A Study of Text Mining Framework

for Automated Classification of Software Requirements

in Enterprise Systems

by

Japa Swadia

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2016 by the
Graduate Supervisory Committee:

Arbi Ghazarian, Chair
Ashraf Gaffar
Srividya Bansal

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

Text Classification is a rapidly evolving area of Data Mining while Requirements Engineering is a less-explored area of Software Engineering which deals the process of defining, documenting and maintaining a software system's requirements. When researchers decided to blend these two streams in, there was research on automating the process of classification of software requirements statements into categories easily comprehensible for developers for faster development and delivery, which till now was mostly done manually by software engineers - indeed a tedious job. However, most of the research was focused on classification of Non-functional requirements pertaining to intangible features such as security, reliability, quality and so on. It is indeed a challenging task to automatically classify functional requirements, those pertaining to how the system will function, especially those belonging to different and large enterprise systems. This requires exploitation of text mining capabilities. This thesis aims to investigate results of text classification applied on functional software requirements by creating a framework in R and making use of algorithms and techniques like k-nearest neighbors, support vector machine, and many others like boosting, bagging, maximum entropy, neural networks and random forests in an ensemble approach. The study was conducted by collecting and visualizing relevant enterprise data manually classified previously and subsequently used for training the model. Key components for training included frequency of terms in the documents and the level of cleanliness of data. The model was applied on test data and validated for analysis, by studying and comparing parameters like precision, recall and accuracy.

DEDICATION

There are a number of people whom I wish to attribute my accomplishments till date. I would like to dedicate my work to my parents, Charvi and Nimish Swadia, and my brother, Dhiman for their unconditional love, patience and support throughout my graduate study; and most importantly for having faith in my capabilities. I would also like to thank my dear friend, Akshay, for being a constant source of encouragement and for always being at my side, through thick and thin. And finally, a big shout of thank-you to all my close friends at Arizona State University, with whom there are the good times and without whom this grueling endeavor of MS in Software Engineering would not have been so memorable to cherish.

ACKNOWLEDGMENTS

I would sincerely like to thank my thesis advisor, Dr. Arbi Ghazarian for not only introducing and steering me towards my area of interest but also for the continual guidance throughout the period of this Thesis. I also acknowledge the prior research made by him which subsequently helped me lay the groundwork for this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements . . . No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later" [1].

In simple terms, ‘Requirements’ of a software system basically ‘tell’ the builders of that software what is to be incorporated in the software, as per the needs and expectations of the stakeholders.

Requirements engineering is an evolving branch of Software Engineering which deals with the process of defining, documenting and maintaining a software system’s requirements. It emphasizes the use of systematic and repeatable techniques that ensure the completeness, consistency, and relevance of the system requirements [3]. Specifically, requirements engineering encompasses feasibility study, requirements elicitation, analysis, specification, verification, and management. It also demands knowledge of the specific domain the system belongs to.

There are two main types of requirements. Functional requirements define the capabilities that a product must provide to its users, such as feeding name and email address as input, getting a password as output, updating/inserting in the database, etc. while Non-functional requirements describe quality attributes, design and implementation constraints and external interfaces that the product must have, such as security, reliability, stability, etc. The current study focuses on Functional Requirements in particular.

Requirements constitute the primary steps in building any software system, and as entailed in this comprehensive study, are realized as one of the most pertinent aspects of software development. Generally, a detailed Software Requirements Specification (SRS) document is published, which acts as a bible for developing, maintaining and rebuilding a software product. It is a complex process leading up to a fundamental communication problem amongst many others, only because it involves three parties – the user of the system, the developer, and the documenter of system requirements.

Efficient Requirements Management makes way for increased productivity, reduced cost of software product and improves workability of a system, in a way that provides for less amount of rework, that is, work done again to correct the previously wronged work. Industry statistics point that 50% of software project failures are caused due to lack of proper Requirements Engineering practices [1]. It is a known fact that Natural language requirements specifications are the most commonly used form, accounting for up to 90% of all specifications [1]. However, natural language requirements specifications are prone to a number of errors and flaws, in particular due to the ambiguity inherent in natural language. Moreover, there is a dearth of available methods and tools to aid software engineers in managing requirements. As the requirements are written in informal natural language, they cannot be easily analyzed for defects, such as inconsistency or ambiguity. [10]

Although a relatively new domain, Requirements Engineering has gained a lot of momentum in the past decade, and has become an indispensable process for developing a piece of software. Moreover, it is a continuous process during the life-cycle of a software, and ultimately provides an insight into the technicalities of the same.

In particular, the area Requirements Classification holds prime importance, and is gaining popularity amongst researchers and data scientists who are aiming to make this tedious process simpler. Requirements Classification essentially involves documenting or specifying the needs and constraints of user in a clear and precise manner. Requirements statements need to be classified in order to provide clarity to developers who are working to build the software. Although the classification process itself aims to make the job easier for developers, in reality that's not the case. High amount of manual labor and time are involved in sifting through requirements documents and classifying requirements statements into categories comprehensible to developers for building pieces of the system. Moreover, there is always room for doubt and variance in judgement as a result of many persons involved in this cumbersome task, who might think differently while putting each statement into a particular category. Previously, requirements engineering required a lot of manual efforts since it involved changing user needs, specifications and documentation of minute system details. As extensive research is taking place in this area, more advanced tools and techniques are being developed in order to make the entire process of classifying requirements much simpler and less-intensive. Since the generation is at a point in the technology era where everything is beginning to get automatic, then why leave behind requirements classification?

Nowadays there are several commercial tools available for efficient requirements management, which automate many tasks such as document scanning, classification, storage, configuration, etc., but however these are not even remotely capable of processing free-form textual requirements documents which are likely to vary from one stakeholder to another.

Although not too widespread till date, the research work related to requirements classification is gaining momentum. Especially, with rapid advances in scientific areas like Machine Learning, Data Mining and Natural Language Processing the future of our goal looks promising. This study explores the usage of above techniques in all the research that has taken place till date. However, the goal of this study aims at incorporating Data and Text mining techniques in particular. Other techniques belonging to Machine Learning, NLP, and so on without a doubt play an integral role in automating the entire classification process. The study also primarily focuses on Enterprise Systems belonging to different industries out there for applying these techniques. Enterprise Systems basically include software belonging to different sectors of industry that support large-scale business processes and organizational workflows; for instance- enterprise systems pertaining to healthcare, banking, consulting, education, so on and so forth. As mentioned earlier, the study is concentrated upon Functional Requirements only. This study selects the science Text Mining to be the predominant technology associated with developing the framework. The availability of different types of documents during the software life-cycle fosters this idea of applying Text Mining (TM) techniques in order to perform an intelligent analysis of the written text with the goal of automating or assisting classification. Text mining involves a set of techniques to organize, classify and extract relevant information from text collections. These practices are part of a much general process of Knowledge Discovery in Databases (KDD), which is the semi-automated process of extracting relevant knowledge from databases (that may be textual), aiming to discover valid knowledge, previously unknown and potentially useful [22].

Chapter 2 presents the background and related work in the area of requirements classification using text mining methods; the research done thus far in automating the process of classification of software requirements. It mainly discusses the different techniques, tools and algorithms employed to classify textual data. I specifically discuss the inner working of the framework and the experiments conducted, along with the algorithm in detail. I would like to point out here that most of the research till now is focused on non-functional requirements, potentially because NFRs deal with intangible features of a software system like quality, security, etc. It becomes easier and faster to mine standard terms associated with these attributes, consequently producing accurate results. On the other hand, it is a bit more difficult to process and classify functional requirements since the manner in which they are written may differ from system to another. This chapter is ultimately concluded with the Problem Statement for this thesis.

Problem Statement

Using data analysis techniques and text classification algorithms to classify functional software requirements of enterprise applications into their respective categories, in order to make the entire process smooth and less laborious.

CHAPTER 2

BACKGROUND LITERATURE

The study ideally requires to start with a groundwork for the classification model to be based upon. This chapter discusses in depth the research conducted thus far in this area.

Groundwork and Requirements Classification Taxonomy

A classification taxonomy was proposed by Ghazarian [2] in his work. The paper studied characterization of different types of requirements into specific categories for Enterprise Systems. It lead to a finding that on an average, 85% of the functionalities in such Enterprise-based systems in a particular domain can be specified using a set of 5 types of requirements categories, although a wider pool of 9 classes were established for depicting requirements of Enterprise Systems. From this, Ghazarian [2] uncovered a taxonomical law, which stated that the emergence of classes in a requirements taxonomical scheme for a particular domain, independent of the order in which specifications of requirements in that domain are analyzed, includes a rapid initial growth phase, where majority of requirements are classified, followed by a rapid slowdown phase with periods of no growth, which is the stabilization phase. The research also lead to the finding that the functional requirements space in a particular domain of business can be effectively characterized by its requirement types and frequency distributions. This extensive work can make the entire process of classification very efficient, in a way that it becomes less time-intensive and labor-intensive. The research addressed some relevant questions appearing in the classification process, such as which are these types of classes that are generic for different enterprise domains? What are the frequency distributions of the same? That is which types occur the most. Can we infer from the frequency distributions the industry trends in requirements? Are there any phenomena related to the functional requirements space that are consistently observed in systems belonging to the same domain or across multiple domains? The experiment was furthered taking into account some 15 systems belonging to different domains and studying their requirements statements after decomposing them into atomic ones. A new approach was followed for classification, and it included data extraction and preparation – where in functional requirements were studied and analyzed, and later decomposed into atomic requirements statements, ready to be classified. Second step involved taxonomy development and classification, where in all statements were studied and clustered into different standard categories for establishing a new taxonomy. The taxonomy that emerged from this study resulted into following classes of functional requirements:

i. Data Input: description of the data items that are to be inputted into the software system.

ii. Data Output: the intermediate or final results of the system operations outputted to a device, including the contents of the outputs and the rules for displaying those contents.

iii. Data Persistence: descriptions of all the database related operations including reading, updating, inserting and deleting from/to a database.

iv. Data Validation: description of the validation rules required to ensure the correctness of the inputted data items in terms of the permissible domain of values, the value ranges, and their correct formats.

v. Business Logic: description of the application or business rules including workflows and calculations that define and govern the operations in a particular application area.

vi. Communication: description of the rules and the contents for electronic communication, such as email communication, between a system and an outside party.

vii. Event Trigger: description of the stimulating actions, such as clicking on a menu item, link, or button, that trigger system operations.

viii. User Interface Navigation: description of the flow of the screens (i.e. the rules for transition between screens) that make up an application.

ix. User Interface: description of the *static* layout of the pages and screens that make up a system's user interface.

x. External Call: description of the function calls between two systems including the description of the parameters used to make such calls and their expected values or responses.

xi. User Interface Logic: - description of the *dynamic* behavior of a system's User interface (i.e., how the user interface interacts with its users).

xii. External Call: description of the behavior of an operation or function in an external component or system.

During the case study process, the extracted atomic requirements were recorded in a Requirements Research Repository (RRR), whose relational database schema was specifically designed to support this study. It provided complete traceability from each target system to its use cases and from each use case to its corresponding atomic functional requirement. This is a classic case of efficient Requirements Management as discussed in previous section.

After frequency distributions for each of the above functional requirements classes were studied across different business systems and as a consequence, it was observed that only 5 of the 11 classes occurred most frequently in any system: Data Input, Data Output, Data Validation, Data Persistence, Business Logic and Event Trigger. From the graphs that were plotted, the law of taxonomic classification of functional requirements was also verified, by employing rigorous study and random ordering of specifications. Graphs of systems studied and number of classified requirements studied versus the requirements classes led to discovery of patterns which showed that the emergence of classes increases at first and after a certain slow-down phase, it remains constant and stabilizes. This study provides valuable insight into requirements classification and from the graph trends, decisions can be taken for classifying incoming requirements of other software systems, leading to ease of requirements management, improved efficiency, traceability and reduced costs. Therefore this study provides the necessary basis for the current study to build upon.

Automated Classification and their various approaches

Moving forward, there is another important piece of work by Slankas & Williams [8] on Automated Extraction of Non-Functional Requirements from Requirements documents using NLP and Data Mining techniques. Although the study focuses explicitly on NFRs as supposed to FRs, it provides significant insight into the automated processing in general. The goal of their research was to aid analysts in more effectively extracting relevant non-functional requirements in available unconstrained natural language documents through automated natural language processing [8]. The researchers used different techniques like K-nearest neighbors, Support Vector Machine and Naive Bayes classifiers to extract and classify NFRs from documents into 14 different categories and evaluated the performances of each. They addressed four important questions in the study as follows - 1) what document types contain NFRs in each of the 14 different categories? 2) What characteristics, such as keywords or entities (time period, percentages, etc.), do sentences assigned to each NFR category have in common? 3) What machine learning classification algorithm has the best performance to identify NFRs? 4) What sentence characteristics affect classifier performance? The research contributed to developing a tool for identifying NFRs into their respective categories and evaluating frequency of occurrence of each kind in the document.

As a part of their background study, Slankas & Williams [3] compared three machine learning techniques - K- nearest neighbors, Naive Bayes classifier and Support Vector Machine algorithm, to determine which would suit their purpose the best. They needed flexible, yet effective classification methods to handle different documents written in different ways with multiple ways of expressing similar concepts. Machine learning and Data Mining are very closely associated in a way that makes them complementary to each other. There are two types of categories in both fields for obtaining actionable information out of data - Supervised learning and Unsupervised learning. In supervised learning, people train classifiers with labeled data. People and systems then use these classifiers to decide in which class a previously unseen instance belongs. In contrast, unsupervised learning algorithms search data for common patterns (clusters). The data is not directly labeled, but rather groups of common instances are created [3]. The k- Nearest neighbors scheme is a kind of supervised learning, where the output is a class membership, determined by calculating ‘distance’ between the incoming data and the existing data. Based on majority vote of ‘neighbors’ around that data point, it is then classified to be in the cluster that it was closest to. Advantages of k-NN classifiers include ability to incrementally learn as new items are classified, to classify multiple types of data, and to handle large number of item attributes. The primary drawback to k-NN classifiers is that if they have n items stored, classification takes time [3]. Slankas & Williams evaluated other machine learning algorithms including naïve Bayes and Support Vector Machine (SVM). A naïve Bayes classifier works by selecting a class with the highest probability from a set of trained data sets given a specific document. It falls under the supervised learning category. Fundamentally, it assumes that each feature of a class exists independently of other features. Despite such an oversimplification, the approach performs effectively in real world problems. Naïve Bayes classifiers typically require fewer trained instances than other classifiers. SVM classifiers work by finding the optimal separator between two classes [8].

Performance measures

So when these three algorithms were evaluated, researchers found k-NN to be the most efficient in this particular case. The comparison results were based on three vital measures called Precision, Recall and F-measure. These three parameters and generally used everywhere in the domains of machine learning and data science in order to measure the performance and efficiency of a technique applied on the given set of data.

To compute these values, the classifier's predictions are divided into three categories for each classification value:

1) True positives (TP) are correctly predicted values.

2) False positives (FP) are predictions in which the sentence of another classification is classified as the one under evaluation.

3) False negatives (FN) are predictions in which a sentence of the same classification under evaluation is placed into another classification [8].

From these values, precision (P) is defined as the proportion of corrected predicted classifications against all predictions against the classification under test: P = TP/(TP + FP). Recall is defined as the proportion of classifications found for the current classification under test: R = TP/(TP+FN). F-measure is the harmonic mean of precision and recall, giving an equal weight to both elements: F = 2 x (P x R)/(P + R).

For the study by Slankas & Williams, as well as the current research topic in question Recall is a more important measure than the other two as it gives the number of statements correctly classified by the classification model. However, as per [8], Precision should not be ignored as it gives the accuracy of the classification model.

Works focused on requirements management

Cybulski and Reed [6] focused their work on the early phases of software development which include requirements engineering and management. They stressed on the importance of 'reuse' of software requirements in a common domain and consequently proposed a RARE (Reuse Assisted Requirements Elicitation) model making use of semi-automated tools like CASE. It uses IDIOM (Informal Document

Interpreter, Organizer and Manager) for processing an SRS document. The concept of reuse is directly linked to traceability of requirements - an essential detail for our current study which is domain-specific and intends to provide better performance for particular domains using requirements classified in preceding projects.

Researchers and students at Concordia University studied ways to employ text mining assistants in order to improve the quality of SRS [5]. Although it doesn’t hold direct relevance to the research topic in question, it sheds some light on important facets of Requirements engineering that can serve as a basis on which certain rules can be formulated in order to develop an efficient text mining framework for automated requirements classification.

The study investigated whether text mining tools could help reduce defects like ambiguity, inconsistency, omission, and redundancy [5]; addressing concerns about i) technical integration (how can text mining tools be introduced into RE tools?), (ii) adoption (can software engineers, who are typically not trained in NLP methods, easily use these tools?), and (iii) effectiveness (can the NLP tools indeed help to improve the quality of a specification?). Sateli, et al. developed ReqWiki, a collaborative, wiki-based platform customized for RE that allows (i) capturing of SRS content into several artifact templates, (ii) formally representing and reasoning over the populated SRS knowledge in an embedded ontology, (iii) applying specialized NLP services to all or parts of artifacts, and (iv) generating query-based, revision and domain-specific traceability links. It incorporated features like writing quality assessment, readability assessment, information extractor, requirements QA, and document indexer. Coming back to our topic, with refined quality of documents and lesser ‘noise’, the framework may perform its analysis more efficiently and give better results in terms of precision and recall factors.

A Natural Language Processing approach

Vlas, et al. [7] ventured into the domain of open-source projects to apply NL techniques to unstructured requirements documents belonging to Open Source projects in order to improve the quality of such software. They developed a system RCNL - Requirements Classifier for Natural Language. It used a pattern-based approach. A key element in the study was its multi-level ontology, in which the lower levels were grammar-based while the upper levels were requirements-based. The RCNL ontology implementation used a multi-level GATE parser. GATE is an open-source software for text processing. An ontology was created for RCNL by dividing types of text into six categories - from L0 to L5, each representing a class of natural language. First two contained common NL grammar concepts, next three contained concepts of logical statements and the final level contained classification statements. The GATE parser then implemented this ontology for classifying statements.

GATE provides JAPE (Java Annotation Pattern Engine), a rule-based text-engineering engine that supports Java and regular expressions. GATE also provides an annotation indexing and search engine with an advanced graphical user interface called ANNIC (Annotations in Context). RCNL makes use of both of these functionalities for rule-based matching of text with ontologies.

For all the 16 projects experimented upon by RCNL [7], they achieved 56% precision and 56% recall values. When these results were compared with those resulting from manual classification by experts, using a plugin provided by GATE, the results were promising - giving 94% precision and 64% recall. Vlas, et al. [7] reasoned that if parsing rules are enriched, quality of classification could be improved. This study also provides a structural and technical insight into developing an automated classifier model for functional requirements. It also imparts useful information about GATE and its functionality which could be used as a reference for developing a text mining classification framework.

An extensive approach to NFR classification

The research paper by Cleland-Huang, et al. [9] is an exhaustive study about the practical approaches to automating requirements classification. It goes into finer details about classifying each NFR into its appropriate category and discusses methods to accomplish the same. Although the techniques were targeted towards NFRs and the results achieved were not perfect and not completely automated as well, it is a significant step towards building an automatic classifier model for FRs and serves a good crux for the current topic at hand. It is also imperative to note that this study was used by many researchers as a solid reference to develop their own classification models, and compare their results with the same.

This paper first describes the classification algorithm and then evaluates its effectiveness through reporting a series of experiments based on 30 requirements specifications developed as term projects by MS students at DePaul University. A new and iterative approach is then introduced for training or retraining a classifier to detect and classify non-functional requirements (NFR) in datasets dissimilar to the initial training sets.

Although the classifier was trained to detect and classify general NFRs, this study emphasized on the necessity of training the model because of the disparity in requirements documents in terms of writing styles, domain-specific terminologies, company standards and so on. The study described experiments conducted to train the model in its initial phase and then went on to present an iterative approach for retraining the model which could be then used across different document types and domains. The classifier made use of information retrieval techniques to find distinct keywords called ‘indicator terms’ that most-closely related to each NFR type. The process was composed of three phases - training, classification and application.

In the training phase, the system was trained to identify requirements. This required a requirements engineer or analyst to manually classify several structured requirements specifications beforehand. These pre-classified documents were then fed into the classifier system, which mined the document for indicator terms pertaining to specific NFR types (security, performance, availability, extensibility, etc.). After identifying a set of indicator terms, each term was given a probabilistic weight for every NFR type. This weight was based on how strongly it correlated with a given NFR type. It was studied that some indicator terms might correlate to multiple NFR types, so every indicator term was given separate weights for each NFR type. In the classification phase, the system used the set of indicator terms to detect and classify NFRs from unclassified requirements specifications and other documents. One requirement, be it sentence or phrase, might have contained indicator terms related to different NFR types. The probabilistic weights for each indicator term related to one requirement determine which type it should be classified as. For each requirement, the weights of all corresponding NFR types were summed, and these sums were called the classification score. Requirements receiving classification scores above a certain threshold for a given NFR type were classified into that type. Any requirements that did not reach this threshold for any NFR type were assumed to be functional requirements, and were classified as such. This finding can be used as an important aspect in our topic, by mirroring the steps for FR instead of NFRs.

The result of this activity was a list of classified requirements. In the application phase, the classified requirements were used in subsequent software engineering activities such as requirements prioritization, architectural design, and so on. In their paper, the authors offered very little attention to the application phase. Their method was mainly concerned with automating the classification of requirements, and so is ours.

An ontology model

A similar and interesting ontology-based study conducted by Rashwan, et al. [10] was aimed at using a Support Vector Machine (SVM) based classifier for automated requirements classification. The researchers developed a whole new 'gold standard' corpus containing annotations for different NFR types. Although this is again a case where the focus was on NFRs, the study nevertheless is technically relevant for classifying FRs as well, which could essentially involve a different ontology based on their gold corpus.

Although there existed an ontology-based requirements elicitation approach for assisting analysts [8], making use of knowledge repositories that capture requirements categories from elicitation interviews, it is based on specific ontologies that vary over different domains. However, this SVM base classifier is a generic approach - independent of the context in which requirements are specified. The main goal of this study was to incorporate semantic analysis methods to automatically extract requirements written in natural language in order to make requirements machine-process able using ontological representation, then applying QA methods to analyze them for detecting defects like ambiguities and finally attempting to establish traceability links between NFRs and FRs for creating estimation models. For their analysis, the study made use of SRS documents belonging to the PROMISE corpus [9] which consists of 15 SRS documents, developed as term projects by MSc students at DePaul University. These specifications contain a total of 326 nonfunctional requirements and 358 functional requirements. One important observation in this study was that one requirement statement could be classified into more than one type of category. This point will play an important role in our research topic as well.

The requirements ontology was modeled using OWL (Web Ontology Language) by limiting the categories to those majorly occurring in the SRS. Initially, a manual annotation was done by analysts who analyzed each statement in the documents and classified requirements into their respective types, which could be more than one. After the corpus was defined with four main classes, each containing a set of categories and compared with manual annotation, an average of 78.36% was found which showed a high level of agreement between annotators. A gold standard was then established for each document based on group discussions. The NFR classifier was implemented using GATE by first preprocessing the documents and then training SRS data. It extracts features from documents and parses them using machine learning components. It uses ANNIE for finding tokens, splitting sentences and stemming relevant wordings. Rashwan, et al. used SVM algorithm with third-order polynomial kernel which gave best performance.

After all sentences had been classified, they populated the ontology with OWL using OwlExporter plugin of GATE and linking sentences to their source documents with corresponding classes. This enabled rich querying using SPARQL and OWL-DL reasoner tools for developing a knowledge base.

The model was evaluated using precision, recall and F-measure metrics. When the algorithm was applied to PROMISE corpus using Weka, the performance of SVM based classifier was compared with that of Cleland-Huang et al. [17]. Both approaches were roughly equal in Precision - 77% and 76% respectively, but this approach showed a significantly higher Recall - 60% as compared to 14%. In the final analysis phase, confusion matrices were created, that showed the false positives and false negatives of the classifier outcomes. The confusion matrix of the Functional Requirement (FR) Classifier shows that 76 sentences are classified as false positive, and 4 sentences as false negative. This meant that there is still a room for improvement in FR category and it could be achieved by increasing the amount of training data.

A semi-supervised approach

The study by Casamayor, et al. proposed a recommender system based on a semi-supervised learning approach for assisting analysts in the detection and classification of NFRs from textual requirements descriptions. The main goal was to provide suggestions to the analyst about candidate NFRs and their corresponding categories. That is, the classifier would automatically recognize a given requirement from the requirements document written in natural language and suggest suitable category. Later the results would be presented to an analyst for inspecting the results and who would give feedback for successive iterations.

The approach used here was a semi-supervised approach as opposed to supervised techniques like Bayesian methods, K-nearest neighbors, etc. with contain labeled training examples for analyzing the incoming unlabeled data. The semi-supervised approach makes use of learning from both labeled and unlabeled data. The Expectation-Maximization algorithm is one of such kind, as is used in this study for developing a recommender system. It basically consists of two steps, the Expectation step (E-step) and the Maximization step M-step). The E-step fills in missing data based on an estimation of parameters while the M-step re-estimates the parameters to maximize or increase the likelihood of those parameters. The parameters that EM estimates in this case are the probability of each word given a class and the class prior probabilities.

Initially, the documents in the labeled set L have class labels, whereas the documents in the unlabeled set U have missing class labels. EM algorithm is used to estimate these missing class labels based on current labels that is by assigning probabilities to class labels in each document belonging to U. This 2-step process was reiterated until the probability parameters became stable.

Nigam et al. [21] proposed the EM algorithm for LU learning with Naive Bayes classification, which was hence used in this study. For experimenting this approach, Casamayor, et al. used the PROMISE repository. The collection was split into 468 requirements as training set, and remaining 156 as the test set for applying EM. When compared with classical algorithms like Naive-Bayes and k-NN, and text mining algorithm like TF-IDF on same data, EM outperformed these in terms of accuracy, as unlabeled documents provided some good insights such as words which tend to appear together in a specific type of document. Even when the training data was increased, it gave similar accurate results. As a parting note on empirical analysis of this model, the researchers pointed out that performance could be improved if documents belonging to a single software project are considered during learning.

Since this was a semi-supervised approach as opposed to fully supervised techniques discussed before, the authors pointed out that the main drawback of applying supervised methods to requirements detection is the amount of pre-categorized requirements needed to reach good levels of precision in the classification process. However, the use of distinctive vocabulary, domain terminology and writing styles across different projects as well as requirement elicitation process could tend to hinder the application of the current method. A substantial concluding point was that semi-supervision involves less human effort in labeling the requirements, including manual revision and classification of textual requirements statements than fully supervised methods thereby saving time and labor resources, and most importantly taking a step towards automated classification of requirements.

Another variant of NLP approach

Hussain, et al. [12] focused on natural-language processing tools to extract, differentiate FRs and NFRs, and classify NFR statements from SRS documents. They proposed a text-classifier model equipped with a POS (part-of-speech) tagger which resulted into very high accuracy of 95.86% when applied to data same as that used by

Cleland-Huang, et al. [9]. They used the same corpus as before - PROMISE repository SRS documents [19], with the Stanford Parser for stemming grammatical words and extracting 5 syntactic features from each of the training sentences from the documents. These features were Number of Adjectives, Number of Adverbs, and Number of Adverbs that modify Verbs, Number of Cardinals, and Number of degree Adjectives/Adverb, and were identified as candidate terms that influence the process of classifying NFRs the most. To determine and automatically select which of these features were valid for detecting NFRs, a probabilistic ranking measure was used based on higher probabilities of occurrences in the training set. A cutoff threshold value of 0.8 was then selected manually and all features exceeding this value were selected as valid. Now the study by Cleland-Huang, et al. [9] identified specific keywords, but not in context of their parts-of-speech group. After analyzing types of most probable words used in NFR, as described in [12], they considered keywords of 9 different parts-of-speech groups separately. After repeating the ranking process on these groups using two different methods - Unsmoothed and Smoothed Probability Measures. To classify the sentences, they developed a feature-extraction program in Java that parses the sentences from the corpora, and extracts the values of all the features mentioned above. It used Weka to train the decision-tree algorithm. The results came out to be exceptionally well when using the whole dataset for training and testing. Since the dataset was not very large, they also used 10-fold cross validation, and the results were as satisfactory. On creation of its confusion matrix, the study reaped a precision of 97.8% and a recall of 100% with no false negatives. Even the standard deviation was pretty low during each iteration of cross-validation in the entire process.

When compared with the work of Cleland-Huang, et al. [9], this study showed significant improvement - by a large margin in terms of overall accuracy, precision and recall. In conclusion, this research is based on linguistic tools for classifying requirements. Although it does not make use of data mining or machine learning paradigms per se, it definitely gives an insight into the performance gain of such a system as compared to other techniques.

Hussain, et al. [16] developed a tool for annotation of software requirements, which basically extracted textual statements and classified them by assigning appropriate labels. They showed the importance of annotation of requirements in practical use by stating the need to build annotated sets of such documents that are used in number of recent projects that attempt to learn the workings of the human brain behind different requirements analysis tasks like classification and automate these tasks by using supervised or semi- supervised learning techniques. This could come in handy for our approach which is targeting domain-specific software projects that can make use of annotated corpora like these to train the model. The tool - LASR (Live Annotation of Software Requirements) is a client-server based web application with a rich UI built upon the CakePHP framework and makes use of lightweight NLP tools like sentence delimiter and a noun-phrase chunker, that can automatically extract requirements instances at the levels of passages, sentences and noun-phrases from the requirements documents and save them to the backend. The results were compared with statements manually annotated by expert students as well as students having no prior experience or background in annotating statements, and were found pretty reliable. LASR attempts to compute the annotation for each instances, by first assigning a custom score to each of the annotation labels based on the level of confidence submitted by the annotators. Thus, the annotation label with the highest score, and that is also greater than some threshold, was selected as the gold-standard annotation. They aim to reduce manual efforts in their future work.

Other related works

Raamesh and Uma [4] attempted to automatically generate through their study, test cases from SRS documents using data mining techniques for facilitating automated or better manual software testing. They used Weka data mining tool for classifying functional and nonfunctional requirements. The study followed a rather circuitous approach wherein the model first created UML state diagrams out of textual sentences and then proceeded to apply data mining tools to classify requirements statements. It started by generating classification rules, selecting a predicate on state diagram transition, which was then transformed into a predicate function. Finally mining techniques like association mining and clustering are applied to determine categories of FRs and NFRs. As stated earlier, it turned out to be complicated manner of classifying requirements out of use-cases, and using unsupervised methods are not recommended for the current study in question. Supervised text classification algorithms would give better results in context of this study.

Ko et al. (2000) [13] proposed a Web-based analysis-supporting system. For automatically classifying the collected informal requirements into several views or topics, the system required as input a set of words representing the viewpoint of each analyst. They used similarity matching techniques, frequency of words and part-of -speech tagging to expand the initial data set and find co-occurrence of words. The main shortcoming of this method was its reliance on analysts for extracting keywords for different views of the software to be developed. Again, this was not a fully automated method and such studies shed more light on the need for automatic classification system that requires minimum manual effort. In 2007, they went ahead to develop a bootstrapping framework for classifying requirements based upon their previous study [23]. Each statement was separated, and keywords were extracted for each topic. A centroid-sentence was derived for each topic that is the sentence that defines the core meaning of that topic. It was also a topic-keyword statement which included topic keyword or any word or a requirement statement as a whole, possessing high similarity to the topic category obtained using a similarity-matching technique. A Naive-Bayes classifier was then applied to train this topical data. This system does reduce manual effort and analysts can easily detect the structure of collected requirements by inputting topic words only. However the same drawback applies to this as of the previous work [13].

Palmer and Liang [20] proposed a Two-tiered clustering algorithm for indexing and clustering requirement specifications by functionality. This study is an example of utilization of unsupervised methods as opposed to supervised technique of classification. Hussain, et al. [17] further ventured into developing an annotated text classification system for assessing quality of requirements statements and detecting ambiguities automatically, again using a similar technique as described in the previous section. Park et al. [21] proposed a requirement support system that evaluated the resemblance of requirement sentences using similarity measures by using Information Retrieval methods to identify possible redundancies and inconsistencies as well as to detect potentially ambiguous requirements.

These studies can aid in smoothing of input data to the framework visualized in the current topic. As obvious it can be, such enriched data with no defects can enable the model to work efficiently, giving high precision and recall.

CHAPTER 3

METHODOLOGY

This chapter discusses in depth the methods and techniques as well as the data and tools that were used to conduct the study of text classification and categorization of software requirements. Figure 1 below shows a block diagram representation of methodology employed in this study.

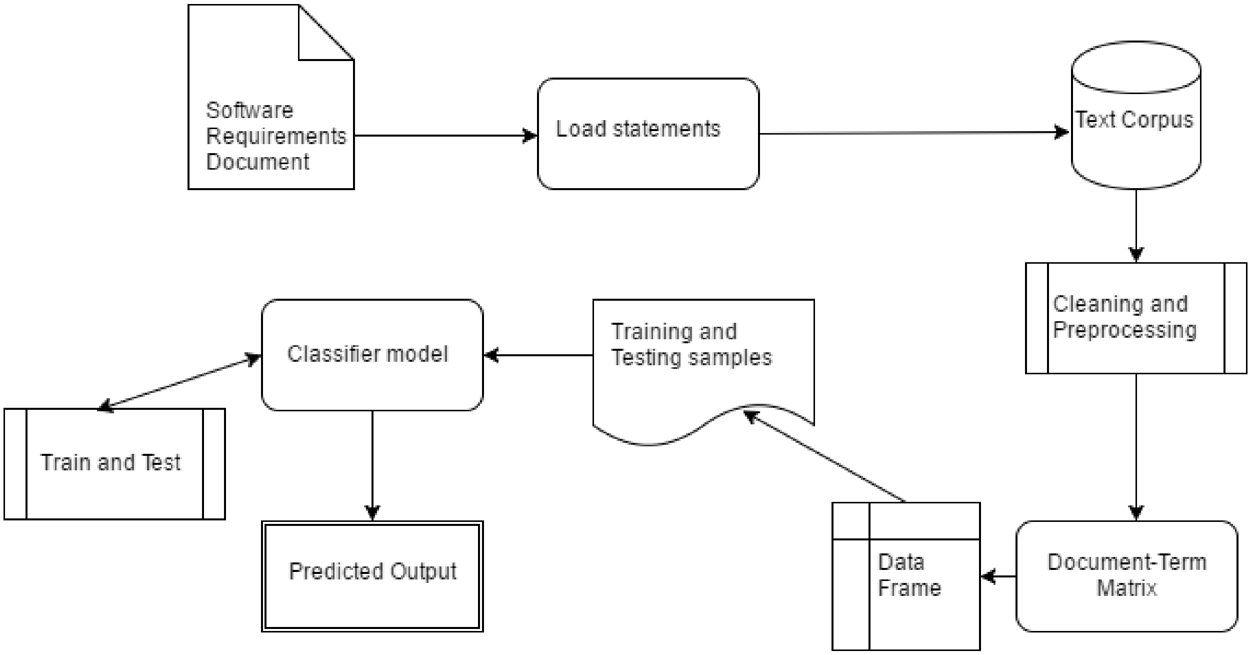


Figure 1. Classification process methodology block-diagram

Data

For mining and extracting actionable information, data is everything. It is highly influential for the results produced and analysis of the same. As the study deals with requirements data pertaining to different enterprise systems, data is variable for achieved results. For this project, I used the requirements dataset belonging to domain of mathematical software systems. It consisted of 1495 requirements statements that were already classified into respective categories as a part of a prior research in requirements engineering. However, the size of data set was refined and reduced to 672 statements for applying kNN classifier by further distilling it to carefully include non-redundant statements, sufficient to be trained upon. The collection of statements was derived from the following mathematical systems - Geogebra 4, Graph, wxMaxima, MS Excel 2007, Mathcad 14.0, Mathematica 8.0, Stata 12, Minitab 15, CPMP Tools and Maple. As these requirements come from verified software systems, it is safe to assume that they are authentic. This dataset, in form of a comma separated (csv) file served as an input for the classification model, and was sampled into two parts for training and testing of the framework during the course of text classification process. Throughout the thesis, we sometimes refer to these statements as documents, as it conforms to the terminologies used in Text mining. There are three columns in the set - ID, Requirement.Statement and Requirement.Type. The first is simply a numerical ordering of statements while the second column actually holds all atomic sentences of requirements; third column being the type or category of corresponding statement in column 2, which is discussed next.

Categories

In the dataset, the column Requirement.Type contains the 12 categories in which the textual statements are and will be classified into. As discussed in Chapter 2 - Background study, the different categories of classification determined by Dr. Ghazarian’s research on requirements taxonomy [2] were used in this study. The description of each of the 12 categories is briefly described below, in context of mathematical enterprise software.

1. Data Input: Description of data items that serve as input to the software system.

2. Data Output: The intermediate or final results of the system operations output to a device, including the contents of the outputs and the rules for displaying those contents.
3. Data Persistence: Descriptions of all database related operations including reading, updating, inserting and deleting from/to a database.
4. Data Validation: Description of the validation rules required to ensure the correctness of the inputted data items in terms of the permissible domain of values, the value ranges, and their correct formats.
5. Event Trigger: Description of the stimulating actions, such as clicking on a menu item, link, or button, that trigger system operations.
6. User Interface Navigation: Description of flow of the screens (i.e. the rules for transition between screens) that make up an application.
7. User Interface: Description of the static layout of the pages and screens that make up a system’s user interface.
8. User Interface Logic: Description of the dynamic behavior of a system’s User interface (i.e., how the user interface its users).
9. Application Logic: Description of the application or business rules including workflows and calculations that define and govern the operations in a particular application area.
10. External Call: Description of function calls between two systems and the expected value of the calls, such as ‘system uses OS’s file browser for navigation’.
11. Mathematical Algorithm/Procedure: Description of a specific function performed mathematically, such as 'system then calculates area of triangle'.

12. Mathematical Rule/Principle: Description of a specific rule, such as 'area cannot be negative'.

Algorithms and Classifiers

Text categorization is the process of grouping text documents into one or more predefined categories based on their content. In supervised learning, a number of statistical classification and machine learning techniques have been applied to text categorization, including regression models, Bayesian classifiers, decision trees, nearest neighbor classifiers, neural networks, and support vector machines [24]. This section describes them in context of the model developed.

In its first version, the model used the k-Nearest Neighbors (kNN) classification algorithm. To classify a class-unknown document X, the k-Nearest Neighbor classifier algorithm ranks the document's neighbors among the training document vectors, and uses the class labels of the ‘k’ most similar neighbors to predict the class of the new document. The classes of these neighbors are weighted using the similarity of each neighbor to X, where similarity is measured by Euclidean distance or the cosine value between two document vectors. The kNN classifier is based on the assumption that the classification of an instance is most similar to the classification of other instances that are nearby in the vector space. The knn function in R applies the algorithm using the Euclidean distance, which is a distance measure is based on the Pythagorean formula in a 2-dimensional vector space. Compared to other text categorization methods such as Bayesian classifier, kNN does not rely on prior probabilities, and it is computationally efficient. The main computation is the sorting of training documents in order to find the k nearest neighbors for the test document.

In its second version, the model used several other algorithms, all at once. This approach is referred to as ensemble learning and is a popular technique in machine learning. Ensemble methods basically aim to obtain better predictive performance than could be obtained from any of the constituent learning algorithms. I have used 8 algorithms in my ensemble - Support Vector Machine (SVM), Generalized Linear Model Network (GLMNET), Maximum Entropy (MAXENT), Boosting, Bagging, Random Forests, Neural Network and Decision Trees.

A Support Vector Machine (SVM) is a supervised, discriminative classifier defined by a separating hyperplane, and given labeled training data the algorithm outputs an optimal hyperplane for classifying incoming (test) data. The GLMNET is an extremely fast algorithm that uses fitted generalized linear models via maximum likelihood techniques for classifying sparse data. The Maximum Entropy classifier is a probabilistic classifier which belongs to the class of exponential models. Unlike the Naive Bayes classifier, the Max Entropy does not assume that the features are conditionally independent of each other. The MaxEnt is based on the Principle of Maximum Entropy and from all the models that fit the training data, selects the one which has the largest entropy. Bagging and boosting are meta-algorithms that pool decisions from multiple classifiers. Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models misclassified.

Bootstrap aggregating, often abbreviated as bagging, and involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set [26]. Random forests is an extremely accurate classifier that works by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of individual trees. A neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand [27]. The decision tree classifier organizes the training set into a series of test questions and conditions in form of a tree structure and makes decisions based on answers traversed.

Language and development environment

For developing and studying the classification model, I used **R** - a language and environment for statistical computing and data analysis [28]. It is available on the Comprehensive R Archive Network (CRAN). I used the RStudio IDE for writing and executing code. The version I employed is R 3.2.3 Wooden Christmas Tree. RStudio provides a set of integrated tools to make most out of the language R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management [29].

Packages and tools

One of the packages used here was the Text Mining (tm) package, a framework in R built specially for text mining applications. It is feature-rich in a way that incorporates text pre-processing functions for loading, cleaning and exploring text data. Activities like corpus creation, text preprocessing and document term matrix creation were performed using tm package.

Another very useful text mining package called RTextTools was also used in this study to exploit its functionalities in comparison to tm. This package specifically serves text classification for machine learning in R. It allows use of combination of algorithms for classification and includes nine algorithms for ensemble classification (svm, slda, boosting, bagging, random forests, glmnet, decision trees, neural networks, maximum entropy), 8 of which I applied to the current model.

Other packages used for singular functionalities in this project were class - for generating kNN classifier; SnowballC - for word stemming and ggplot2 - for data visualization and plotting results.

Data Preparation and Preprocessing

At the start, data was loaded from the csv file into the R script in form of a data frame, which is an efficient matrix-like data structure in R for holding any-dimensional data. Next, the libraries discussed in previous section were loaded to perform text cleaning and preprocessing. A corpus, which is a structured collection for managing documents in text mining, was then created for holding the text from Requirements.Statement column of the data frame. This corpus was then cleaned up using tm_map function of tm package in order to make it easier for the model to train upon it over for classification. Cleaning the corpus involved removing whitespaces, numbers, punctuations and stop words (unimportant words like is, the, for, etc.); transforming text to lowercase and stemming words - converting them all into their root words. These text terms would be used for training the model. Next task was to put these training terms into a Document Term Matrix, a matrix containing frequencies of occurrence of all training terms from the corpus. This enables training model in way that helps it decide classification criteria based on frequency of each term in the training text. This DTM was made sparse with a factor of 0.97, that is, with lowest frequency terms eliminated the matrix then contained terms that were present in 97% of the documents. After this step, the DTM was loaded into a data frame and the Requirements.Type column was bound to it for training the data over already classified documents. The data in data frame was then partitioned into two samples - training data and test data. The training set contains 765 statements while the test set contains 735 statements. The classifier column renamed as 'Category' was isolated into a vector as required by the kNN function for training.

As interpretation of data analytics is rather a cumbersome task for those not experienced it, data visualization practice helps depict raw data or results in form of graphs, charts and diagrams. For visualizing data graphically in terms of frequency and percentage distribution, I used the ggplot package to create histograms for training and test data sets. The following figure represent a pictorial view of requirements in the data set and their prevalence or frequency of occurrence in the system.

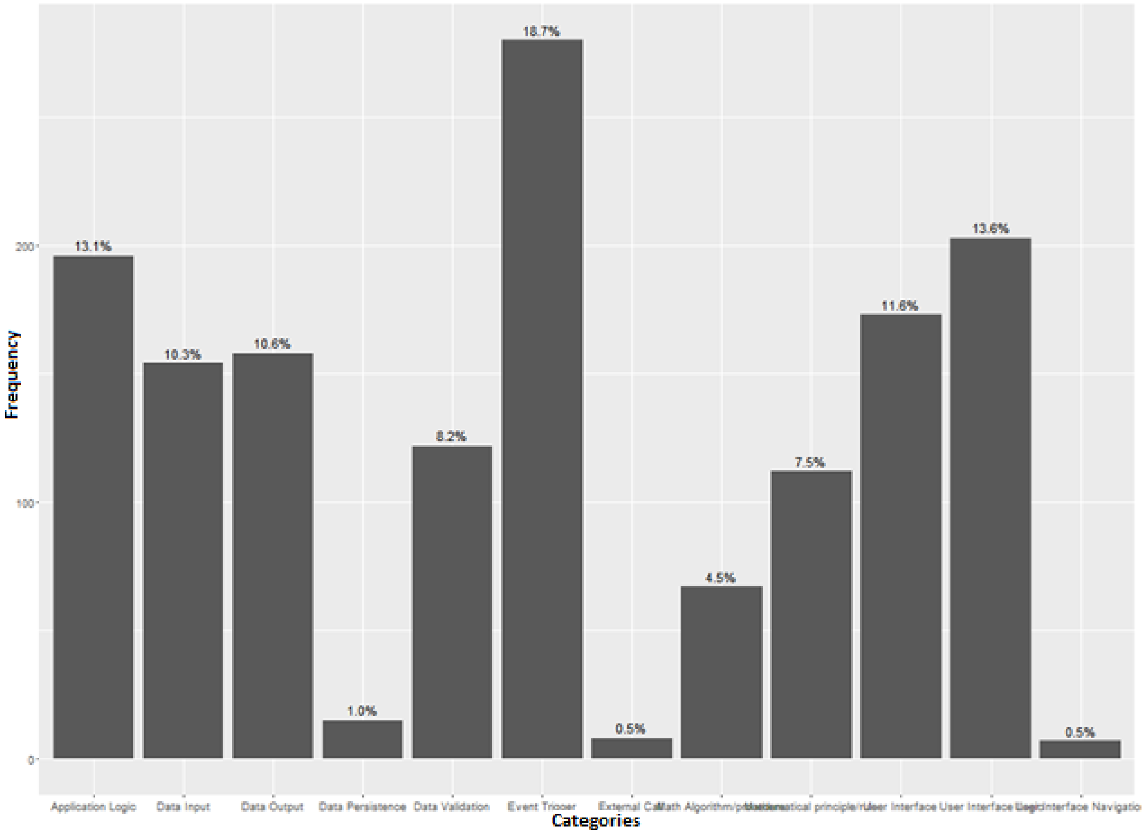
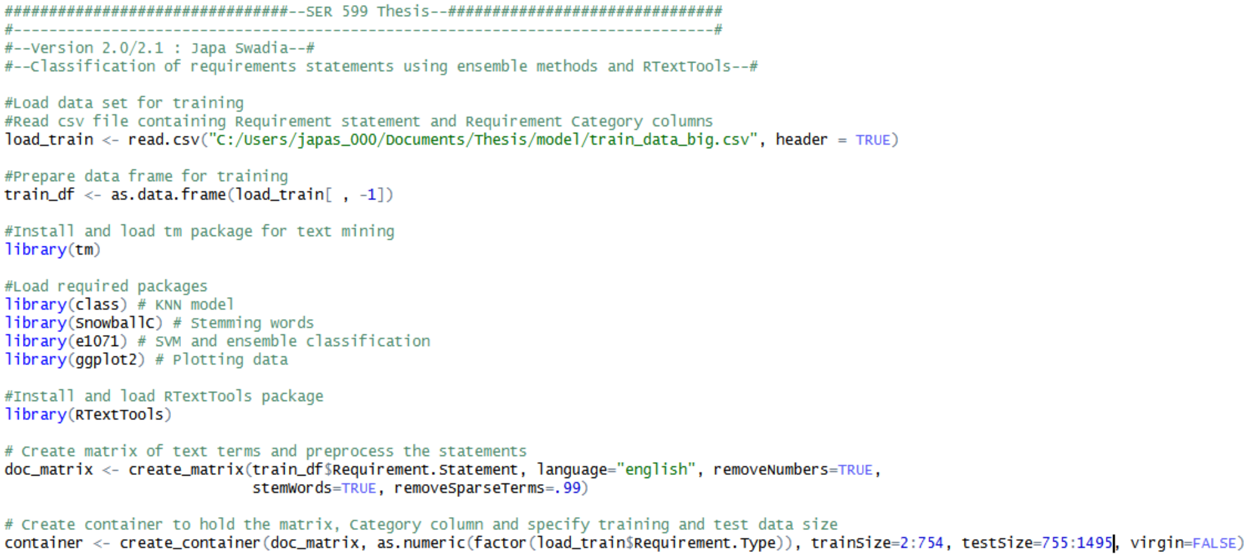


Figure 2. Number and percentage distribution of requirements over categories

Training and testing

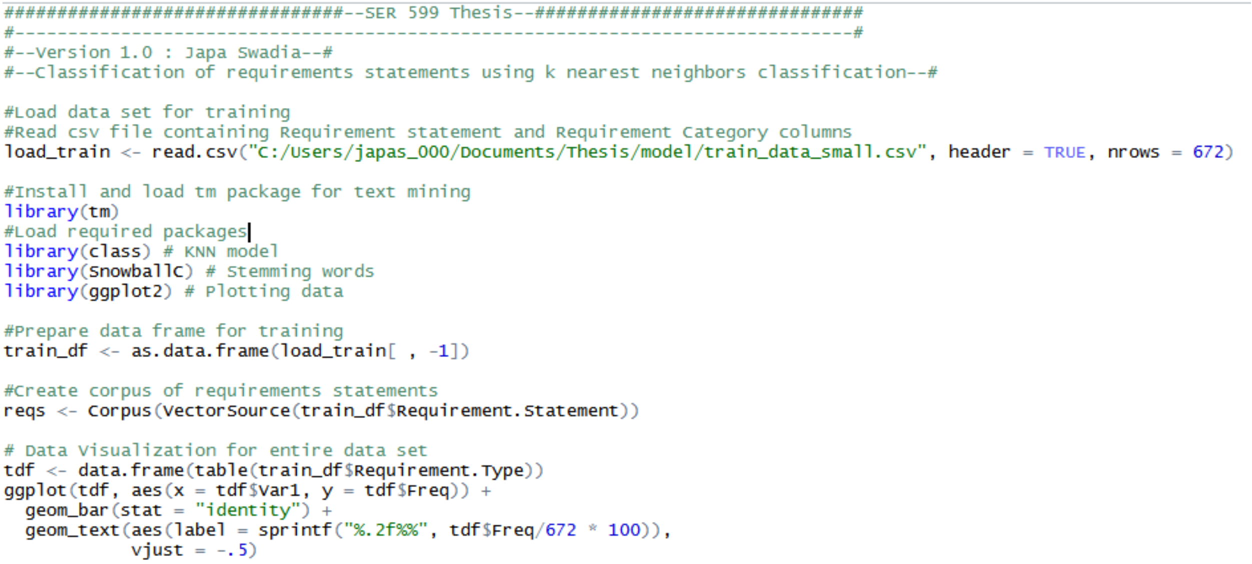
Once the data was prepared for training, I conducted two versions or approaches of classifying text as discussed previously. I first applied the kNN classification technique using its function knn which takes training sample, test sample and the classifier as its arguments. The output of this classifier was then bound to the category column and stored in a data frame; and subsequently written to a csv file. In its second version I applied ensemble techniques by first training them each over the training set using train_model functions, and then using the classify_model function to test the model over the test set. I also used 4-fold cross-validation to check and try to improve accuracy of results. Figures 3 and 5 represent the source code from preprocessing till training stages in ensemble approach while figures 4 and 6 represent the same for kNN approach.

The results and discussion on analysis parameters is carried forward in the next chapter.



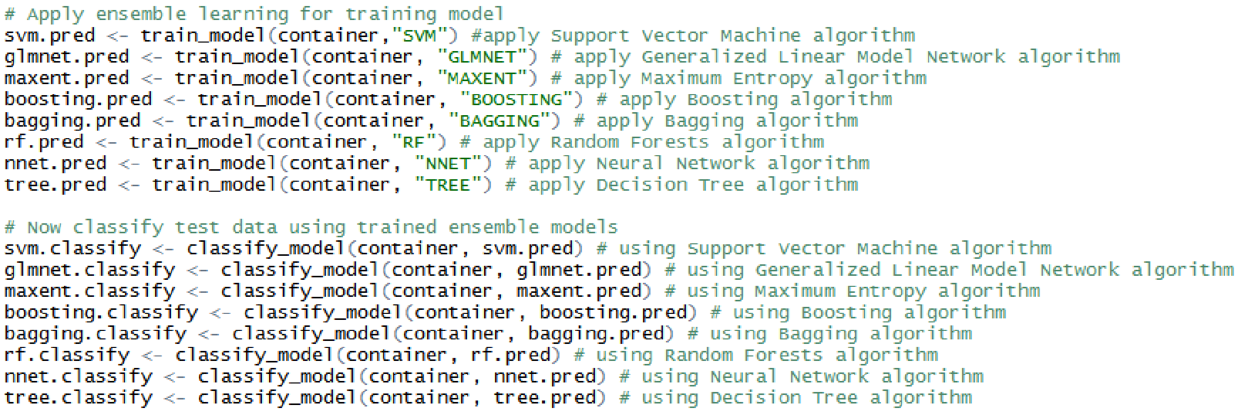
```
################################--SER 599 Thesis--###############################
#-------------------------------------------------------------------------------#
#--Version 2.0/2.1 : Japa Swadia--#
#--Classification of requirements statements using ensemble methods and RTextTools--#

#Load data set for training
#Read csv file containing Requirement statement and Requirement Category columns
load_train <- read.csv("C:/Users/japas_000/Documents/Thesis/model/train_data_big.csv", header = TRUE)

#Prepare data frame for training
train_df <- as.data.frame(load_train[ , -1])

#Install and load tm package for text mining
library(tm)

#Load required packages
library(class) # KNN model
library(SnowballC) # Stemming words
library(e1071) # SVM and ensemble classification
library(ggplot2) # Plotting data

#Install and load RTextTools package
library(RTextTools)

# Create matrix of text terms and preprocess the statements
doc_matrix <- create_matrix(train_df$Requirement.Statement, language="english", removeNumbers=TRUE,
                            stemWords=TRUE, removeSparseTerms=.99)

# Create container to hold the matrix, Category column and specify training and test data size
container <- create_container(doc_matrix, as.numeric(factor(load_train$Requirement.Type)), trainSize=2:754, testSize=755:1495, virgin=FALSE)
```

Figure 3. Loading and preprocessing data in ensemble approach.



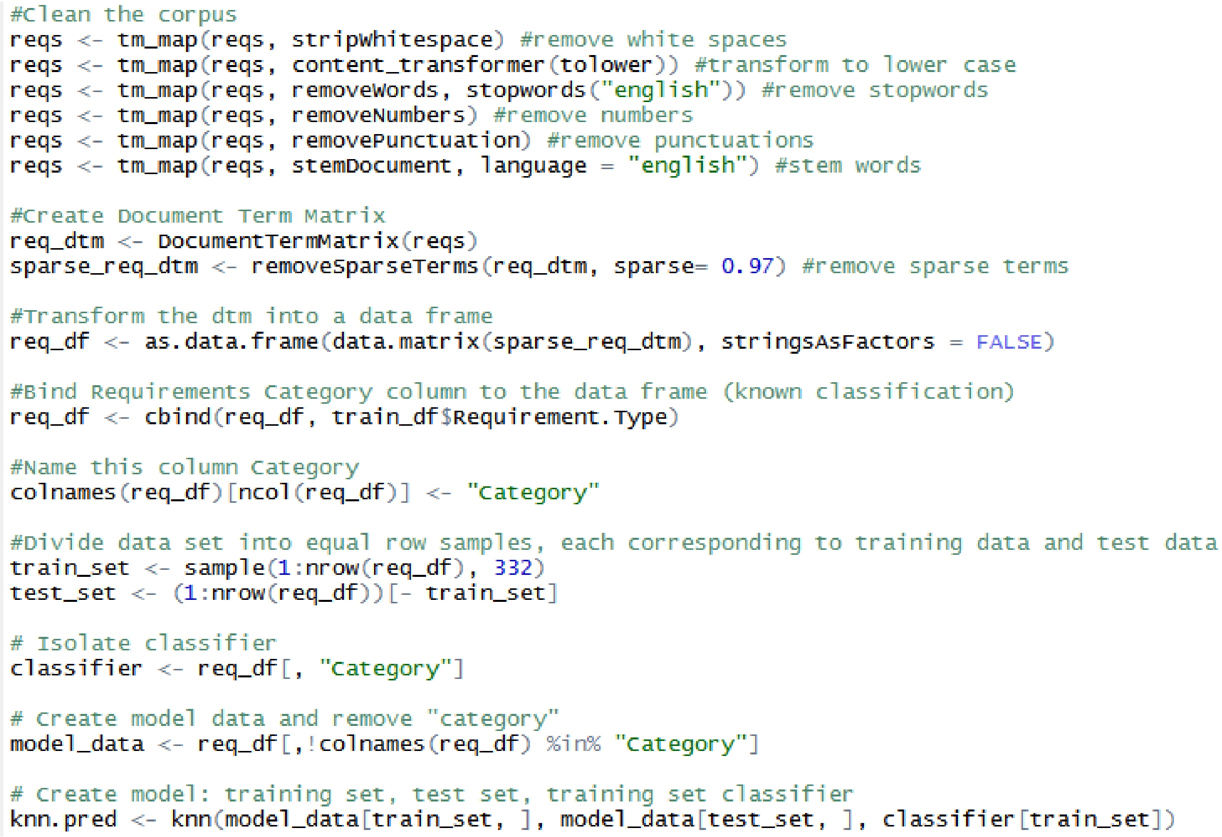
```
################################--SER 599 Thesis--###############################
#-------------------------------------------------------------------------------#
#--Version 1.0 : Japa Swadia--#
#--Classification of requirements statements using k nearest neighbors classification--#

#Load data set for training
#Read csv file containing Requirement statement and Requirement Category columns
load_train <- read.csv("C:/Users/japas_000/Documents/Thesis/model/train_data_small.csv", header = TRUE, nrows = 672)

#Install and load tm package for text mining
library(tm)
#Load required packages
library(class) # KNN model
library(SnowballC) # Stemming words
library(ggplot2) # Plotting data

#Prepare data frame for training
train_df <- as.data.frame(load_train[ , -1])

#Create corpus of requirements statements
reqs <- Corpus(VectorSource(train_df$Requirement.Statement))

# Data Visualization for entire data set
tdf <- data.frame(table(train_df$Requirement.Type))
ggplot(tdf, aes(x = tdf$Var1, y = tdf$Freq)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = sprintf("%.2f%%", tdf$Freq/672 * 100)),
            vjust = -.5)
```

Figure 4. Loading and preprocessing data in kNN approach.



```
# Apply ensemble learning for training model
svm.pred <- train_model(container,"SVM") #apply Support Vector Machine algorithm
glmnet.pred <- train_model(container, "GLMNET") # apply Generalized Linear Model Network algorithm
maxent.pred <- train_model(container, "MAXENT") # apply Maximum Entropy algorithm
boosting.pred <- train_model(container, "BOOSTING") # apply Boosting algorithm
bagging.pred <- train_model(container, "BAGGING") # apply Bagging algorithm
rf.pred <- train_model(container, "RF") # apply Random Forests algorithm
nnet.pred <- train_model(container, "NNET") # apply Neural Network algorithm
tree.pred <- train_model(container, "TREE") # apply Decision Tree algorithm

# Now classify test data using trained ensemble models
svm.classify <- classify_model(container, svm.pred) # using Support Vector Machine algorithm
glmnet.classify <- classify_model(container, glmnet.pred) # using Generalized Linear Model Network algorithm
maxent.classify <- classify_model(container, maxent.pred) # using Maximum Entropy algorithm
boosting.classify <- classify_model(container, boosting.pred) # using Boosting algorithm
bagging.classify <- classify_model(container, bagging.pred) # using Bagging algorithm
rf.classify <- classify_model(container, rf.pred) # using Random Forests algorithm
nnet.classify <- classify_model(container, nnet.pred) # using Neural Network algorithm
tree.classify <- classify_model(container, tree.pred) # using Decision Tree algorithm
```

Figure 5. Training in ensemble approach



```
#Clean the corpus
reqs <- tm_map(reqs, stripWhitespace) #remove white spaces
reqs <- tm_map(reqs, content_transformer(tolower)) #transform to lower case
reqs <- tm_map(reqs, removeWords, stopwords("english")) #remove stopwords
reqs <- tm_map(reqs, removeNumbers) #remove numbers
reqs <- tm_map(reqs, removePunctuation) #remove punctuations
reqs <- tm_map(reqs, stemDocument, language = "english") #stem words

#Create Document Term Matrix
req_dtm <- DocumentTermMatrix(reqs)
sparse_req_dtm <- removeSparseTerms(req_dtm, sparse= 0.97) #remove sparse terms

#Transform the dtm into a data frame
req_df <- as.data.frame(data.matrix(sparse_req_dtm), stringsAsFactors = FALSE)

#Bind Requirements Category column to the data frame (known classification)
req_df <- cbind(req_df, train_df$Requirement.Type)

#Name this column Category
colnames(req_df)[ncol(req_df)] <- "Category"

#Divide data set into equal row samples, each corresponding to training data and test data
train_set <- sample(1:nrow(req_df), 332)
test_set <- (1:nrow(req_df))[- train_set]

# Isolate classifier
classifier <- req_df[, "Category"]

# Create model data and remove "category"
model_data <- req_df[,!colnames(req_df) %in% "Category"]

# Create model: training set, test set, training set classifier
knn.pred <- knn(model_data[train_set, ], model_data[test_set, ], classifier[train_set])
```

Figure 6. Preprocessing data and training in kNN approach

CHAPTER 4

DATA ANALYSIS AND RESULTS

This chapter discusses the results of analytics obtained after applying techniques for classification of requirements as specified in the preceding chapter. It explains the analysis parameters and the effect of each algorithm used in classification.

Analysis of kNN classifier

After applying the kNN function for classification, a confusion matrix was created to study and obtain analysis results. A confusion matrix, or contingency table, is a matrix representation of an algorithm's accuracy of classification and is formed by drawing up the actual or expected outcomes against the classified or predicted outcomes, as rows and columns. By looking at this matrix and examining it, one can discern whether or not a document or statement was classified correctly or not, and if it was classified at all then as what it was classifies as. From this matrix I calculated values like True Positives, False Positives, True Negatives and False Negatives; followed by performance measures like Precision, Recall, F-measure and Accuracy. These parameters have been discussed in detail in Chapter 2. Precision answers the following question - 'Out of all the statements the classifier labeled as belonging to a particular category (say Data Input), what fraction were correct?' On the other hand recall answers the question - 'Out of all the (say Data Input) requirements statements that were there, what fraction did the classifier pick up?

The answers to these questions have been summarized into Table 1 below. It can be observed that the Euclidean similarity measure obtained using 1 nearest neighbor (knn1) function gave better results than the knn function which assumes value of k automatically. Precision values of 61% and 86% were achieved respectively, while recall values of 77% and 76% were achieved respectively for knn and knn1 classifier functions. The overall accuracy of the model came up to 52% from 50%. This result shows that precision greatly improved when the number of nearest neighbors was fixed at 1. A larger value of k produces a smoother boundary for classes and reduces noise.

	kNN (k=auto)	**kNN (k=5)**	**kNN1 (k=1)**
Precision (%)	**61**	**95**	**86**
Recall (%)	**77**	**80**	**76**
F-measure (%)	**68**	**87**	**78**
Accuracy (%)	**50**	**52**	**52**

Table 1. Performance measures of kNN classifier methods.

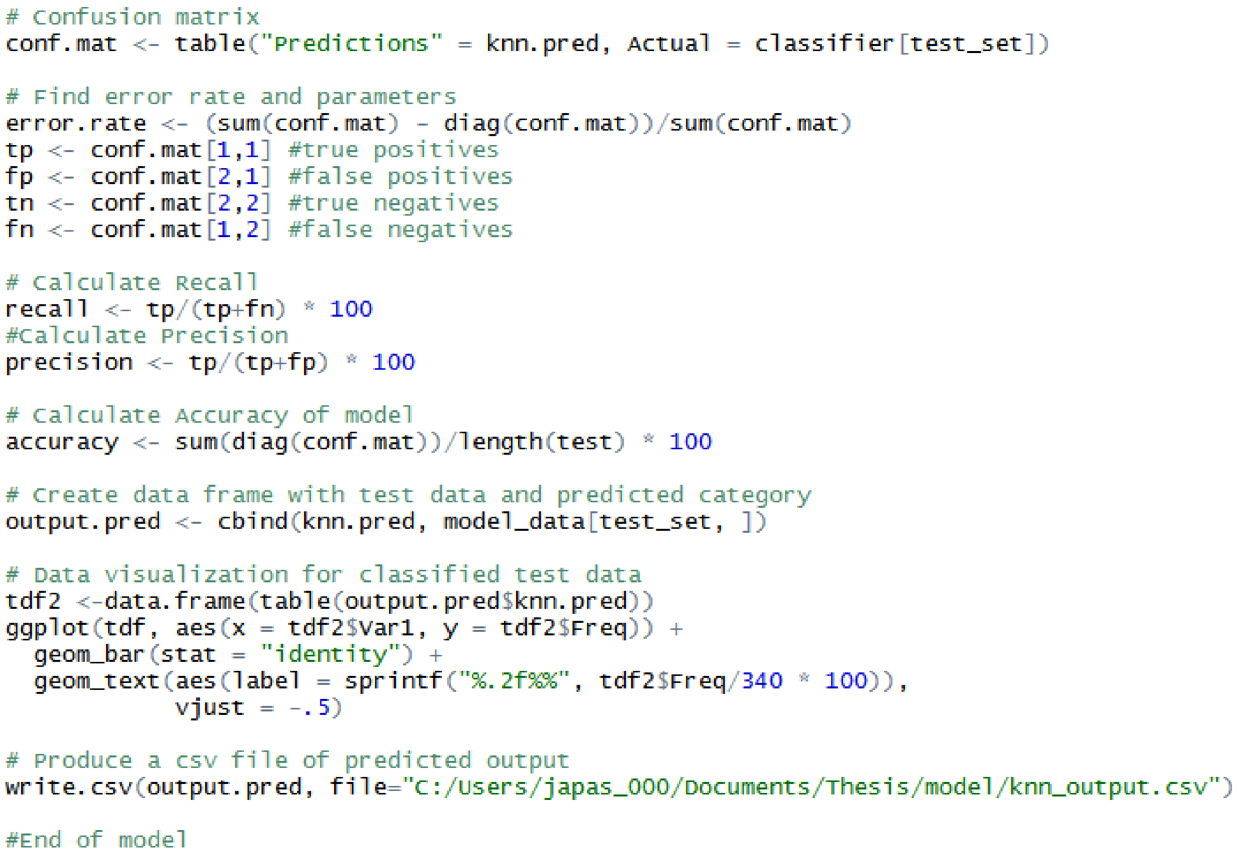
Using a useful table function in R, we could summarize the original test data and compare it with the kNN-classified data. The table below shows frequency of statements in original test data and classified test data. This also provides an overview of how the model performed. However, this does not give an idea about whether these were correctly classified.

Category	**Frequency in original test data**	**Frequency in kNN-classified test data**
Application Logic	32	35
Data Input	71	50
Data Output	18	23
Data Persistence	15	13
Data Validation	22	23
Event Trigger	56	64
External Call	3	7

Math Algorithm/procedure	37	31
Mathematical principle/rule	16	41
User Interface	40	25
User Interface Logic	27	25
User Interface Navigation	3	3

Table 2. Frequencies of requirements statements

Figure 7 below depicts the source code for determining the performance measures discussed above using a confusion matrix.



```
# Confusion matrix
conf.mat <- table("Predictions" = knn.pred, Actual = classifier[test_set])

# Find error rate and parameters
error.rate <- (sum(conf.mat) - diag(conf.mat))/sum(conf.mat)
tp <- conf.mat[1,1] #true positives
fp <- conf.mat[2,1] #false positives
tn <- conf.mat[2,2] #true negatives
fn <- conf.mat[1,2] #false negatives

# Calculate Recall
recall <- tp/(tp+fn) * 100
#Calculate Precision
precision <- tp/(tp+fp) * 100

# Calculate Accuracy of model
accuracy <- sum(diag(conf.mat))/length(test) * 100

# Create data frame with test data and predicted category
output.pred <- cbind(knn.pred, model_data[test_set, ])

# Data visualization for classified test data
tdf2 <-data.frame(table(output.pred$knn.pred))
ggplot(tdf, aes(x = tdf2$Var1, y = tdf2$Freq)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = sprintf("%.2f%%", tdf2$Freq/340 * 100)),
            vjust = -.5)

# Produce a csv file of predicted output
write.csv(output.pred, file="C:/Users/japas_000/Documents/Thesis/model/knn_output.csv")

#End of model
```

Figure 7. Analysis of kNN classifier

Analysis of ensemble classifier

The algorithms applied in ensemble learning method were evaluated differently than kNN using powerful tools available in the RTextTools package of R. The create_analytics() function rendered all performance measures relevant to the test data classified by an ensemble of algorithms - SVM, MaxEnt, GLMNET, Random Forests, Neural Networks, Boosting, Bagging and Decision Trees. The function returns a container with four different summaries: by label (e.g., topic), by algorithm, by document, and an ensemble summary. In this case, as all data in the training and testing sets have corresponding labels, create_analytics() will check the results of the learning algorithms against the true values to determine the accuracy of the process [28].

The label summary provides statistics on each of the requirement categories in the classified data. It includes number of documents classified into each label by the ensemble method as well as their probabilities. The algorithm summary provides performance values: Precision and Recall, provided by each of the algorithm used in the ensemble. The document summary provides statistics on each document (statement here) classified and includes each algorithm's prediction, the algorithm's probability score, the number of algorithms that agreed on the same label, which algorithm had the highest probability score for its prediction, and the original label of that statement. Finally, the ensemble summary provides details on the ensemble classifier as a whole (Table 3), also including coverage for an n-ensemble agreement. For this model, n=8 as 8 algorithms were employed. Coverage simply refers to the percentage of documents that meet the recall accuracy threshold [25]. For example, if there are 10 statements and only two statements meet the eight ensemble agreement threshold, then our coverage is 20%.

It is mathematically defined as k/n, where k is the percentage of cases that meet the ensemble threshold, and n represents total cases. Table 2 reports the coverage and recall accuracy for different levels of ensemble agreement. The general trend is for coverage to decrease while recall increases. In the current model, just 18% of the requirements have eight algorithms that agree. However, recall accuracy is 96% for these when the 8 algorithms do agree.

n	n-ensemble coverage	n-ensemble recall
n >= 1	1.00	0.67
n >= 2	1.00	0.67
n >= 3	0.99	0.68
n >= 4	0.93	0.69
n >= 5	0.77	0.75
n >= 6	0.63	0.79
n >= 7	0.49	0.84
n >= 8	0.18	0.96

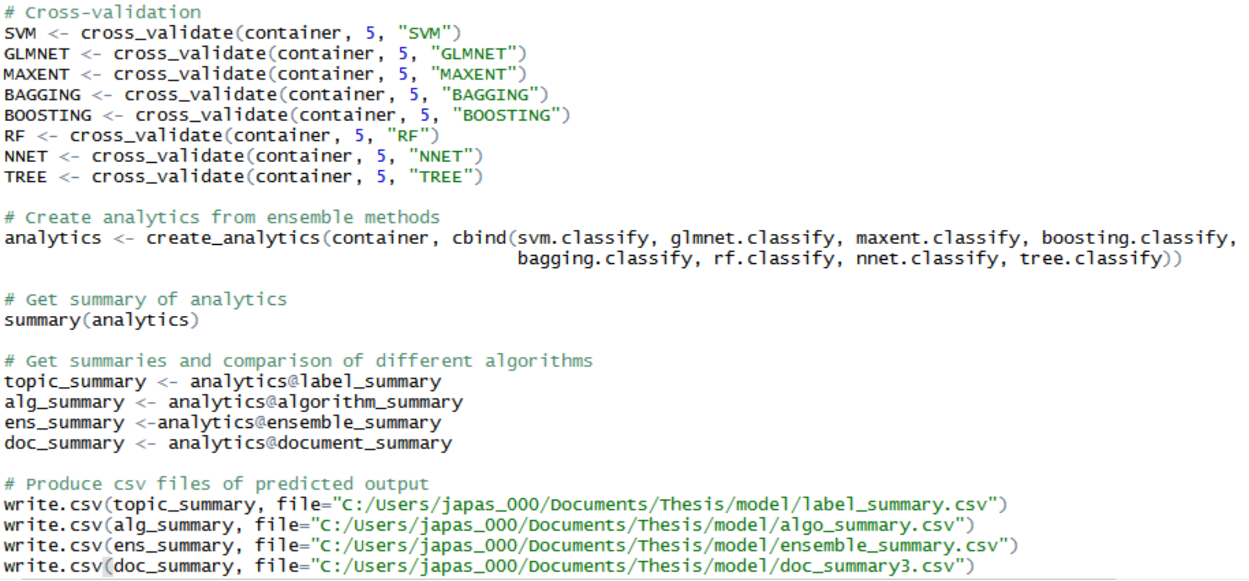
Table 3. Ensemble Summary

Algorithm	Precision	Recall	F-measure
SVM	0.63	0.54	0.56
GLMNET	0.57	0.60	0.58
MaxEnt	0.63	0.64	0.59
Boosting	0.62	0.59	0.58
Bagging	0.65	0.59	0.59
Random Forest	0.69	0.63	0.63

Neural Network	0.17	0.21	0.17
Tree	0.45	0.49	0.44

Table 4. Ensemble algorithm performance summary

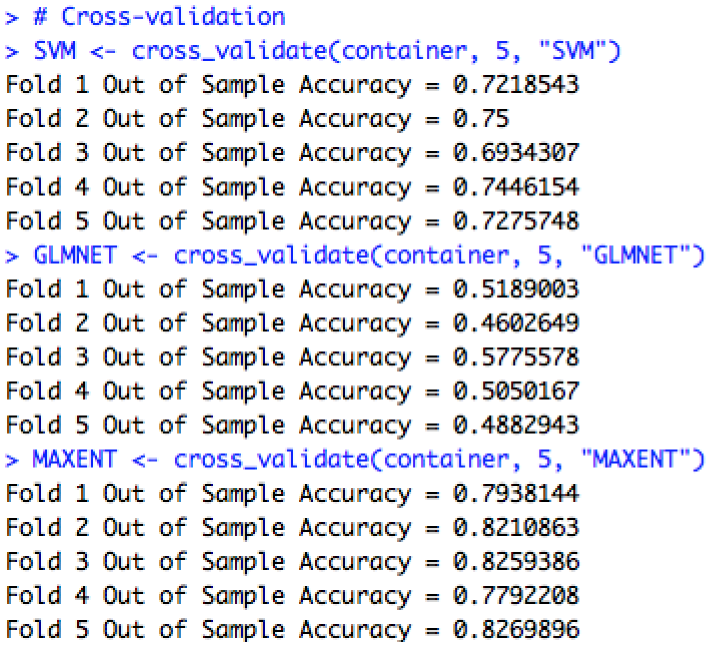
Figure 8 below presents the source code for obtaining all summaries and parameters discussed above using create_analytics() method.



```
# Cross-validation
SVM <- cross_validate(container, 5, "SVM")
GLMNET <- cross_validate(container, 5, "GLMNET")
MAXENT <- cross_validate(container, 5, "MAXENT")
BAGGING <- cross_validate(container, 5, "BAGGING")
BOOSTING <- cross_validate(container, 5, "BOOSTING")
RF <- cross_validate(container, 5, "RF")
NNET <- cross_validate(container, 5, "NNET")
TREE <- cross_validate(container, 5, "TREE")

# Create analytics from ensemble methods
analytics <- create_analytics(container, cbind(svm.classify, glmnet.classify, maxent.classify, boosting.classify,
                                               bagging.classify, rf.classify, nnet.classify, tree.classify))

# Get summary of analytics
summary(analytics)

# Get summaries and comparison of different algorithms
topic_summary <- analytics@label_summary
alg_summary <- analytics@algorithm_summary
ens_summary <-analytics@ensemble_summary
doc_summary <- analytics@document_summary

# Produce csv files of predicted output
write.csv(topic_summary, file="C:/Users/japas_000/Documents/Thesis/model/label_summary.csv")
write.csv(alg_summary, file="C:/Users/japas_000/Documents/Thesis/model/algo_summary.csv")
write.csv(ens_summary, file="C:/Users/japas_000/Documents/Thesis/model/ensemble_summary.csv")
write.csv(doc_summary, file="C:/Users/japas_000/Documents/Thesis/model/doc_summary3.csv")
```

Figure 8. Analysis of ensemble approach

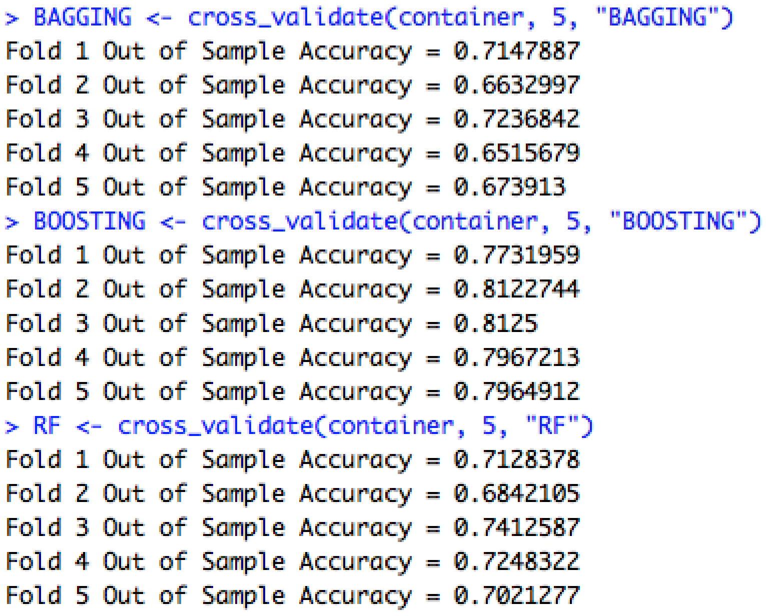
I also attempted to apply 5-fold cross-validation for the ensemble in order to improve performance as can be seen from Figure 8. For a data set of medium size as this one, 5 is a good number for cross-validation. For larger datasets one can use a number like 10. The accuracy after every fold for each algorithm can be seen from Figures 9, 10 and 11 below.



```
> # Cross-validation
> SVM <- cross_validate(container, 5, "SVM")
Fold 1 Out of Sample Accuracy = 0.7218543
Fold 2 Out of Sample Accuracy = 0.75
Fold 3 Out of Sample Accuracy = 0.6934307
Fold 4 Out of Sample Accuracy = 0.7446154
Fold 5 Out of Sample Accuracy = 0.7275748
> GLMNET <- cross_validate(container, 5, "GLMNET")
Fold 1 Out of Sample Accuracy = 0.5189003
Fold 2 Out of Sample Accuracy = 0.4602649
Fold 3 Out of Sample Accuracy = 0.5775578
Fold 4 Out of Sample Accuracy = 0.5050167
Fold 5 Out of Sample Accuracy = 0.4882943
> MAXENT <- cross_validate(container, 5, "MAXENT")
Fold 1 Out of Sample Accuracy = 0.7938144
Fold 2 Out of Sample Accuracy = 0.8210863
Fold 3 Out of Sample Accuracy = 0.8259386
Fold 4 Out of Sample Accuracy = 0.7792208
Fold 5 Out of Sample Accuracy = 0.8269896
```

Figure 9. Cross-validation results

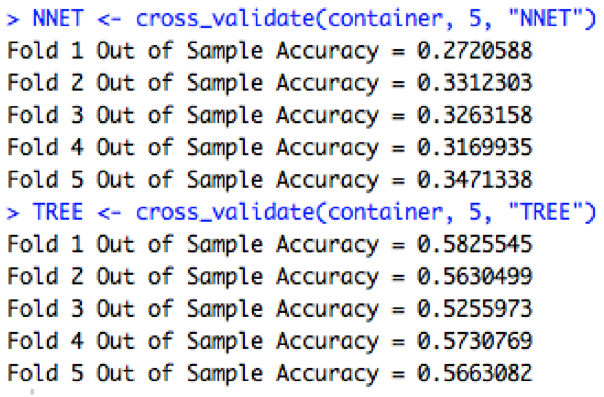
From above, it can be seen that on applying 5-fold cross validation to the ensemble, the Max Entropy algorithm yielded an accuracy value of 82% which is notable.



```
> BAGGING <- cross_validate(container, 5, "BAGGING")
Fold 1 Out of Sample Accuracy = 0.7147887
Fold 2 Out of Sample Accuracy = 0.6632997
Fold 3 Out of Sample Accuracy = 0.7236842
Fold 4 Out of Sample Accuracy = 0.6515679
Fold 5 Out of Sample Accuracy = 0.673913
> BOOSTING <- cross_validate(container, 5, "BOOSTING")
Fold 1 Out of Sample Accuracy = 0.7731959
Fold 2 Out of Sample Accuracy = 0.8122744
Fold 3 Out of Sample Accuracy = 0.8125
Fold 4 Out of Sample Accuracy = 0.7967213
Fold 5 Out of Sample Accuracy = 0.7964912
> RF <- cross_validate(container, 5, "RF")
Fold 1 Out of Sample Accuracy = 0.7128378
Fold 2 Out of Sample Accuracy = 0.6842105
Fold 3 Out of Sample Accuracy = 0.7412587
Fold 4 Out of Sample Accuracy = 0.7248322
Fold 5 Out of Sample Accuracy = 0.7021277
```

Figure 10. Cross-validation results

From above it can be noted that Boosting also improved the accuracy to almost 80% using 5-fold cross-validation. As expected from summary values, Neural Network and trees did not do any good to the accuracy of the model.



```
> NNET <- cross_validate(container, 5, "NNET")
Fold 1 Out of Sample Accuracy = 0.2720588
Fold 2 Out of Sample Accuracy = 0.3312303
Fold 3 Out of Sample Accuracy = 0.3263158
Fold 4 Out of Sample Accuracy = 0.3169935
Fold 5 Out of Sample Accuracy = 0.3471338
> TREE <- cross_validate(container, 5, "TREE")
Fold 1 Out of Sample Accuracy = 0.5825545
Fold 2 Out of Sample Accuracy = 0.5630499
Fold 3 Out of Sample Accuracy = 0.5255973
Fold 4 Out of Sample Accuracy = 0.5730769
Fold 5 Out of Sample Accuracy = 0.5663082
```

Figure 11. Cross-validation results

From the label summary discussed before, we obtained an important result on the percentage of documents correctly classified by the ensemble method. The table below shows number of statements classified by the ensemble method and the percentage of them which were correctly classified. It can be seen that for the label ‘External Call’ over 92% of the statements were correctly classified and for ‘User Interface Navigation’ all of them were correctly classified, while only 5% were correctly classified for ‘Mathematical principle/rule’.

Category	**Number of statements classified**	**Percentage of statements correctly classified**
Application Logic	154	57.14
Data Input	63	58.66
Data Output	62	60
Data Persistence	3	42.85
Data Validation	65	100

Event Trigger	166	92.14
External Call	2	66.67
Math Algorithm/procedure	38	78.78
Mathematical principle/rule	8	5.45
User Interface	92	62.79
User Interface Logic	84	69
User Interface Navigation	4	100

Table 5. Label Summary

Comparison

From the analyses above, it can be observed that the kNN approach clearly outperformed the ensemble approach in terms of performance measures – Precision and Recall.

CHAPTER 5

DISCUSSION

The final chapter in the thesis summarizes the work performed and provides conclusion and future scope for the study conducted. It also discusses challenges faced throughout this project.

Summary

The first chapter introduced readers to the subject of this thesis, highlighting the importance of requirements management and need to automate classification to ease out the subsequent phases of development. It also talked about the role of machine learning and text mining in requirements engineering and provided an insight into how data analytics works in context of textual statements of software requirements.

The second chapter discussed background work in the core area of classification of software requirements using text mining, machine learning and NLP. It explained and compared different prior research works conducted and using the insights gained, ultimately charted a route for the current work to proceed.

The third chapter explained in detail the methodology employed in conducting the study and creation of classifier model, including the language and platform; data, tools, technologies and packages; and the classification algorithms hence used.

The fourth chapter dealt with analysis of performance measures of individual and ensemble algorithms and drew comparisons amongst them in terms of those measures.

Challenges

In text mining, it is always a challenge working with free-form text data. Naturally, it was quite challenging to work with data containing all text and refining it to extract relevant terms on which the model can be trained. Another issue arises from the complexity of natural language itself. Although in enterprise systems, requirements are clearly specified the language and style of writing them may differ from person to person. Therefore it is a huge challenge to unify different styles of writing or established common format for them. In addition to this, one word may have more than one meanings. It is a challenge to determine semantics of each text. In many approaches like this one, the model is trained using document frequencies - the frequencies of occurrence of terms in the text. However the semantic aspect of text should also be taken into consideration by the model to effectively train over it, along with the causal relationships between words. There is also the issue of the model getting ‘confused’. In this case, there were many statements which a human could possibly classify into more than one labels depending on the context in which it appeared in the system. Therefore, domain knowledge integration is of utmost importance here. Lastly, there is the issue of memory. With large size of text data the memory requirement becomes large. This requires high RAM machines with high processing powers.

There were no major impediments faced in this project than the ultimate question of improving the efficiency of classifier model. Data cleaning was one minor obstacle faced in the creation of classifier model in this study. When data was less cleaned or uncleaned it resulted in memory requirement being very high for a relatively less number of documents/statements. Another issue was selection of classification algorithm that would serve best for the task at hand. However, with the help of ensemble learning this challenge was overcome. Lastly, there was the problem of context, wherein there were many statements that could very well be classified into category other than the one it was classified manually. For instance the statement ‘The user then clicks on the formula button present in the toolbar’ appeared as Event Trigger category but could also be classified as User Interface Navigation if the context is unknown.

Conclusion, recommendations and future scope

In a nutshell, this thesis aimed to study and apply text classification over functional software requirements belonging to enterprise applications using different algorithms and determine how accurately the classifier was able to predict requirement type for each statement presented to it. However, one limitation of this study was the mediocre value of precision obtained. There is a potential to improve this value by training more data - meaning, consolidating requirements from more enterprise applications of a particular type. To be on a neutral front so as to avoid potential confusion during manual classification, an approach of crowd-sourcing could be followed, where in paid surveys or activities are conducted for masses all over the world to contribute to the classification, thereby improving the quality of training data. As discussed previously there are some factors which affect the accuracy of classifier, such as semantics and word relationships and context. To avoid the model from getting confused, validity of training data is of high significance. The future scope can also take in the approach of a weighted-kNN, where in the number of neighbors are determined based on weighing of terms in training data, which in this case could be the terms associated with the enterprise domain. R project recently launched a knn function called kknn for the same.

Overcoming these issues in future could significantly take up the accuracy of such classifiers. In addition to this the label summary showed a very small percentage of statements correctly classified as ‘Mathematical principle/rule’. This category corresponds to a specific application of domain of applications in enterprise systems. In this case, it would be best to accumulate such domain rules or subject-matter expertise to train over them in order to improve accuracy of the model. Again this leads us to realize the importance of having historical data for training effectively. A data warehouse would serve the purpose for large-scale enterprise applications.

REFERENCES

[1] Software Engineering Institute, Carnegie Mellon University http://www.sei.cmu.edu/productlines/frame_report/req_eng.htm.

[2] Ghazarian, A. (2012, September). Characterization of functional software requirements space: The law of requirements taxonomic growth. In *Requirements Engineering Conference (RE), 2012 20th IEEE International* (pp. 241-250). IEEE.

[3] Sommerville, I., Software Engineering, 9th Edition, Addison-Wesley, ISBN-13: 978-0-13-703515-1, 2011.

[4] Raamesh, L., & Uma, G. V. (2010). Reliable Mining of Automatically Generated Test Cases from Software Requirements Specification (SRS). *arXiv preprint arXiv:1002.1199*.

[5] Sateli, B., Angius, E., Rajivelu, S. S., & Witte, R. (2012). Can text mining assistants help to improve requirements specifications? *Mining Unstructured Data (MUD 2012), Canada*.

[6] Cybulski, J. L., & Reed, K. (1998, December). Computer-assisted analysis and refinement of informal software requirements documents. In *Software Engineering Conference, 1998. Proceedings. 1998 Asia Pacific* (pp. 128-135). IEEE.

[7] Vlas, R., & Robinson, W. N. (2011, January). A rule-based natural language technique for requirements discovery and classification in open-source software development projects. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on* (pp. 1-10). IEEE.

[8] Slankas, J., & Williams, L. (2013, May). Automated extraction of non-functional requirements in available documentation. In *Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on* (pp. 9-16). IEEE.

[9] Cleland-Huang, J., Settimi, R., Zou, X., & Solc, P. (2007). Automated classification of non-functional requirements. *Requirements Engineering*, *12*(2), 103-120.

[10] Rashwan, A., Ormandjieva, O., & Witte, R. (2013, July). Ontology-based classification of non-functional requirements in software specifications: a new corpus

and svm-based classifier. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual* (pp. 381-386). IEEE.

[11] Casamayor, A., Godoy, D., & Campo, M. (2009). Semi-Supervised Classification of Non-Functional Requirements: An Empirical Analysis.*Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, *13*(44), 35-45.

[12] Hussain, I., Kosseim, L., & Ormandjieva, O. (2008). Using linguistic knowledge to classify non-functional requirements in SRS documents. In *Natural Language and Information Systems* (pp. 287-298). Springer Berlin Heidelberg.

[13] Ko, Y., Park, S., & Seo, J. (2000). Web-based requirements elicitation supporting system using requirements categorization. In *Proceedings of Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE 2000), Chicago, USA* (pp. 344-351).

[14] van Lamsweerde, A. (2009). Reasoning about alternative requirements options. In *Conceptual Modeling: Foundations and Applications* (pp. 380-397). Springer Berlin Heidelberg.

[15] Luisa, M., Mariangela, F., & Pierluigi, N. I. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, *9*(1), 40-56.

[16] Hussain, I., Ormandjieva, O., & Kosseim, L. (2012, September). Lasr: A tool for large scale annotation of software requirements. In *Empirical Requirements Engineering (EmpiRE), 2012 IEEE Second International Workshop on* (pp. 57-60). IEEE.

[17] Ormandjieva, O., Hussain, I., & Kosseim, L. (2007, September). Toward a text classification system for the quality assessment of software requirements written in natural language. In *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting* (pp. 39-45). ACM.

[18] Al Balushi, T. H., Sampaio, P. R. F., Dabhi, D., & Loucopoulos, P. (2007). ElicitO: a quality ontology-guided NFR elicitation tool. In *Requirements Engineering: Foundation for Software Quality* (pp. 306-319). Springer Berlin Heidelberg.

[19] Menzies, T., Krishna, R., Pryor, D. (2015). The Promise Repository of Empirical Software Engineering Data*; http://openscience.us/repo. North Carolina State University*, Department of Computer Science.

[20] Palmer, J. D., & Liang, Y. (1992). Indexing and clustering of software requirements specifications. *Information and decision Technologies*, *18*(4), 283-299.

[21] Park, S., & Palmer, J. D. (1994, June). Automated support to system modeling from informal software requirements. In *SEKE* (pp. 86-93).

[22] Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996, August). Knowledge Discovery and Data Mining: Towards a Unifying Framework. In *KDD* (Vol. 96, pp. 82-88).

[23] Ko, Y., Park, S., Seo, J., & Choi, S. (2007). Using classification techniques for informal requirements in the requirements analysis-supporting system.*Information and Software Technology*, *49*(11), 1128-1140.

[24] Aas, K., Eikvil, L. (1999). *Text Categorisation: A Survey*.

[25] Jurka, T. P., Collingwood, L., Boydstun, A. E., Grossman, E., & van Atteveldt, W. (2013). RTextTools: A supervised learning package for text classification. *The R Journal*, *5*(1), 6-12.

[26] Ensemble Learning. (n.d.). Retrieved March 20, 2016, from https://en.wikipedia.org/wiki/Ensemble_learning.

[27] Neural Network Classifier. (n.d.). Retrieved March 20, 2016, from http://www.robots.ox.ac.uk/~dclaus/digits/neural.htm.

[28] The R Project for Statistical Computing. (n.d.). Retrieved March 20, 2016, from https://www.r-project.org/.

[29] Herbert Julius Garonfolo. (n.d.). Retrieved October 25, 2015, from http://garonfolo.dk/herbert/2015/05/r-text-classification-using-a-k-nearest-neighbour-model/.