Bug Reports

Binary rationale, alternatives, and decisions had higher validation scores from the model generated with ACO than with GA. The rest had higher validation scores from the model generated with GA than with ACO. Every rationale subclass had higher validation scores from the model generated with feature selection than with using all features. The max difference between training and validation F-measure (FM) in any BR experiment was .036, and the average difference between training and validation FM in any BR experiment was .006. This shows that there was almost no overfitting in the models generated. Further, the max difference between training and validation FM in any ACO experiment was .034 and the average was -0.001 (meaning on average the validation FM were slightly higher than training FM), while for the GA the max was .036 and the average was .009. Therefore there was less

overfitting with ACO models than with GA models. In the tables below, the highest validation score is bolded, to indicate the best model created. (See appendix C for graphs of each of the following experiment's results over 40 iterations.)

Binary Rationale (all rationale subclasses).
Percentage of sentences with this classification: 17.4%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.708	.943	.809	.001
	Validation	.688	.953	**.799**	.003
GA	Training	.725	.932	.815	.0003
	Validation	.696	.936	.798	.003
All Features		.593	.984	.740	
Inter-Annotator Agreement				.752	

Argumentation Subset (alternative, argument, assumption, decision, requirement)
Percentage of sentences with this classification: 9.5%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.478	.849	.610	.002
	Validation	.507	.843	.633	.007
GA	Training	.490	.831	.616	.0004
	Validation	.520	.827	**.638**	.006
All Features		.339	.986	.504	
Inter-Annotator Agreement				.524	

Arguments-All (requirement, argument, assumption)
Percentage of sentences with this classification: 5%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.348	.730	.470	.002
	Validation	.334	.672	.446	.016
GA	Training	.380	.696	.490	.001
	Validation	.357	.623	**.454**	.002
All Features		.186	.988	.314	
Inter-Annotator Agreement				.261	

Alternative

Percentage of sentences with this classification: 2.7%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.224	.559	.318	.001
	Validation	.246	.563	**.342**	.023
GA	Training	.231	.546	.323	.005
	Validation	.238	.505	.324	.012
All Features		.078	.984	.145	
Inter-Annotator Agreement				.332	

Answer

Percentage of sentences with this classification: 5.6%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.296	.712	.417	.001
	Validation	.320	.743	.448	.003
GA	Training	.308	.711	.428	.0005
	Validation	.331	.738	**.457**	.005
All Features		.190	.992	.319	

Argument

Percentage of sentences with this classification: 4.1%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.301	.697	.420	.002
	Validation	.284	.654	.396	.007
GA	Training	.326	.689	.440	.001
	Validation	.312	.617	**.414**	.003
All Features		.143	.977	.250	

Assumption

Percentage of sentences with this classification: .6%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.069	.297	.108	.012
	Validation	.063	.270	.102	.030
GA	Training	.105	.305	.153	.004
	Validation	.086	.216	**.123**	.018
All Features		.011	1.000	.021	

Decision

Percentage of sentences with this classification: 2.5%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.331	.496	.394	.009
	Validation	.335	.484	**.396**	.021
GA	Training	.323	.559	.408	.004
	Validation	.301	.509	.379	.009
All Features		.077	.994	.142	
Inter-Annotator Agreement				.554	

Procedure

Percentage of sentences with this classification: 3%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.200	.563	.293	.001
	Validation	.202	.562	.297	.009
GA	Training	.222	.547	.314	.012
	Validation	.215	.515	**.303**	.016
All Features		.090	.985	.165	

Question

Percentage of sentences with this classification: 3%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.906	.513	.652	.002
	Validation	.886	.474	.618	.007
GA	Training	.983	.516	.673	.001
	Validation	.969	.474	**.637**	.003
All Features		.097	.990	.176	

Requirement

Percentage of sentences with this classification: .5%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.110	.580	.175	.013
	Validation	.126	.561	.206	.043
GA	Training	.203	.271	.223	.009
	Validation	.250	.268	**.259**	.015
All Features		.009	1.000	.018	

6.2 Dataset 2: Studying Professional Software Design

Alternative, assumption, decision, procedure, question, and requirement had higher validation FM from the model generated with ACO than with GA. The rest had higher validation FM from the model generated with GA than with ACO. Every experiment had higher validation FM from the model generated with feature selection than without. The max difference between training and validation FM in any SPSD experiment was .12, and the average difference between training and validation FM in any SPSD experiment was .007. This shows there was minimal overfitting overall. Further, the max difference between training and validation FM in any ACO experiment was .072 and the average was -.006, while for GA the max was .12 and the average was .02. The GA models over-fit more than ACO models, but this was mostly due to the outlier GA Decision model with the highest overfitting. In the tables below, the highest validation score is bolded, to indicate the most representative model created. (See appendix D for graphs of each of the following experiment's results over 40 iterations.)

Binary Rationale (all rationale subclasses)
Percentage of sentences with this classification: 53.5%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.627	.934	.750	.004
	Validation	.626	.926	.7467	.007
GA	Training	.648	.946	.768	.002
	Validation	.625	.929	**.7474**	.007
All Features		.664	.754	.706	
Inter-Annotator Agreement				.622	

Argumentation Subset: (alternative, argument, assumption, decision, requirement)
Percentage of sentences with this classification: 50.8%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.618	.918	.736	.021
	Validation	.612	.915	.733	.032
GA	Training	.621	.939	.745	.004
	Validation	.609	.945	**.741**	.006
All Features		.615	.846	.713	
Inter-Annotator Agreement				.606	

Arguments-All (requirement, argument, and assumption)
Percentage of sentences with this classification: 11.2%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.284	.478	.349	.012
	Validation	.313	.554	.400	.035
GA	Training	.319	.526	.389	.013
	Validation	.333	.554	**.416**	.071
All Features		.133	.969	.234	
Inter-Annotator Agreement				.205	

Alternative
Percentage of sentences with this classification: 32.1%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.496	.745	.594	.003
	Validation	.544	.809	**.651**	.016
GA	Training	.524	.755	.616	.006
	Validation	.555	.764	.643	.011
All Features		.441	.914	.595	
Inter-Annotator Agreement				.391	

Answer
Percentage of sentences with this classification: .7%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.025	.100	.040	.008
	Validation	.069	.333	.114	.030
GA	Training	.100	.100	.100	.024
	Validation	.200	.167	**.182**	.024
All Features		.011	1.000	.021	

Argument

Percentage of sentences with this classification: 6.5%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.243	.677	.352	.016
	Validation	.192	.526	.282	.024
GA	Training	.255	.663	.358	.020
	Validation	.198	.553	**.292**	.034
All Features		.067	.947	.124	

Assumption

Percentage of sentences with this classification: 2.1%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.224	.400	.268	.054
	Validation	.200	.375	**.261**	.035
GA	Training	.343	.483	.364	.042
	Validation	.200	.313	.244	.038
All Features		.029	1.000	.056	

Decision

Percentage of sentences with this classification: 9.8%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.292	.490	.350	.015
	Validation	.281	.439	**.342**	.043
GA	Training	.294	.535	.369	.016
	Validation	.222	.386	.282	.028
All Features		.113	.895	.201	
Inter-Annotator Agreement				.262	

Procedure

Percentage of sentences with this classification: 1.6%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.300	.167	.207	.014
	Validation	.375	.273	**.316**	.052
GA	Training	.350	.167	.217	.010
	Validation	.286	.182	.222	.045
All Features		.018	.909	.035	

Question
Percentage of sentences with this classification: 1.1%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.075	.083	.073	.018
	Validation	.048	.143	**.071**	.016
GA	Training	.084	.217	.112	.020
	Validation	.024	.143	.042	.008
All Features		.013	1.000	.025	

Requirement
Percentage of sentences with this classification: 2.8%

		Precision	Recall	F-Measure	Std. Deviation
ACO	Training	.260	.523	.310	.024
	Validation	.389	.412	**.400**	.066
GA	Training	.256	.615	.346	.020
	Validation	.265	.529	.353	.045
All Features		.031	1.000	.060	

6.3 Generalizability of Results

We use the feature set selected for BR dataset for classifying sentences in SPSD dataset and vice versa. All training F-measures are from 10-fold cross validation with Naïve Bayes classifier. All validation F-measures are from classifying test sentences with the Naïve Bayes classifier. Whichever feature selection method led to higher validation F-measure will be used as optimized feature set, because that was the feature set most generalizable for the dataset it was optimized on. The BR and SPSD datasets differ in many aspects, which makes it interesting to see if the feature sets optimized for one would be effective in classifying data for the other. The SPSD dataset had a higher percentage of rationale than BR. On the other hand, the BR dataset had more than 11x more sentences than the SPSD dataset.

We look at the difference in validation F-measure between using the feature-set optimized for its own dataset, and feature-set optimized for the other dataset. For BR-Binary Rationale the difference was only .036 using SPSD optimized features, while for SPSD-Binary Rationale the difference was .071 using BR optimized features. Thus, for binary rationale, the feature set optimized for SPSD was more general. For BR-Argumentation Subset the difference was .083, while for SPSD-Argumentation Subset the difference was .067. Thus, for argumentation subset, the feature set optimized for BR was more general. For BR-Arguments-All the difference was .057, while for SPSD-Arguments-All the difference was .232. Thus, for arguments-all, the feature set optimized for SPSD was much more general. The feature set optimized for BR was not general enough to effectively classify SPSD-Arguments-All.

Binary Rationale

Data Set		BR Optimized (ACO)	SPSD Optimized (GA)
Bug Report	Training	**.809**	.778
	Validation	**.799**	.763
SPSD	Training	.651	**.768**
	Validation	.676	**.7474**

Argumentation Subset

Data Set		BR Optimized (GA)	SPSD Optimized (GA)
Bug Report	Training	**.616**	.528
	Validation	**.638**	.555
SPSD	Training	.662	**.745**
	Validation	.674	**.741**

Arguments-All

Data Set		BR Optimized (GA)	SPSD Optimized (GA)
Bug Report	Training	**.490**	.413
	Validation	**.454**	.397
SPSD	Training	.171	**.389**
	Validation	.184	**.416**

6.4 Comparing Results

These thesis experiments are utilizing more features than were used in previous research [19]. Additionally, we were able to run multiple trials of each experiment, while previous experiments were only run once. Thus, direct comparisons of those results to the results in this thesis cannot be made. However, we can still look at the results to see whether it was due to improving the extraction of features in the preprocessing step, or due to the new implementations of the ACO and GA, that the experiments in this thesis have improved upon the previous research results. For this section, comparison will refer to comparison against research published in Rogers, et al., 2017. Rogers, et al., only published results for argumentation subset, so we will compare results from just this rationale subclass. We can see that now for both datasets, for the argumentation subset, both training and test F-measures achieved in this thesis are have surpassed the inter-annotator agreement score. Further, the simple solution of just using all features without any selection procedure surpasses the previous GAFFS results. For all precision, recall, and F-measure for Bug Reports, and for recall and F-measure for SPSD, the all-feature results are better than GAFFS.

In the comparison of the feature selection experiments with Bug Reports, we can see the improvements in F-measure are predominantly due to an increase in precision. The previous GAFFS test had a precision of only .288 which significantly lowered the F-measure. On the other hand, in the results from this thesis, the precision was .507 and .520. For SPSD the opposite seems to be true. The recall for test is only .246 in the previous results. Yet the recall for test from the thesis is .945 and .915. This contributes to the much higher F-measure in the thesis compared to GAFFS, because the precision is actually slightly lower than with GAFFS.

Comparison: Chrome Bug Reports – Argumentation Subset

Strategy		**Precision**	**Recall**	**F-Measure**
Thesis GA	Train	.490	.831	.616
	Validation	.520	.827	.638
Thesis ACO	Train	.478	.849	.610
	Validation	.507	.843	.633
Thesis All Features	Validation	.339	.986	.504
GAFFS	Train	.446	.815	.576
	Validation	.288	.859	.432
Inter-Annotator Agreement				.461

Comparison: SPSD – Argumentation Subset

Strategy		**Precision**	**Recall**	**F-Measure**
Thesis GA	Train	.621	.939	.745
	Validation	.609	.945	.741
Thesis ACO	Train	.618	.918	.736
	Validation	.612	.915	.733
Thesis All Features	Validation	.615	.846	.713
GAFFS	Train	.575	.752	.652
	Validation	.630	.246	.354
Inter-Annotator Agreement				.61

Comparison: Generalizability – Argumentation Subset

Data Set	BR Optimized F-Measure		SPSD Optimized F-Measure	
	Thesis: GA	**Prev: GAFFS**	**Thesis: GA**	**Prev: GAFFS**
Bug Report Training	**.616**	.576	.528	.422
Bug Report Validation	**.638**	.432	.555	.439
SPSD Training	.662	.625	**.745**	.562
SPSD Validation	.674	.466	**.741**	.354

6.5 Feature Categories Selected

Below are the resulting feature categories selected by the feature selection algorithm in each SPSD-GA, SPSD-ACO, BR-GA, and BR-ACO experiments for each valid rationale class. If (n) precedes the feature category name, then that feature category contains n-grams of feature instances. If {m} proceeds the feature name, then the feature includes instances of that category from the m-adjacent sentences. See appendix B for the full list of feature category abbreviations.

I counted how many times each feature category was selected overall throughout all the experiments. The most frequently selected feature categories and frequencies respectively were 'to': 68 (*to*), 'wp': 64 (wh-pronoun), 'ex': 34 (existential *there*), 'cc': 33 (coordinating conjunction), 'md': 31 (modal), 'uh': 29 (interjection), 'nnps': 29 (proper noun, plural), 'jjs': 29 (adjective, superlative), 'rbr': 28 (adverb, comparative), and 'pdt': 28 (predeterminer). Thus, these feature categories were chosen most often as being best indicators of rationale in documents. I also analyzed the breakdown of n-grams and m-adjacent categories. Overall unigrams

were favored with 486 unigram categories, compared with 110 5-grams categories – the next highest n-gram represented. There was an approximately equal split between categories from m-adjacent sentences: with 304 categories from 3-adjacent sentences, 293 from single sentences, and 283 from 2-adjacent sentences.

Below I present the selected categories for each valid rationale class of experiments. The "common" features indicated reflect the most frequent feature categories and the respective frequencies for that valid rationale group. The top categories only includes those with frequency greater than or equal to 4.

Binary Rationale

SPSD-GA:
[{2}adverbs, {2}C_sentLength, wdt, {3}wdt, ls, {2}(2)jjs, {3}(2)jjr, domain, (3)roots, {3}wp, unigrams, {2}vb, {2}nnp, {2}ex, {3}(4)nnps, {3}rbs, (3)ls, {3}rbr, {2}(3)cd, rbr]
SPSD-ACO:
[{3}(5)prp$, {2}vb, {2}(4)rb, (3)nnp, (2)ls, {2}(5)pdt, uh, (2)unigrams, (3)rbr, (2)roots, roots, (5)general, {3}(3)vbz, {2}sym, (3)unigrams, {3}cc, wp, {3}nn, {3}wp, {2}cc]
BR-GA:
[{2}(5)domain, (4)in, {3}C_sentLength, {2}nouns, wdt, {3}(2)wp$, (2)wp, {3}(4)wp, (2)uh, (2)vb, {2}(5)nouns, wrb, {3}(2)vb, {2}nnp, {2}(2)nns, {3}(2)nns, {2}(2)vbz, {3}wp$, {2}(4)adverbs, {2}(3)pronouns]
BR-ACO:
[{2}(2)rb, {2}nns, vbd, {2}(3)rb, {2}(4)to, {2}prp$, {2}(2)vbz, {3}(2)nouns, {3}(4)verbs, {2}(2)nnp, (4)ex, (4)vbn, {3}vbg, {3}(2)wp, (2)to, {3}to, {3}(2)to, {2}to, to, wp]

Common: 'wp': 7, 'to': 6, 'nnp': 4, 'vb': 4

Argumentation Subset

SPSD-GA:
[(5)md, {3}(5)poss, {2}(2)verbs, (3)dt, {2}C_sentLength, {2}(4)adverbs, (4)rp, {2}(3)vbn, {3}(5)nnps, {3}(5)rbr, {2}md, unigrams, (3)roots, {2}(4)vbd, {2}nnp, {2}(3)cd, {2}(3)jjs, roots, {2}poss, {2}(4)vbz]
SPSD-ACO:
[nouns, (3)roots, {3}(3)rb, {2}(5)uh, wp, verbs, (4)cd, {2}(4)vbn, {3}(2)conjunctions, {3}(5)ls, (5)cd, C_sentLength, (4)vbd, {3}(4)to, {2}(3)pdt, {3}wp, {2}wp, {3}cc, {2}cc, {2}pdt]
BR-GA:
[{3}(2)vbz, {2}sym, poss, {3}(4)cc, (4)wp, {3}(4)to, {3}(4)jjs, {2}wp, {3}wp, {3}(4)conjunctions, (5)vbz, {2}(4)domain, {3}(3)jjs, {2}(4)prp$, (2)rb, {3}(3)sym, {3}nns, {2}(4)vbp, {3}(5)ls, {3}(5)pdt]
BR-ACO:
[(4)domain, {2}(2)wp, {2}(5)ls, cd, {3}(4)conjunctions, {2}(3)md, {3}(3)poss, {3}(4)prp, {3}(4)adverbs, {2}nnp, nns, to, rbr, (3)wrb, (2)vbz, {3}to, {2}to, (2)to, {2}(2)to, {3}(2)to]

Common: 'to': 8, 'wp': 7, 'poss': 4, 'cd': 4, 'vbz': 4

Arguments-All

SPSD-GA
[{3}(4)nns, {2}adjectives, pdt, (3)prp, {2}(5)ex, (4)vbn, {3}cc, (3)nns, md, {2}(2)wrb, {2}pronouns, {3}prp$, (4)prp$, {3}prp, {3}(5)ontology, {2}nnps, (3)wp, {3}(5)wp, vbp, (5)wrb]
SPSD-ACO
[in, {2}(5)pdt, (5)sym, {2}(2)cc, {3}(2)wrb, {3}(5)rbr, conjunctions, {2}(2)vbd, {3}jj, {3}(4)prp, (3)nouns, {3}(3)nnp, {2}(5)vb, {2}(3)ontology, (2)poss, {3}wp, {2}wp, wp, {3}cc, cc]
BR-GA
[{2}vbz, {3}vbz, (4)pdt, {2}(2)wdt, (4)wp$, {3}sym, {2}(3)ex, (5)rbr, {2}(5)rbr, {2}(4)pdt, {2}(4)uh, {3}(5)ex, (4)md, (4)prp, {2}rb, (5)wp, {2}(2)ex, {2}(4)ex, {2}(3)to, {2}(4)rbr]
BR-ACO
[(3)vbn, {2}(4)rbs, (3)adverbs, {3}(4)pronouns, {2}(5)wrb, {2}(4)wrb, (5)adverbs, {3}(2)vbz, {3}(3)rp, {2}(2)to, {2}(3)md, {2}(4)adverbs, {2}rb, {2}(3)vbn, {3}(2)jjr, (2)wp, {2}wp, wp, (2)to, to]

Common: 'wp': 9, 'ex': 5, 'wrb': 5, 'cc': 4, 'rbr': 4, 'pdt': 4, 'prp': 4, 'to': 4

Alternative

SPSD-GA
[nouns, {2}(3)md, vbz, (4)jjr, {3}(3)nnps, {3}(5)nnps, {2}rbs, wp, (2)vb, (2)md, wrb, (3)verbs, {3}(2)nn, vbd, {2}(4)ex, {3}nns, (3)uh, (5)pronouns, {2}(4)unigrams, {2}(5)poss]
SPSD-ACO
[{2}(3)verbs, (4)jjs, {3}(4)verbs, nouns, (5)rbs, {3}poss, {3}(4)prp, {2}(2)fw, {3}(2)jjs, (3)fw, {3}(4)unigrams, {3}(3)ex, (4)pdt, (3)jjs, {2}(2)md, {3}wp, {2}cc, {3}cc, {2}wp, {2}uh]
BR-GA
[(3)md, {2}(2)vbn, {3}(5)prp$, {2}(4)wdt, {2}(2)wdt, (3)to, {3}(2)wp$, {3}(4)wp$, {3}(4)to, {2}(3)rbr, {3}(3)vbp, {3}(2)to, {3}(2)ls, {3}(4)nnps, {2}(2)prp$, {3}vb, {3}(3)rp, {3}rbr, {2}(5)to, {2}(4)rbr]
BR-ACO
[(2)rbr, (3)uh, wp, {2}(2)rp, {3}wp, {3}(4)ex, {2}(2)nnps, (2)to, to, {2}(2)vbg, {2}(2)cc, (5)to, (4)to, {2}(2)vbn, wdt, {2}to, {3}to, {3}(2)to, {2}(2)to, {2}wp]

Common: 'to': 12, 'wp': 6, 'rbr': 4, 'md': 4, 'nnps': 4

Answer

SPSD-GA
[{2}pdt, {3}cc, {2}(5)vbn, {3}(3)fw, {2}(4)jjs, (5)vbp, (3)nouns, (5)adjectives, {3}md, {3}(3)jjs, {3}(2)ls, (5)verbs, {3}(5)ontology, (2)nnp, prp, {3}(3)jjr, {2}(5)poss, {3}(3)wrb, {2}(4)sym, {3}ontology]
SPSD-ACO
[(2)vbn, {3}(4)cc, {3}(2)dt, {3}(4)wp$, {3}(5)jjr, {3}(4)jjr, {2}(4)ontology, (5)nns, {3}(2)wdt, {2}(5)rp, {2}md, {2}(2)general, (4)ex, {2}(2)vbd, (3)nouns, {3}vbz, {3}conjunctions, {3}pronouns, {2}vbz, {3}dt]
BR-GA
[{3}(3)ls, {2}(5)prp, {2}(2)pdt, (4)jjr, {2}jjs, (4)wrb, {2}(2)uh, (4)ls, (4)md, {2}nn, nn, {3}(3)ex, {3}(3)rp, {3}nnp, {2}(4)ex, (4)rbr, {3}(4)rbr, {3}(5)ls, {3}(4)vbd, {3}jjs]
BR-ACO
[{2}nn, {3}(4)jjs, nnp, {3}(5)wdt, {2}(5)rbr, (4)ls, (5)pronouns, {3}(4)wrb, (2)wp$, {2}(5)wrb, {3}to, {3}(3)prp$, {3}(5)vbz, {3}(5)poss, {3}(4)jjr, {2}to, to, {2}(2)to, (2)to, {3}(2)to]

Common: 'to': 6, 'jjs': 5, 'jjr': 5, 'ls': 5, 'wrb': 4

Argument

SPSD-GA
[adverbs, {2}jjr, {3}(3)ex, {2}(4)wp$, {3}sym, {3}poss, {3}cc, {2}(2)poss, uh, {2}(2)uh, {3}(4)wrb, {3}(2)wp$, {2}conjunctions, {3}ex, (2)fw, {2}(4)vbp, {3}(4)fw, (5)poss, (4)nns, {3}(4)ex]
SPSD-ACO
[{3}(5)jjs, (2)rp, {3}dt, {2}(4)nnp, (4)md, rb, {2}(4)vbg, {3}(4)prp$, (5)prp$, {3}general, {2}wp, {2}(4)uh, conjunctions, {2}(5)rbr, {3}(5)uh, wp, {3}cc, {3}wp, {2}dt, {2}cc]
BR-GA
[vbz, {3}(5)vbd, {2}(2)wdt, (5)ex, {2}(5)vbg, sym, wp$, (5)vbp, (2)wp, {3}(4)jjs, (4)wp, (4)prp$, {2}(4)prp, {2}ex, {3}(3)pdt, {3}rb, {2}(2)nnps, {3}(2)nnps, {3}(4)rbr, (5)pronouns]
BR-ACO
[{2}(4)prp, (4)prp, {3}(2)rb, {3}(3)prp$, (5)vbd, {2}(5)vbz, {3}(5)wrb, {3}(2)ls, {2}nnps, (3)rbs, {2}vbz, {3}(2)to, (5)md, {2}(5)rp, {2}(2)to, to, {2}to, (2)to, {3}to, wp]

Common: 'wp': 6, 'to': 6, 'ex': 5, 'prp$': 4, 'uh': 4

Assumption

SPSD-GA
[vbz, {3}rp, (2)ontology, {3}pos, {2}(4)jjr, (2)to, poss, {2}conjunctions, {3}conjunctions, {3}(4)domain, {2}(4)prp$, {3}(2)domain, {3}(2)ex, {3}(5)rp, {3}(5)jj, (4)nnp, (3)sym, {3}(5)wp, vbp, (5)to]
SPSD-ACO
[{3}pos, {2}(4)ex, (5)rbs, domain, {2}dt, {2}wp, {3}vbz, {2}(3)ex, {3}(5)to, (4)adjectives, {3}(5)nnps, vb, (2)pronouns, {2}(3)vbd, {3}dt, {2}cc, {3}wdt, {3}cc, {3}wp, cc]
BR-GA
[{2}(5)vb, {2}vbz, (4)jjr, (3)md, (3)rbr, {3}(3)nnps, (4)uh, {3}wp, {2}(2)md, {2}(4)vbd, {2}(5)wdt, {3}(5)jjr, {2}(4)rbr, {2}(3)jjs, (4)rbs, {3}(5)ls, (5)rbs, (2)nnps, {3}(5)rp, (5)rbr]
BR-ACO
[{3}(5)vbg, {3}(4)md, {3}(5)rbr, {2}(5)vbz, {2}(5)rp, {3}(3)pdt, {2}vbz, (5)wp, {2}(4)rbr, {3}to, {2}(5)wp, wp, (4)rbs, {3}(2)ex, sym, {3}(2)to, (2)to, {2}to, to, {3}wp]

Common: 'wp': 8, 'to': 8, 'vbz': 5, 'rbr': 5, 'ex': 4, 'rbs': 4, 'rp': 4

Decision

SPSD-GA
[rp, {3}(3)vb, {2}(4)wdt, {3}(2)wdt, {2}nnps, {2}jjs, (2)cd, {3}(2)pos, {2}(4)jjs, {3}(5)general, {3}(5)rbs, nn, {3}md, {3}(5)ontology, {2}(3)wdt, (3)pdt, (5)sym, {2}(4)fw, (3)pronouns, vbp]
SPSD-ACO
[(5)jj, {2}(4)vbn, {2}cc, {3}prp, {3}wp, (5)cc, {2}nouns, (3)wrb, {3}wdt, {3}(5)nns, {3}adjectives, (2)prp$, (4)vb, {3}nnps, ontology, {3}cc, {3}in, {3}(2)pos, {3}pos, {2}dt]
BR-GA
[{2}(5)md, {3}(5)vbd, (3)ex, {3}(4)pdt, {3}(5)nnp, (4)nnps, ls, (4)poss, {2}(4)uh, wp, {3}(5)wp$, (5)rp, {3}(4)vbd, ex, {2}(5)pdt, {3}(3)pdt, {2}(3)wp$, nns, {3}(5)wdt, (3)poss]
BR-ACO
[{2}(5)md, {3}(5)vbd, (3)ex, {3}(4)pdt, {3}(5)nnp, (4)nnps, ls, (4)poss, {2}(4)uh, wp, {3}(5)wp$, (5)rp, {3}(4)vbd, ex, {2}(5)pdt, {3}(3)pdt, {2}(3)wp$, nns, {3}(5)wdt, (3)poss]

Common: 'pdt': 7, 'wdt': 6, 'ex': 4, 'vbd': 4, 'poss': 4, 'nnps': 4, 'wp$': 4

Procedure

SPSD-GA
[wdt, {2}to, {3}poss, uh, {3}(2)uh, (3)poss, {3}(2)ls, {3}(3)rbs, {2}(2)vbd, {3}(4)prp$, {2}(3)wdt, {3}(5)jj, nnp, (3)wp, {3}(2)sym, (5)conjunctions, (5)nnps, {2}(3)prp$, (5)poss, {2}(2)fw]
SPSD-ACO
[{3}(5)ls, {2}cc, (5)jjr, {2}(3)sym, {3}(4)ex, {2}(5)adjectives, (5)jjs, {3}(3)vbn, (5)vbz, nnp, {3}(2)general, (5)cc, (4)ontology, {3}(2)ontology, {3}(4)vbn, wp, {2}wp, {3}wp, {3}wdt, {3}uh]
BR-GA
[(5)md, {2}(5)prp$, {3}pdt, {2}(3)ex, (4)wp$, {3}nnps, (4)wrb, {3}(4)ls, {2}(2)vb, nnp, {2}(5)rp, {2}(5)wdt, (2)ex, {2}(3)jjr, {3}(2)nnps, (3)uh, (5)wrb, {2}(4)rbs, {3}(3)ls, {2}(5)wp$]
BR-ACO
[{2}(3)nnps, {2}nnp, {3}to, (4)vbg, (5)poss, {3}(4)pdt, (2)vb, {2}(2)to, {3}(4)vbg, {3}(5)vb, {3}(4)wrb, {3}(2)jjs, {2}(5)jjs, {2}(4)uh, {2}(4)vbn, {2}to, to, (2)to, {3}(2)to, {3}(4)ls]

Common: 'to': 7, 'ls': 5, 'uh': 5, 'poss': 4, 'wdt': 4, 'nnps': 4, 'wp': 4, 'nnp': 4

Question

SPSD-GA
[(4)verbs, {2}(5)md, {2}(3)vbd, {3}(4)rp, {2}(4)pdt, (4)wp$, {3}(3)nnp, (3)rb, {3}jjs, {2}(5)rbr, {2}(3)rbs, {2}prp$, {2}(5)adjectives, (3)ontology, (5)jjr, {2}nns, (5)uh, {2}(2)ls, {3}wp$, {2}ex]
SPSD-ACO
[{3}(5)wrb, (4)domain, {2}(3)jjr, (4)nn, (4)ontology, {2}cc, {3}cc, {3}pos, {3}(3)wdt, (5)md, cc, {2}jjs, {2}(4)nns, {3}(4)rp, (2)in, {3}wp, {3}wdt, {2}wdt, dt, {3}dt]
BR-GA
[wdt, {3}(5)vbd, {2}(2)pdt, {3}(3)ex, {2}(4)jjr, {3}wdt, {3}(4)jjr, {2}(2)wp, {3}(3)rbr, {3}(5)rbr, {2}(3)wp, {3}(5)nnps, {2}(3)rbr, (2)wp, (3)uh, (4)ex, {3}(3)pdt, {2}(5)jjr, (3)wp, {2}(5)poss]
BR-ACO
[{3}(4)poss, (3)poss, {3}(2)wdt, {2}(4)nnps, {2}poss, {2}(5)wp, jjs, (5)poss, {2}(5)jjr, (4)rbs, {3}(4)rbs, {2}wp, (3)nnps, {2}(5)vbg, (5)uh, {3}(2)to, {3}to, {2}to, to, {2}(2)to]

Common: 'wp': 7, 'wdt': 6, 'jjr': 6, 'poss': 5, 'to': 5, 'rbr': 4

Requirement

SPSD-GA
[{3}vbp, {2}adjectives, {3}rp, (5)dt, {2}cc, (3)vbn, {3}(2)jjr, {2}(5)nnps, {3}(3)nnps, uh, md, {2}uh, (5)vb, {3}prp, {3}(5)rp, {2}(5)sym, {3}(5)jjs, {3}(4)rbs, (5)wp, (3)nnps]
SPSD-ACO
[{3}md, (5)cd, {2}(5)nnp, (2)vb, {2}(2)jjs, {3}(3)vbd, {2}pronouns, (4)vbz, {2}(3)ls, {2}(5)nns, {3}(3)prp$, {2}(3)wp$, (2)jjs, (4)vbp, {3}vbp, {2}cc, {3}cc, cc, {3}wp, {2}wp]
BR-GA
[{2}(5)uh, (3)ex, (5)wp$, {2}cc, {2}sym, (4)wp, {3}(5)nnps, {2}(5)vbp, {3}wp$, {3}md, {2}(3)wp, (3)uh, {3}(3)wp, {3}(4)ex, (5)jjs, {2}(3)jjs, {3}rbs, {2}(3)uh, {3}(5)uh, {3}(4)wp]
BR-ACO
[{2}(5)poss, pdt, {3}pdt, md, {3}(5)nnps, {2}(2)md, to, {2}rbs, {3}(2)rbs, {2}pdt, {3}(5)vbn, {2}md, {2}(2)jjs, {3}(2)md, {2}(3)rbr, {3}to, {2}(2)to, {2}to, {3}(2)to, (2)to]

Common: 'md': 7, 'wp': 7, 'jjs': 6, 'to': 6, 'uh': 6, 'cc': 5, 'nnps': 5, 'vbp': 4, 'rbs': 4

6.6 Evaluation of Results

We can see that there is a benefit to feature selection because for every experiment, the best F-measure with feature selection is higher than without. Using feature selection improved precision significantly in all valid rationale classes where percentage of rationale was less than 50%. Thus the hypothesis that there are noisy, redundant features that hurt classification is supported. Further, we achieved better F-measure than all inter-annotator agreement F-measures given except BR-Decision. (BR-Decision-GA was an outlier in that it was also the model that seemed to over-fit the most.) Thus, although automatic extraction is not perfect by any means, it is as effective as manual extraction. That is very promising for the feasibility of utilizing the automatic extraction technique for any future application.

One result that occurred was that in some experiments the validation score was higher than the training score. I hypothesize that for experiments where the test scores were higher than training scores, this disparity is due to the number of sentences for which each respective model is trained. The training models use 10 fold cross validation of training sentences, meaning each model is trained with 9/10 of 70% of the sentences – or 63% of the sentences. On the other hand, the models from which the test scores are evaluated are trained with all of the training sentences – or 70% of the sentences. Thus it makes sense that the test scores sometimes surpass the training scores because the model learns over more total instances. This is just one possible explanation, and it is also likely that some of the fault lies in the random split of the training and test sentences.

Chapter 7 Future Work

An important future extension of this work is integrating the automatic design rationale capture algorithms into a robust tool for software engineers, which would provide an interface to help them best utilize the extracted information. This could take the form of an application in which users upload new documents to the interface, or could be an add on to a software documentation management system already in use. To optimize the user experience with such a tool, it would be important to focus on refining the machine learning algorithm to improve the precision of classification. With a less precise classifier, more non-rationale data gets captured along with the actual design rationale. It is more cumbersome for a user to utilize the relevant information from the design rationale when there is extra non-rationale data accidentally captured. Precision could be improved by using the precision statistic as the measure of fitness instead of F-measure. It would be interesting to investigate how this effects the recall of the resulting classifier, and it would be useful to see how dramatic the tradeoff in recall would be from optimizing for precision.

Also interesting would be to explore what feature sets are selected when the length of the subset is not fixed to 20 features in feature selection. Allowing more flexibility in this respect could improve the results of the feature selection algorithm. It would be important to penalize candidate solutions that had more features, to try to find a short and optimal feature set. I suspect that the runtime of the GA and ACO would increase significantly, as data including more features take longer to classify.

To further probe the flexibility of the rationale capture technique, it could be valuable to try combining the SPSD and BR datasets. We would be able to use more data for feature selection and training the classifier if we combined the data from both sources before splitting the sentences into the training and testing portions. This might produce a more general model, since the classifier built will be incorporating features from two different sources.

Finally, for this thesis we started with the preprocessed sentences.csv databases for BR and SPSD data. It could be valuable to refine the preprocessing step in order to examine if there are other techniques for extracting relevant features that have not been tried yet. The data and features available for classification are very crucial for the performance of the classifier, so this could have a significant effect on the success of the automatic design rationale extraction procedure.

Chapter 8 Conclusion

In this study I answered my four guiding research questions:

1. **What parameters are best for selecting the optimal feature set with the ant colony optimization algorithm?**

   In section 5.2, the selection of these parameters was discussed. Although parameters were assigned based on analogous parameters for the GA when possible, some were completely unique to ACO. The number of ants was set to 100 to be parallel to population size in the GA. The length of feature subset was fixed at 20, and was comparable to chromosome length for the GA. The top k=20 ants were selected to update pheromone concentrations on the paths they constructed, corresponding to the 20% selection rate of chromosomes as parents in the GA. After experimental trial and error, pheromone evaporation rate was set to .1. The probabilistic transition rule parameters for relative weight of pheromone intensity and local importance were set respectively as 2 and 1 – giving more influence to pheromone intensity for feature set construction. To preserve high quality solutions, ¾ of features were chosen from k best ants of the previous generation, and ¼ of features were chosen to maximize the update selection measurement rule.

   The selection of ACO parameters, had a strong impact on the ACO rate of convergence. In early experiments with some variations on these parameters,

the ACO would converge on poor solutions quickly, leading the iterations to be wasted computational effort and to a selection of a suboptimal feature set. In these cases, ants were getting stuck in poor local optima. With the correct parameters selected, the ACO was able to keep exploring the search space as intended. The first graph in Appendix C, BR-ACO-Binary Rationale, illustrates that at iteration 33 of the ACO experiment, the ants were still discovering new optimal feature sets. This was the searching behavior we hoped to find with the ACO metaheuristic

2. **What textual feature categories do the ant colony optimization algorithm and genetic algorithm select?**

   As discussed in section 6.5, the most frequently selected feature categories and frequencies respectively were 'to': 68 (*to*), 'wp': 64 (wh-pronoun), 'ex': 34 (existential *there*), 'cc': 33 (coordinating conjunction), 'md': 31 (modal), 'uh': 29 (interjection), 'nnps': 29 (proper noun, plural), 'jjs': 29 (adjective, superlative), 'rbr': 28 (adverb, comparative), and 'pdt': 28 (predeterminer). Overall unigram features were favored over the other n-gram features. There was an approximately equal split of categories chosen from single, 2-adjacent, and 3-adjacent sentences.

3. **How does the ant colony optimization selected feature set compare to the genetic algorithm selected feature set for classification?**

   In sections 6.1-6.2 we compare the results of the ACO and GA experiments for both datasets. For the BR dataset: binary rationale, alternatives, and decisions had higher validation F-measures from the model generated with ACO than with GA. For the SPSD dataset: alternative, assumption, decision, procedure, question, and requirement had higher validation F-measure from the model generated with ACO than with GA. Thus for only 9 out of 22 different experiments, the ACO outperformed the GA. In the remainder of experiments, GA outperformed the ACO. However, the results from the ACO and GA were actually generally very close. For example, for the BR-Argumentation Subset experiment, the difference between GA and ACO F-measure was .638 and .633 respectively.

4. **How generalizable are models created for a specific dataset?**

   In section 6.3, the generalizability of the results are discussed. The feature set optimized for SPSD was extremely general. Classifying the BR data with the feature set optimized on SPSD resulted in only a .036 reduction in F-measure. The resulting F-measure of .763 classifying BR data was still higher than the inter-annotator agreement of .524. The feature set optimized for BR was comparatively slightly less general, yet still effective. Classifying SPSD data with feature set optimized for BR led to a .071 reduction in F-measure. Again, the resulting F-measure of .676 was higher than the inter-annotator agreement

of .606. These results are very promising for future applications in which we would want to classify data from new sources of documentation, but do not have the resources to manually annotate them.

Since the ACO performed comparably to the GA, and took half the time on average to run on the large BR dataset, the ACO overall would be advantageous to use compared to the GA. The efficiency of the ACO means it would be an especially good choice for applications with larger datasets. We see that there is a clear benefit to feature selection because for every experiment, the best F-measure with feature selection is higher than without feature selection. Feature selection especially improved precision results. In all the conducted experiments except BR-decision, we attained a higher F-measure than the inter-annotator agreement. Based on these results, a tool created with this machine learning extraction technique would be more effective in capturing design rationale than manual extraction.

Appendix

A. Features Categories Extracted

All features below were extracted, and all features below with each of these prefixes: [(2), {2}(2), {3}(2), (3), {2}(3), {3}(3), (4), {2}(4), {3}(4), (5), {2}(5), {3}(5)] were extracted. (where (n) represents n-grams, and {m} represents m-adjacent sentences)

Penn-Treebank features:
cc, cd, dt, ex, fw, in, jj, jjr, jjs, ls, md, nn, nnp, nnps, nns, pdt, poss, prp$, prp, rb, rbr, rbs, rp, sym, to, uh, vb, vbd, vbg, vbn, vbp, vbz, wdt, wp$, wp, wrb

General parts of speech features:
adjectives, adverbs, conjunctions, nouns, pronouns, verbs

Rationale terms:
general, domain, ontology

Verb roots:
roots

Parts of Speech:
pos

Bag of words:
unigrams

Sentence Length:
C_sentLength

B. Feature Category Abbreviations

Penn Treebank Terms [47]

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

C. Graphs of Bug Reports Experiments Over Iterations

Since the training and test F-measures are not always converging in the first 20 iterations, the graphs show that the decision to continue running experiments to iteration 40 instead of 20 is supported. Note that although test F-m was tracked over the iterations, it was only printed out and not used in any way to influence what feature set was selected.

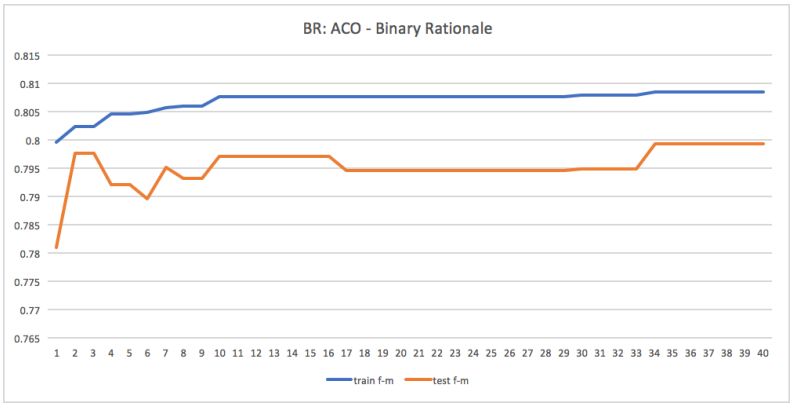

Figure 0-1: BR-Binary Rationale (ACO) Graph

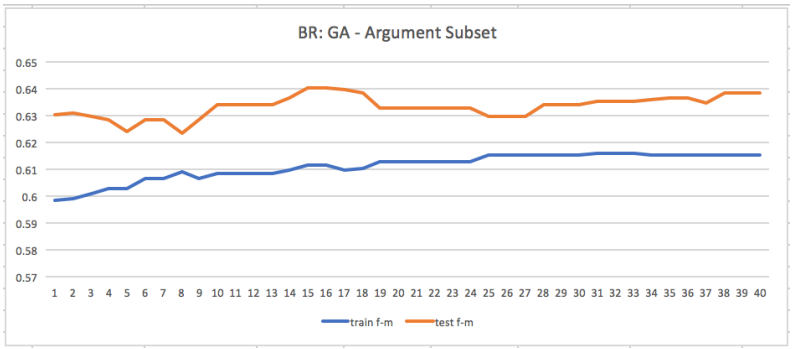

Figure 0-2: BR-Argument Subset (GA) Graph

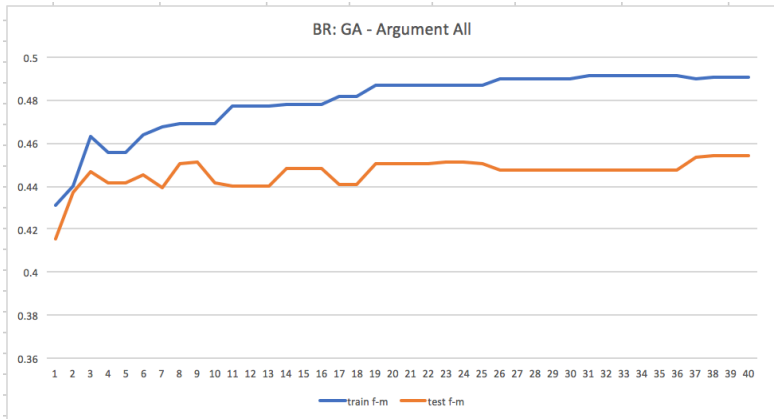

Figure 0-3: BR-Arguments All (GA) Graph

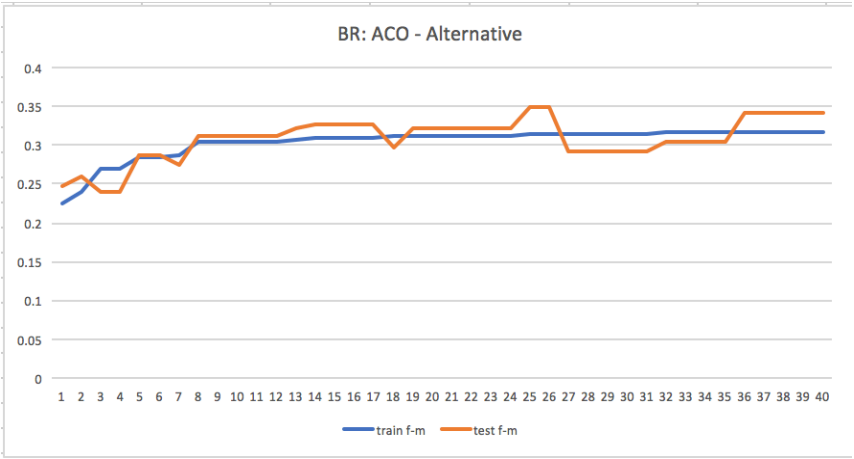

Figure 0-4: BR-Alternative (ACO) Graph

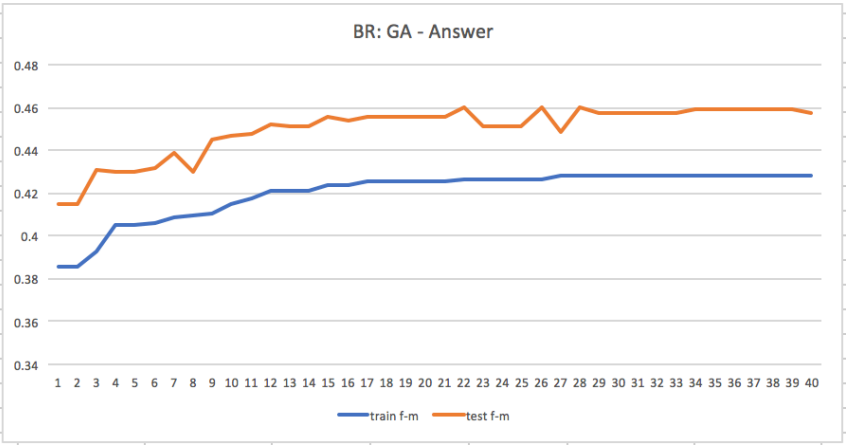

Figure 0-5: BR-Answer (GA) Graph

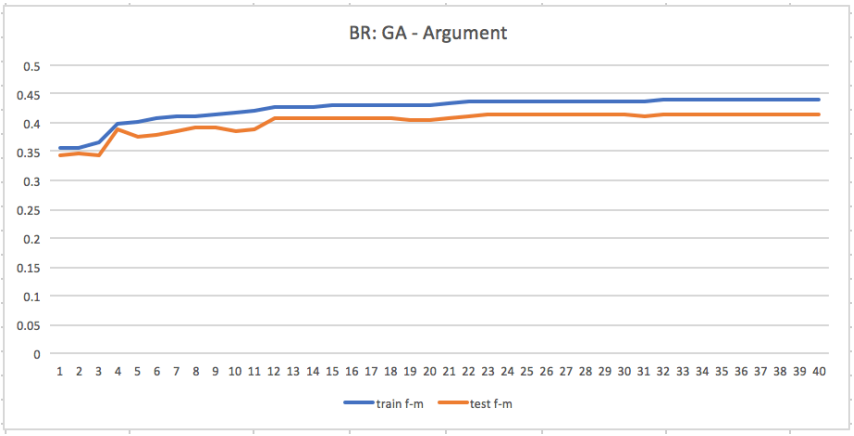

Figure 0-6: BR-Argument (GA) Graph

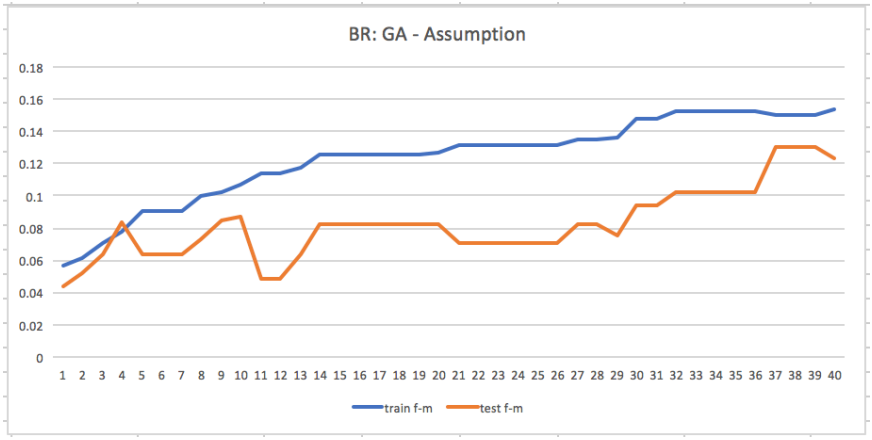

Figure 0-7: BR-Assumption (GA) Graph

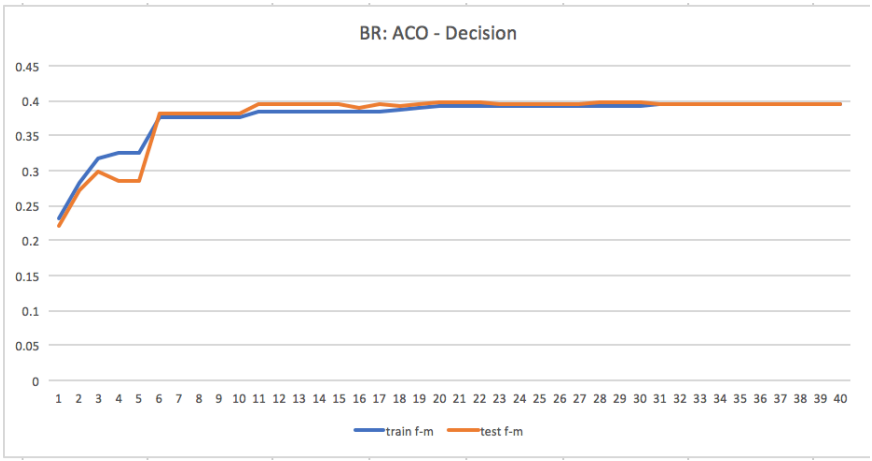

Figure 0-8: BR-Decision (ACO) Graph

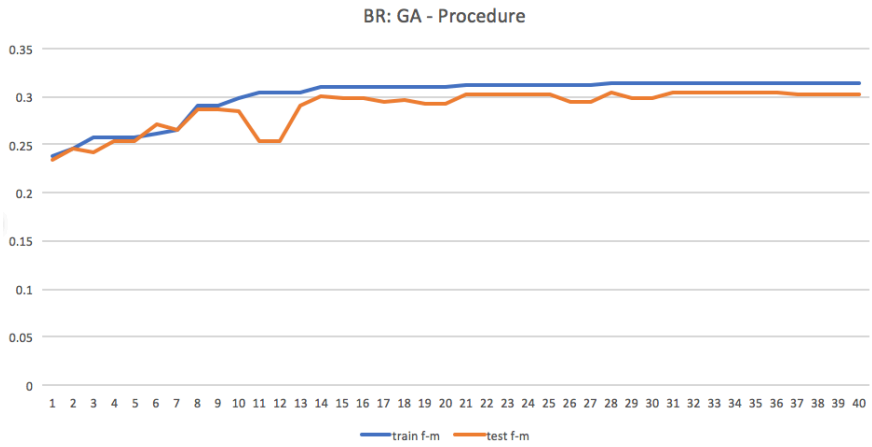

Figure 0-9: BR-Procedure (GA) Graph

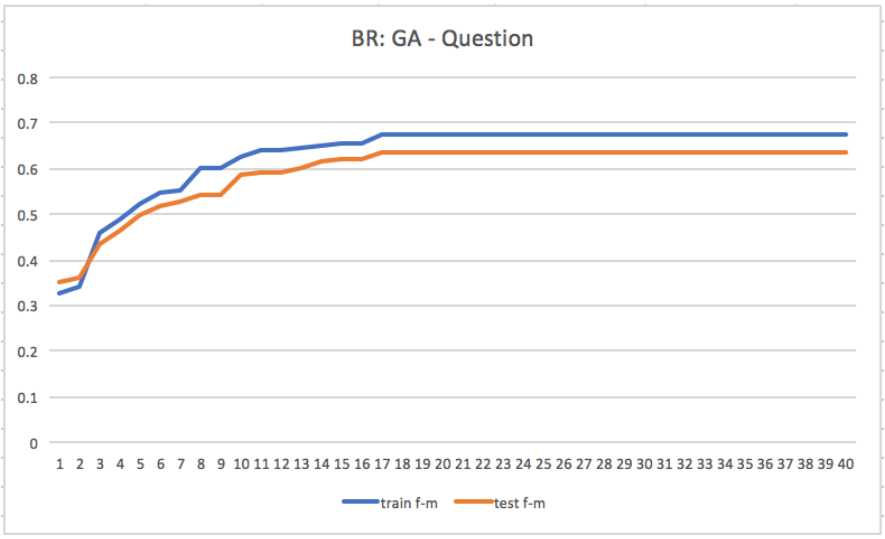

Figure 0-10: BR-Question (GA) Graph

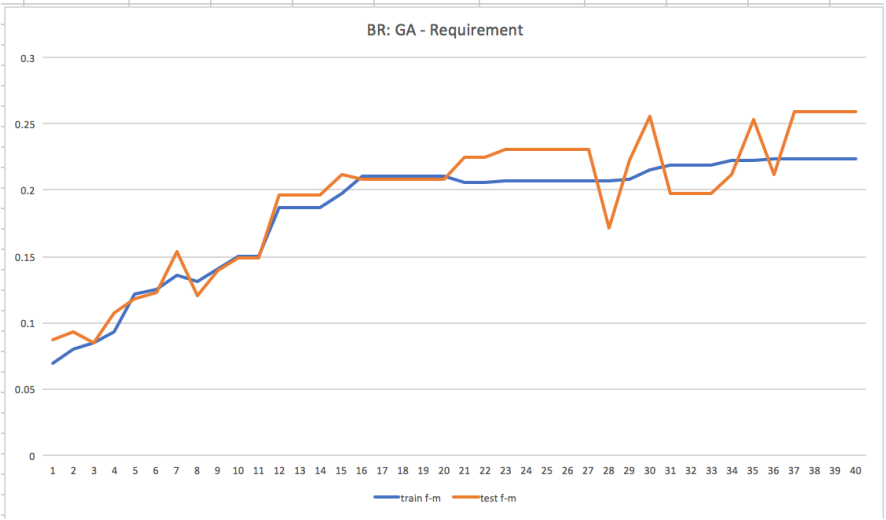

Figure 0-11: BR-Requirement (GA) Graph

D. Graphs of SPSD Experiments Over Iterations

Since the training and test F-measures are not always converging in the first 20 iterations, the graphs show that the decision to continue running experiments to iteration 40 instead of 20 is supported. Note that although test F-m was tracked over the iterations, it was only printed out and not used in any way to influence what feature set was selected.

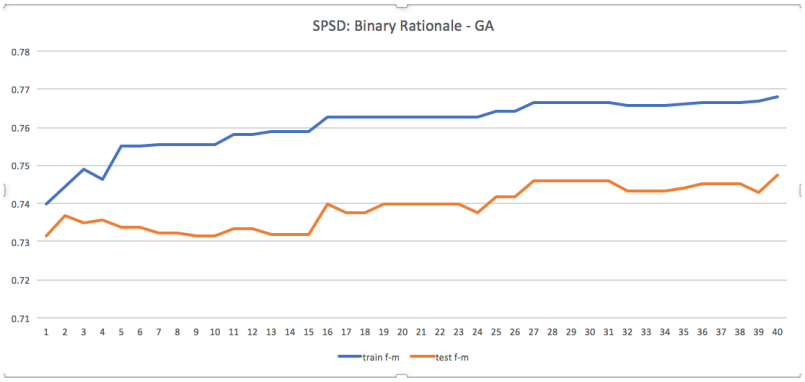

Figure 0-12: SPSD-Binary Rationale (GA) Graph

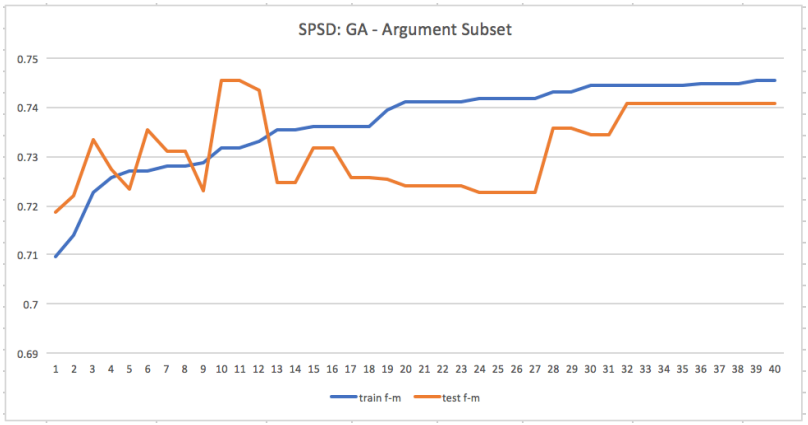

Figure 0-13: SPSD-Argumentation Subset (GA) Graph

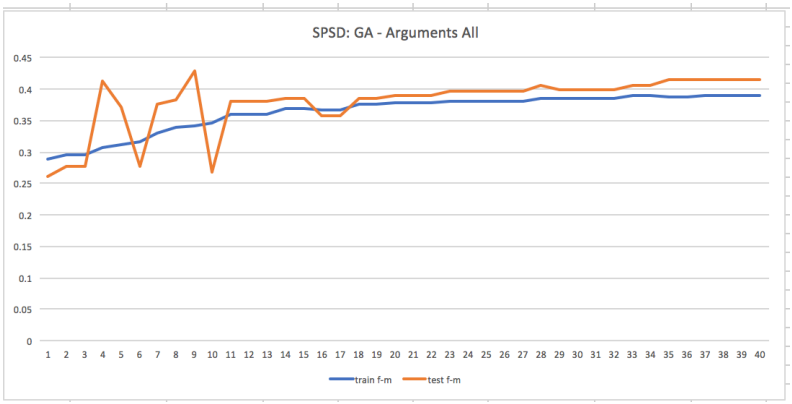

Figure 0-14: SPSD-Arguments All (GA)

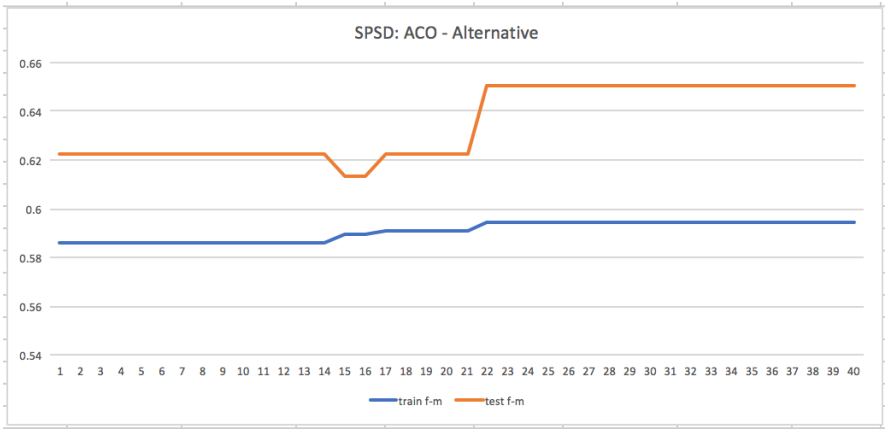

Figure 0-15: SPSD-Alternative (ACO) Graph

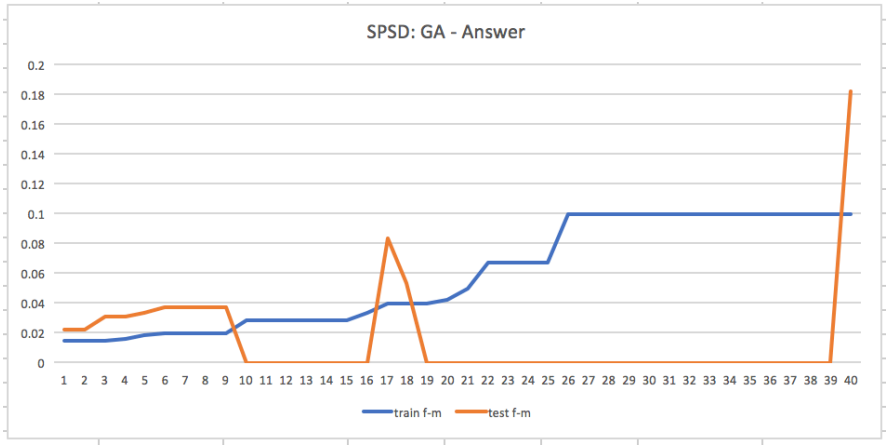

Figure 0-16: SPSD-Answer (GA) Graph

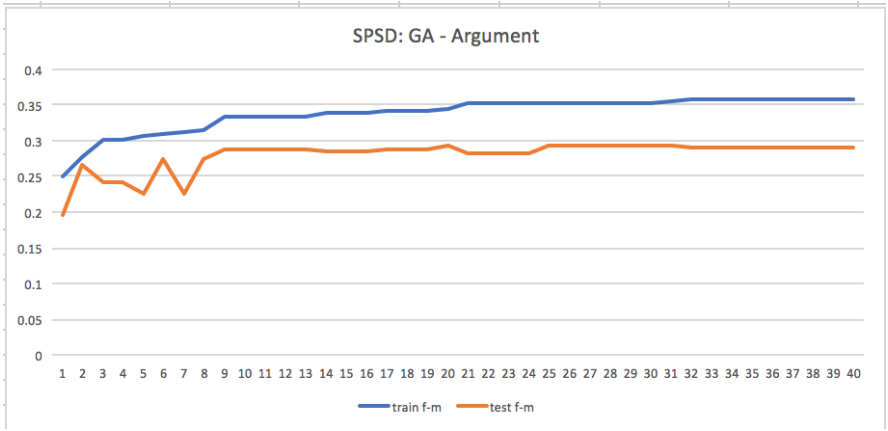

Figure 0-17: SPSD-Argument (GA) Graph

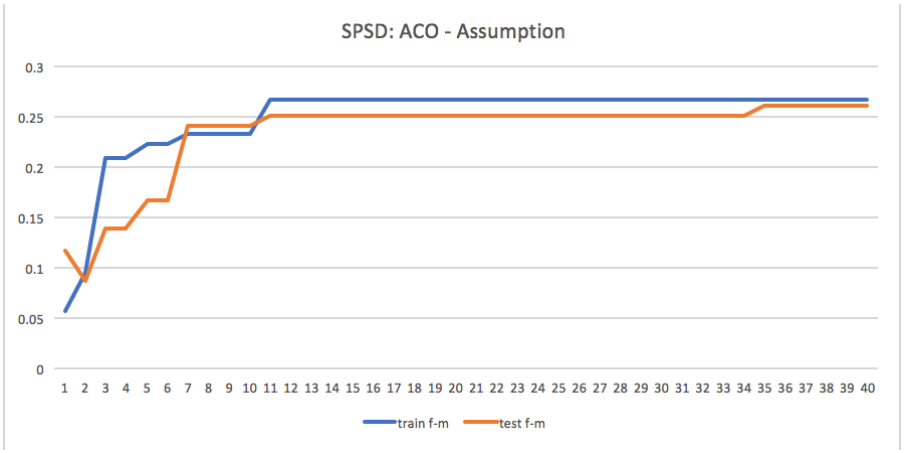

Figure 0-18: SPSD-Assumption (ACO) Graph

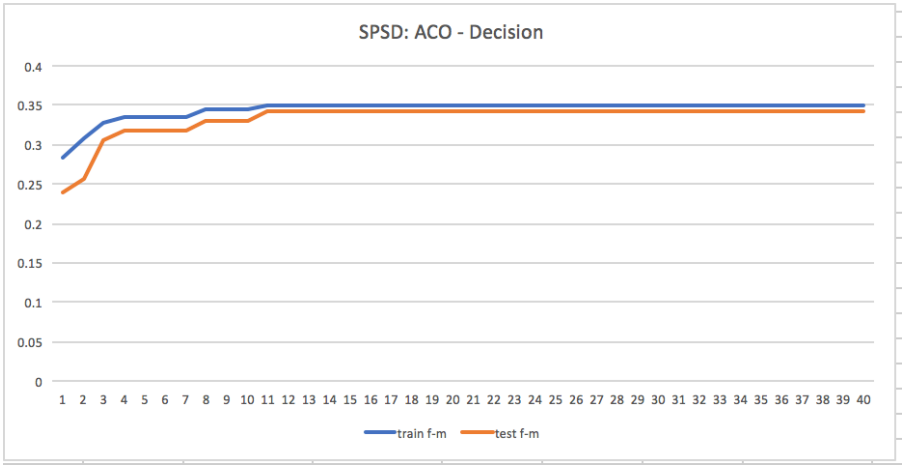

Figure 0-19: SPSD-Decision (ACO) Graph

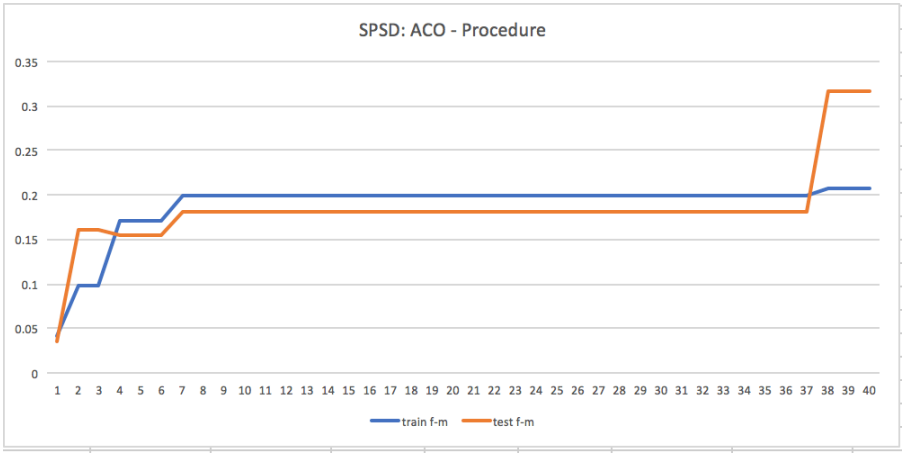

Figure 0-20: SPSD-Procedure (ACO) Graph

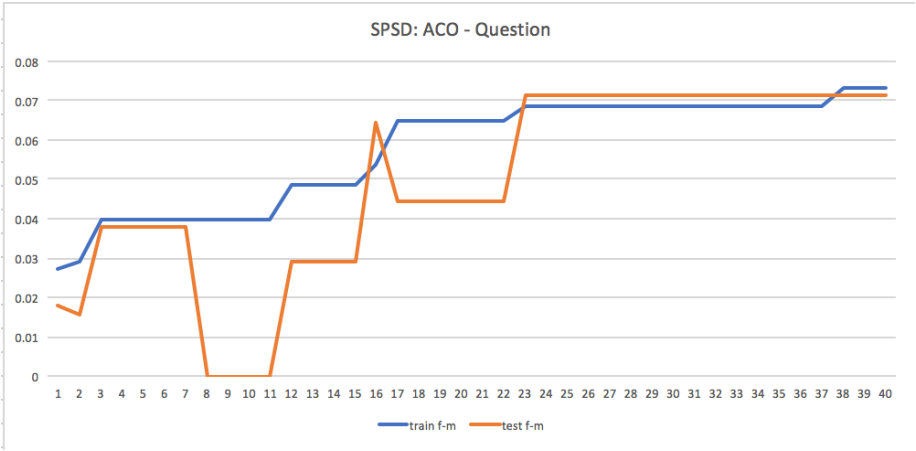

Figure 0-21: SPSD-Question (ACO) Graph

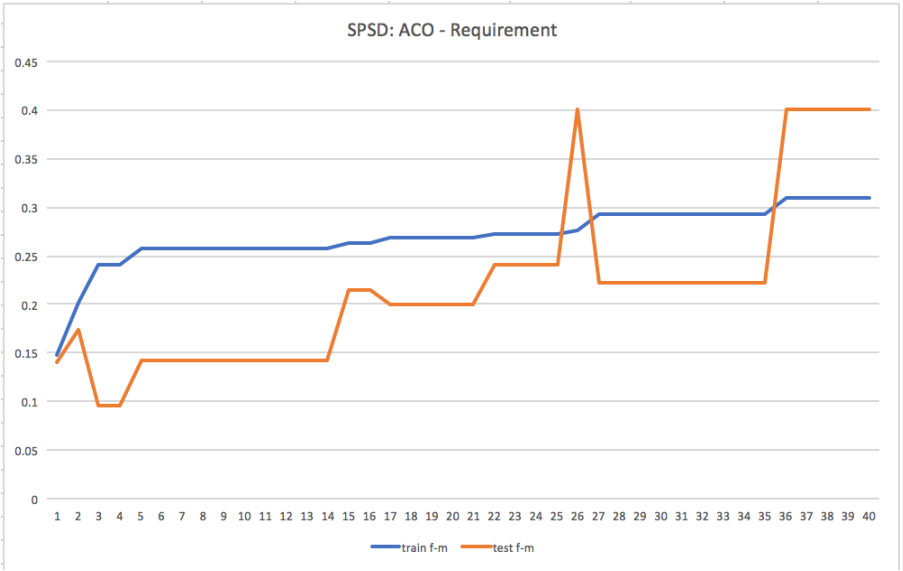

Figure 0-22: SPSD-Requirement (ACO) Graph

Bibliography

1. Lee, J., *Design rationale systems: understanding the issues.* IEEE expert, 1997. **12**(3): p. 78-85.
2. López, C., et al., *Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach.* Science of Computer Programming, 2012. **77**(1): p. 66-80.
3. Burge, J.E. and D.C. Brown, *An Integrated Approach for Software Design Checking Using Design Rationale*, in *Design Computing and Cognition '04*, J.S. Gero, Editor. 2004, Springer Netherlands: Dordrecht. p. 557-575.
4. Lever, J., M. Krzywinski, and N. Altman, *Points of Significance: Model selection and overfitting.* Nature Methods, 2016. **13**(9): p. 703-704.
5. Rogers, *Using Genetic Algorithms for Feature Selection.* 2013: p. Miami University, Oxford, Ohio.
6. Saraç, E. and S.A. Özel, *An ant colony optimization based feature selection for web page classification.* The Scientific World Journal, 2014. **2014**.
7. Aghdam, M.H., N. Ghasem-Aghaee, and M.E. Basiri, *Text feature selection using ant colony optimization.* Expert systems with applications, 2009. **36**(3): p. 6843-6853.
8. Jensen, R. and Q. Shen. *Finding rough set reducts with ant colony optimization*. in *Proceedings of the 2003 UK workshop on computational intelligence*. 2003.
9. Dorigo, M. and T. Stützle, *Ant colony optimization*. 2004, Cambridge, Mass.: MIT Press. xi, 305 p.
10. Al-Ani, A., *Feature subset selection using ant colony optimization.* International journal of computational intelligence, 2005.
11. Dorigo, M., *Optimization, learning and natural algorithms.* Ph. D. Thesis, Politecnico di Milano, Italy, 1992.
12. Deneubourg, J.-L., et al., *The self-organizing exploratory pattern of the argentine ant.* Journal of insect behavior, 1990. **3**(2): p. 159-168.
13. Goss, S., et al., *Self-organized shortcuts in the Argentine ant.* Naturwissenschaften, 1989. **76**(12): p. 579-581.
14. Leguizamon, G. and Z. Michalewicz. *A new version of ant system for subset problems*. in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. 1999. IEEE.
15. Holland, J.H., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. 1975/1992: MIT press.
16. Goldberg, D.E. and J.H. Holland, *Genetic algorithms and machine learning.* Machine learning, 1988. **3**(2): p. 95-99.

17. Martin-Bautista, M.J. and M.-A. Vila. *A survey of genetic feature selection in mining issues*. in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. 1999. IEEE.
18. Mathur, T., *IMPROVING CLASSIFICATION RESULTS USING CLASS IMBALANCE SOLUTIONS & EVALUATING THE GENERALIZABILITY OF RATIONALE EXTRACTION TECHNIQUES.* 2014: p. Miami University, Oxford, Ohio.
19. Rogers, B., et al., *Generalizability of Document Features for Identifying Rationale*, in *Design Computing and Cognition'16*. 2017, Springer. p. 633-651.
20. Atkinson-Abutridy, J., C. Mellish, and S. Aitken, *Combining information extraction with genetic algorithms for text mining.* IEEE Intelligent Systems, 2004. **19**(3): p. 22-30.
21. Wu, Y., et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.* arXiv preprint arXiv:1609.08144, 2016.
22. Mao, F., R.E. Mercer, and L. Xiao, *Extracting imperatives from wikipedia article for deletion discussions.* ACL 2014, 2014: p. 106.
23. Moens, M.-F., et al. *Automatic detection of arguments in legal texts*. in *Proceedings of the 11th international conference on Artificial intelligence and law*. 2007. ACM.
24. Liang, Y., et al., *Learning the "Whys": Discovering design rationale using text mining—An algorithm perspective.* Computer-Aided Design, 2012. **44**(10): p. 916-930.
25. Basiri, M.E. and S. Nemati. *A novel hybrid ACO-GA algorithm for text feature selection*. in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. 2009. IEEE.
26. Yang, Y. and J.O. Pedersen. *A comparative study on feature selection in text categorization*. in *Icml*. 1997.
27. Gheyas, I.A. and L.S. Smith, *Feature subset selection in large dimensionality domains.* Pattern recognition, 2010. **43**(1): p. 5-13.
28. Wang, X., et al., *Feature selection based on rough sets and particle swarm optimization.* Pattern recognition letters, 2007. **28**(4): p. 459-471.
29. Ozyurt, I.B., *Automatic identification and classification of noun argument structures in biomedical literature.* IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 2012. **9**(6): p. 1639-1648.
30. Mukherjee, I., et al. *Content analysis based on text mining using genetic algorithm*. in *Computer Technology and Development (ICCTD), 2010 2nd International Conference on*. 2010. IEEE.
31. Zaiyadi, M. and B. Baharudin, *A proposed hybrid approach for feature selection in text document categorization.* World Academy of Science, Engineering and Technology, 2010. **48**: p. 111-116.

32. Ali, S.I. and W. Shahzad. *A feature subset selection method based on symmetric uncertainty and ant colony optimization*. in *Emerging Technologies (ICET), 2012 International Conference on*. 2012. IEEE.
33. Roeva, O., S. Fidanova, and V. Atanassova. *Hybrid ACO-GA for parameter identification of an E. coli cultivation process model*. in *International Conference on Large-Scale Scientific Computing*. 2013. Springer.
34. Jiang, P.-p., et al. *Optimization of text feature subsets based on GATS algorithm*. in *IT in Medicine & Education, 2009. ITIME'09. IEEE International Symposium on*. 2009. IEEE.
35. Dorigo, M., V. Maniezzo, and A. Colorni, *Ant system: optimization by a colony of cooperating agents.* IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 1996. **26**(1): p. 29-41.
36. Kanan, H.R., K. Faez, and M. Hosseinzadeh. *Face recognition system using ant colony optimization-based selected features*. in *Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007. IEEE Symposium on*. 2007. IEEE.
37. Stützle, T. and M. Dorigo, *ACO algorithms for the traveling salesman problem.* Evolutionary algorithms in engineering and computer science, 1999: p. 163-183.
38. Dorigo, M. and T. Stützle, *Ant colony optimization: overview and recent advances*, in *Handbook of metaheuristics*. 2010, Springer. p. 227-263.
39. Lessing, L., I. Dumitrescu, and T. Stützle. *A comparison between ACO algorithms for the set covering problem*. in *International Workshop on Ant Colony Optimization and Swarm Intelligence*. 2004. Springer.
40. Solnon, C. and S. Fenet, *A study of ACO capabilities for solving the maximum clique problem.* Journal of Heuristics, 2006. **12**(3): p. 155-180.
41. De Campos, L.M., et al., *Ant colony optimization for learning Bayesian networks.* International Journal of Approximate Reasoning, 2002. **31**(3): p. 291-311.
42. Casillas, J., O. Cordón, and F. Herrera. *Learning fuzzy rules using ant colony optimization algorithms*. in *Proc. 2nd International Workshop on Ant Algorithms*. 2000.
43. Manning, C.D. and H. Schütze, *Foundations of statistical natural language processing, Vol. 999*. 1999, MIT Press.
44. McCallum, A. and K. Nigam. *A comparison of event models for naive bayes text classification*. in *AAAI-98 workshop on learning for text categorization*. 1998. Citeseer.
45. Anderson, E.J. and M.C. Ferris, *Genetic algorithms for combinatorial optimization: the assemble line balancing problem.* ORSA Journal on Computing, 1994. **6**(2): p. 161-173.

46. Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning.(1989) AddisonWesley.* Reading, Mass, 1989.
47. Marcus, M.P., M.A. Marcinkiewicz, and B. Santorini, *Building a large annotated corpus of English: The Penn Treebank.* Computational linguistics, 1993. **19**(2): p. 313-330.
48. Burge, J.E., *Software engineering using design RATionale*. 2005, University of Illinois Urbana-Champaign.
49. Hofmann, M. and A. Chisholm, *Text Mining and Visualization: Case Studies Using Open-Source Tools*. 2015: Chapman and Hall/CRC.
50. Kohavi, R. and G.H. John, *Wrappers for feature subset selection.* Artificial intelligence, 1997. **97**(1-2): p. 273-324.