Feature and Variability Extraction from Informal Product Descriptions

Author:
Meng Zhang

27 June, 2017

Advisors:
Prof. Dr. Gunter Saake
Dr.-Ing. Sandro Schulze
MSc. Yang Li
University of Magdeburg
Meng Zhang

Abstract

Software Product Line Engineering (SPLE) techniques has been commonly used in software products development field. Hence, automation in the SPLE executing process is desirable, especially with respect to variability management in which feature model has been extensively used to analyze the commonality and variability in a domain of related software systems.

In most of the previous researches, fundamental artifacts such as software requirements are used to comprise common and variable features and traced to other artifacts efficiently, such as source code. However, the most of these kinds of artifacts is not directly accessible to researchers. And therefore an approach is came up to generate feature models from online product descriptions which is informal compared with artifacts like software requirements, but can be directly accessible.

The main focus of the thesis is trying to come up with an approach which combines both of the semantic and syntactical aspects for measuring the similarity between descriptors retrieved from existing descriptions to extract variability and generate feature models with high accuracy and keeping the whole feature model process as automatic as possible. And the relative stakeholders can also control emphasis points of the generated feature models considering their needs and preferences.

We set up a sample work process to examine if our generated approach is useful by comparing it with an existing approach which is came up in previous research and also using evaluation measures in data mining field to measure the quality of our approach. From the evaluation results, we can prove that our approach can work efficiently and generate more accurate results which are also more comprehensible.

**Keywords:** Software Product Line Engineering, Variability Management, Feature Models, Features
Contents

List of Figures viii
List of Tables ix
List of Code Listings xi

1 Introduction 1
1.1 Motivation ........................................ 2
1.2 Objectives ........................................ 2

2 Background 5
2.1 Natural Language Processing .......................... 5
2.2 General Feature Model Construction Framework .......... 6
  2.2.1 Components Details .............................. 7

3 Design of Feature Model Generation Process 13
3.1 Raw Descriptions Collection .......................... 14
3.2 Text Parsing ......................................... 14
  3.2.1 Descriptor Extraction ............................. 14
  3.2.2 Pronoun Replacement ............................. 15
  3.2.3 Stop Word Removing ............................. 15
  3.2.4 Stemming Each Word to Morphological Root .......... 16
  3.2.5 Semantic Role Labeling ........................... 16
3.3 Similarity Calculation ................................ 17
  3.3.1 Word Similarity Measure .......................... 18
  3.3.2 Text Segments Similarity Measure .................. 23
  3.3.3 Behavioral Vectors Similarity Measure ............... 26
3.4 Feature Model Generation ............................ 28
  3.4.1 Primitive Feature Model Structure Construction ....... 30
  3.4.2 Primitive Tree Structure Simplification and Flattening 45
  3.4.3 Relationships Detection ........................... 46
  3.4.4 Feature Naming ................................... 48

4 Feature Model Construction Process Implementation 51
  4.1 Implementation of Clustering Algorithm for Primitive Hierarchy Con-
struction .............................................. 51
  4.2 Primitive Tree Structure Simplification and Flattening .... 53
  4.3 Relationships Detection ............................... 54
## Contents

4.4 Feature Naming ................................................. 56

5 Evaluation ......................................................... 57
  5.1 Evaluation Criteria ........................................... 57
  5.2 Evaluation Setups ............................................. 58
    5.2.1 Different Aspects to Evaluate ............................. 58
    5.2.2 Methodology for Evaluation Work ........................... 58
  5.3 Evaluation Results Analysis ................................. 65
    5.3.1 Qualified Evaluation Result Analysis ...................... 65
    5.3.2 Quantitative Evaluation Result Analysis ................. 66
    5.3.3 Evaluation Conclusion .................................... 67

6 Threats to Validity .................................................. 69

7 Related Work ......................................................... 71

8 Summary and Feature Work .......................................... 73

Bibliography ............................................................. 75
List of Figures

2.1 The whole general process for feature model detection . . . . . . . . . . 7
2.2 Flattening and simplifying the clustering outcome in the feature dia-
gram creation stage . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.1 An overview of the process in our feature extraction work . . . . . . . 14
3.2 Process of descriptors similarity calculation . . . . . . . . . . . . . . . 18
3.3 Basic architecture of the different types of the text similarity approaches 19
3.4 Basic architecture of the different types of the string-based similarity
approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.5 Basic architecture of the different types of the corpus-based similarity
approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
3.6 Basic architecture of the different types of the knowledge-based sim-
ilarity approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.7 General process for feature naming . . . . . . . . . . . . . . . . . . . 24
3.8 Process for feature model generation . . . . . . . . . . . . . . . . . . . 29
3.9 General process for feature naming . . . . . . . . . . . . . . . . . . . 30
3.10 A hierarchical clustering result of four points shown as a dendrogram
and as nested clusters . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3.11 Single link clustering result of the six points given in table 3.6 . . . . 34
3.12 Complete link clustering of the six points given in table 3.6 . . . . . 35
3.13 Group average clustering of the six points given in table 3.6 . . . . . 35
3.14 Ward’s distance measure of the six points given in table 3.6 . . . . . 36
3.15 Clustering result using MIN distance measure given two non-equal-
size and non-spherical clusters . . . . . . . . . . . . . . . . . . . . . . . 37
3.16 Clustering result using MIN distance measure given two clusters with
outliers between them . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.17 Clustering result using MAX distance measure given two clusters with
outliers between them . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.18</td>
<td>Clustering result using MAX distance given two non-equal-size clusters</td>
<td>38</td>
</tr>
<tr>
<td>3.19</td>
<td>Clustering result using Ward’s distance measure given two clusters with outliers between them</td>
<td>39</td>
</tr>
<tr>
<td>3.20</td>
<td>Clustering result using Ward’s distance measure given two spherical clusters</td>
<td>39</td>
</tr>
<tr>
<td>3.21</td>
<td>Clustering result using Group Average distance measure given two non-spherical and non-equal clusters</td>
<td>40</td>
</tr>
<tr>
<td>3.22</td>
<td>Clustering result using Ward’s distance measure given two spherical clusters</td>
<td>40</td>
</tr>
<tr>
<td>3.23</td>
<td>Flattening and simplifying the clustering outcome in the feature diagram creation stage</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>Hierarchical agglomerative clustering result using MAX distance measure</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Hierarchical agglomerative clustering result using MIN distance measure</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Hierarchical agglomerative clustering result using Group Average distance measure</td>
<td>53</td>
</tr>
<tr>
<td>4.4</td>
<td>Hierarchical structure generated by hierarchical agglomerative clustering with threshold for simplification</td>
<td>54</td>
</tr>
<tr>
<td>4.5</td>
<td>Mandatory/optional relationship detection</td>
<td>55</td>
</tr>
<tr>
<td>4.6</td>
<td>OR/XOR relationship detection</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>The final feature model tree built in our work</td>
<td>56</td>
</tr>
<tr>
<td>5.1</td>
<td>Feature model tree built in our work</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Feature model tree built in [DDH+13]</td>
<td>59</td>
</tr>
<tr>
<td>5.3</td>
<td>Generated feature model on antivirus software field</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>Generated feature model on multimedia player field</td>
<td>60</td>
</tr>
<tr>
<td>5.5</td>
<td>Generated feature model on web browser field</td>
<td>61</td>
</tr>
<tr>
<td>5.6</td>
<td>Truth set on antivirus software field</td>
<td>61</td>
</tr>
<tr>
<td>5.7</td>
<td>Truth set on multimedia player field</td>
<td>61</td>
</tr>
<tr>
<td>5.8</td>
<td>Truth set on web browser field</td>
<td>62</td>
</tr>
</tbody>
</table>
## List of Tables

3.1 Descriptors got from an antivirus software description  . . . . . . . . . . 15
3.2 Examples of behaviors  . . . . . . . . . . . . . . . . . . . . . . . . . . 16
3.3 Comparison of different words similarity measure  . . . . . . . . . . . . 21
3.4 Different weights for some sample words  . . . . . . . . . . . . . . . . . 25
3.5 Samples of splitting descriptors into behavioral vectors  . . . . . . . . 27
3.6 Euclidean distances matrix for 6 points  . . . . . . . . . . . . . . . . . 33

5.1 Evaluation values for our generated feature models  . . . . . . . . . . . 67
List of Code Listings

3.1 Process of inputting the behavioral vectors . . . . . . . . . . . . . . . 17
3.2 Process of implementing similarity measure function . . . . . . . . . 22
3.3 Vector similarity matrix generation . . . . . . . . . . . . . . . . . . . 28
3.4 Basic agglomerative hierarchical clustering process . . . . . . . . . . 32
3.5 Top-down: basic divisive clustering process . . . . . . . . . . . . . . . 32
3.6 Min distance measure between cluster X and Y . . . . . . . . . . . . 34
3.7 Top-down: basic divisive clustering process . . . . . . . . . . . . . . . 34
3.8 Group average distance between cluster X and Y . . . . . . . . . . . 35
3.9 Ward’s distance between cluster X and Y . . . . . . . . . . . . . . . . 36
3.10 K-means clustering algorithm . . . . . . . . . . . . . . . . . . . . . . 41
3.11 Class Node . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
3.12 Relationship detection . . . . . . . . . . . . . . . . . . . . . . . . . . 48
5.1 Cohesion Calculation . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
5.2 Separation Calculation . . . . . . . . . . . . . . . . . . . . . . . . . . 63
List of Code Listings
1. Introduction

Although it has been proven that Software Product Line Engineering (SPLE) technique can be successfully used to cut down the products development cost and time consumption, applying SPLE techniques is still very challenging because it is time-consuming and expensive to extract common and variable features from the existing artifacts which is required to be analyzed. So we need to come up with a ideal approach of feature model generation which can support an efficient SPLE execution process.

SPLE focuses on reusability of software-related artifacts such as software requirements, design documents, source code, text cases and customer reviews. Although several techniques have been evolved aiming at different artifacts such as feature location in source code, software requirement as initial artifact providing more detailed and complete variability information than other artifacts [IRB14b], [IRB14a], [CW07], [CZZM05], [ADCM13], [CYWK12]. When using these artifacts mentioned above as the input materials for feature extraction process, several problems will still be faced. Firstly some kinds of artifacts among the different kinds of artifacts mentioned above such as source code, software requirements, design documents are not directly accessible to the users. So it is very hard to retrieve these artifacts as the input for feature extraction. Secondly, the resources from which the artifacts are retrieved such as the relevant websites form which we get customer reviews are not reliable enough to support our work. For example, when we use customer reviews as the artifact, we will face the problem that there are strong customers subjective emotions to the products filled in the comments which will strongly influence our feature extraction process. Considering these problems, there are also several different approaches which have been suggested for semi or fully automatic feature model construction based on descriptions of products which can be easily retrieved from website of software products [DDH+13], [ACP+12], [CSW08], [ASB+08]. These methods commonly use different semantic similarity measures and clustering techniques to extract features and build the hierarchical structures of features and based on different levels of the artifacts, features in feature models will be organized into multiple levels of increasing details, typically using a tree structure to represent. A recent report have mentioned that until now variability modeling have been the
most frequently used reported notation in products development field \cite{BRN+13}. And there are also several tools which has been developed for academic or industrial aims to automate the analysis, configuration and transformation process of feature model generation \cite{ACP+12}, \cite{TKB+14}, \cite{CBH11}. The previous researches are greatly helpful for future development of feature model construction technique as the foundational knowledge store.

1.1 Motivation

For the previously proposed processes of feature extraction for feature model construction in SPL\textsc{e}, there are still some problems need to be solved. Firstly, in most of the situations, software requirements are used as fundamental artifact comprising complete information regarding variant points and provide traceability links to other artifacts, such as source code. However, this kind of artifacts is not accessible to researchers. And therefore an approach is required to generate feature models from online product descriptions which is informal compared with requirements, but accessible. Another problem is that although there are some studies conducted previously that propose methods to realize feature extraction and feature model generation, the previous studies commonly only concentrate on pure semantic similarity or string sequences similarity metrics used between two single phrases or sentences in order to identify and analyze variability without considering the basic syntactical structures of them which will lead to the decrease of accuracy and efficiency of the result and there are also so much human manual involvement in the whole feature model generation process.

In our work, we are trying to come up with a ideal approach for feature model construction which can solve the problems existing in current situation and is different from previous existing approaches with higher feasibility and accuracy.

1.2 Objectives

There are basically 3 main objectives we want to achieve in the work of the feature detection from textual descriptions. Achieving these 3 goals is the main aim and the mainline in the thesis.

1) Find suitable approaches for raw descriptions collection, preprocessing and descriptors semantic similarity calculation

We need retrieve the textual descriptions of the software products from the website using the file capture tool and pre-process the retrieved descriptions to transfer them into suitable formation for later use. Then we have to choose the suitable similarity measure and calculate the similarities for each pair of descriptors to construct the similarity matrix used for feature model generation.

2) Build suitable process for feature model generation

After coming up with the idea of how to do raw descriptions collection, preprocessing and descriptors similarity calculation, based on it, we need to try to choose one of the most suitable algorithms e.g. hierarchical clustering to build the primitive feature model structure. Then try to flatten and simplify the primitive feature structure we have got and detect the relationship types between the parent nodes and child nodes in our primitive feature model tree to get the final resultant feature model.
3) Evaluation of our own approach for feature model generation
After implementing our designed process of feature model construction, we will evaluate our process through comparing our resultant generated sample feature model with other finished feature models created in other previous studies given the same data domain and use some suitable evaluation measures in data mining field to check if our work can have a better performance and which part of our work should be improved.
Implementation of such procedures will greatly support the improvement of the feature model generation from textual description process and help researchers more easily understand the relevant studying fields of this topic. And our approach will also make the process of constructing a product line feature model from natural-language production descriptions or requirements texts to generate the robust and accurate derivation of products more reliable.

The rest of the paper is structured as follows. Section 2 introduce the necessary techniques and general process used for feature model generation., while section 3 talks about the implementation of the whole process of our own feature model generation in greater details. Section 4 presents the evaluation results for our own approaches of feature model generation. And finally, section 5 concludes all the contents in the thesis and explain the feature works needed to be done.
1. Introduction
2. Background

Before we describe our approach of feature model generation in greater details, in this part, we will illustrate the natural language processing technique which is one of the most important techniques used in feature model construction from descriptions work and the general process for feature extraction from texts.

2.1 Natural Language Processing

Natural language processing (NLP) is a field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human natural languages, and in particular concerned with programming computers to fruitfully process natural language. Challenges in natural language processing frequently involve natural language understanding, natural language generation which is frequently in formal, machine-readable logical forms, connecting language and machine perception, managing human-computer dialog systems, or some combinations thereof.

There are several natural languages processing steps which can support to achieve the whole natural language processing work. These steps are mainly used to do the relative processing works for example split one graph into several separated sentences, deal with the separated sentences like replace the pronouns e.g. He, she, and it with their anaphors i.e. the nouns to which they refer, stem each word to its morphological root, remove stop words which are extremely common and not helpful for feature extraction process and analysis the syntactic structure of the natural language sentence and put different segments into the different components such as subject, verb, object and complement and so on. Each term in the sentences also needs to be tagged with their part-of-speech positions in some situations for better understanding of natural language segments.

1). Tokenization

A tokenization process divides text into a sequence of tokens, which roughly correspond to ”words”. An ancillary can be used with this tokenization process to provide the ability to split text into sentences.
2). Sentence Breaking

For this part, the sentences breaking work needs to be achieved, also known as sentence boundary disambiguation (SBD) or sentence boundary detection which is used to decide where sentences begin and end. For the input of natural language processing, it is always required to be spitted into sentences for many reasons. Sentence splitting is a deterministic consequence of tokenization: a sentence ends when a sentence-ending character (., !, or ?) is found which is not grouped with other characters into a token (such as for an abbreviation or number), though it may still include a few tokens that can follow a sentence ending character as part of the same sentence such as quotes and brackets.

3). Morphological Root Stemming

For this step, after dividing the input into several sentences, the words in the sentences need to be analyzed about how they are formed, and their relationship to other words. Then make the words to their morphological root forms.

4). Part-Of-Speech Tagger

A Part-Of-Speech Tagger (POS Tagger) is a piece of work that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc..

5). Natural Language Parser

A natural language parser is used to work out the grammatical structure of sentences, for instance, which groups of words go together as phrases and which words are the subject or object of a verb. In another word, a natural language parser is used to understand the exact meaning of the sentences emphasizing the importance of grammatical divisions such as subject and predicate.

6). Named Entity Recognizer

Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizer for English, particularly for the 3 classes (PERSON, ORGANIZATION, LOCATION), and we also make available based on other models for different languages and circumstances.

2.2 General Feature Model Construction Framework

In this section, we will present a general framework, which consists of 2 main steps during the whole feature model construction process from detection of raw natural language documents of products as input to output of finished feature model or feature list with different structures such as trees as result. The first one is prepressing step in which we use natural language processing techniques to process the textual contents like descriptions or requirements got from external resource to
make them with suitable formations for later feature extractions and second step is post-processing in which we extract features based on the result of pre-processing step which can be done with different algorithms such as hierarchical clustering, association rules mining and so on. Figure 2.1 shows the whole general process for feature extraction as we have mentioned above containing 2 steps.

**2.2.1 Components Details**

For this part, some details will be illustrated about how the natural language processing and other techniques are used in general feature extraction process and how the whole overall process can work step by step.

1). Raw Information Collection

First of all, the useful raw natural language documents of products such as product descriptions or requirements need to be mined in order to get useful information for later feature extraction. The different types of tools can be used to extract descriptions or requirements from the websites such as the repository website containing general product descriptions or other resources. A tool named Screen-scraper was used in one previous research [DDH+13] to scrape the raw software descriptions from the website.
2). Natural Language processing

In preparation for the later feature extraction, the contents contained in retrieved natural language documents of products such as product descriptions or requirements need to be preprocessed for later feature extraction work use.

a). Tokenisation

Tokenisation process divides text into a sequence of tokens and also groups the characters into their corresponding tokens for later sentence breaking step. In previous researches [NSN14], [ACP12], [HCHM13], Tokenisation was seemed as the necessary step for text pre-processing.

b). Sentence Breaking

The contents contained in natural language documents such as requirements or descriptions need to be split into several sentences because of some reasons. In previous research work [IRB14a], [IRB14b], [DDH13], Sentence Breaking step was used to split the requirements or descriptions into several individual segments used for later feature extraction.

c). Morphological Root Stemming

The words in the sentences need to be analyzed to make the words to their morphological root forms. For some researches [KTWF12], [IRB14b], [DDH13] in which feature extraction process was based on the similarity metrics, Morphological Root Stemming is one of the most important steps.

d). Part-of-Speech Tagging

Part-Of-Speech Tagging (POS Tagging) is a piece of work that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective. It is one of the important steps for text pre-processing. In previous researches [IRB14a] and [IRB14b], Part-of-Speech Tagging was used to analyze the syntactical structure of the text segments used for feature extraction.

e). Natural Language Parsing

Natural Language Parsing is used to work out the grammatical structure of sentences, for instance, which groups of words go together as phrases and which words are the subject or object of a verb. In [IRB14a] and [IRB14b], Natural Language Parsing was used to analyze the different constituents of requirements to label them with semantic roles.

f). Terms Weighting

Terms Weighting or sometimes referred to as Weighting Schema is the optimal step used to assign different weights to terms representing their specificities when the similarity between texts need to be calculated for later feature extraction which can be based on the the occurrences in the documents [FCC13], in which $tf - idf$ was used. In previous researches [HCHM13] and [MWG09], $tf - idf$ was used to assign the weights to the terms in order to support the clustering algorithms for feature extraction.
g). Similarity Metrics

Similarity metrics are used to measure the similar degree between two text segments. Cosine similarity metrics were used in [KTWF12]. In [MCS06] and [IRB14a], one measure named Mihelcea, Corley and Strapparava’s (MCS) measure was introduced to calculate the similarity between text segments. The result of feature extraction through clustering will be influenced by choosing the different similarity measures. The analysis of different effects of choosing different similarity measures in clustering has been done in [Hua08].

h). Name Entity Recognizing

Name Entity Recognition (NER) is an optional step which labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. In [BRN13], Name entity recognizing technique was used to build the NER model to label features and integrate constraints.

3). Post-processing

After implementing the pre-processing step and getting the useful information, feature extraction steps can be done.

a). Primitive Feature Extraction

There are many approaches mentioned in previous related research works which can be used for primitive feature extraction. There are two main types of approaches for primitive feature extraction. One includes various clustering algorithms, for example Hierarchical Agglomerative Clustering, K-Means, K-Medoids, and Fuzzy K-Means. Hierarchical Clustering was used in [IRB14a], [IRB14b], [CZZM05] to build the primitive hierarchy based on a set of requirements or descriptions. In [BEG12], the process was provided to label the features and integrate the constraints without considering the structural relations between the extracted features. And A Two-Stage Spherical k-Means Clustering algorithm was used in [DDH13] to get a hierarchy in this step. The other miscellaneous approaches includes approaches such as Association Rule Mining, Latent Dirichlet Allocation, Propositional Logic. And Association Rule Mining technique was also used in [BKS15] to extract the features. After this step, set of primitive features or primitive hierarchy structure will be generated which represent the rudiment of our feature extraction result. For some research works such as [BEG12], only set of features were extracted without considering the structural relations between them. But as for the research works of the feature modeling such as [DDH13] and [IRB14a], after this step, variability information is also needed to be extracted.

b). Variability Information Extraction

Variability information extraction work is mainly be done with following two steps.

I. Hierarchical Tree Construction

This step is used after getting the primitive feature extraction result to get the final resultant feature model with the hierarchical structure. And there are many different candidate processes such as hierarchy simplification and feature association mining which can be used for this step considering the different result types of the
last primitive feature extraction step which might be a primitive hierarchy or a set of features.

If using algorithms such as Hierarchical Agglomerative Clustering in the primitive feature extraction process and getting a hierarchy which is a tree of clusters (each leaf node is a text segments such as descriptors or requirements and also represents a cluster with only one instance). In order to decrease the complexity of the output to better and more precisely represent all of the extracted features, it will be flattened and simplified to get a simplified hierarchy in which each node clearly represents a potential feature. And in [IRB14a], a similarity threshold was set to merge the subnodes in hierarchy to get the simplified hierarchy structure.

As for the situation of using the algorithms such as K-means Clustering to get a set of feature in the primitive feature extraction process, the methods such as feature association mining can be used to extract the feature hierarchy. In [HCHM+13], after getting a set of features, the associations between the different features were mined.

II. Relationship Detection

This is a step used to extract the structural relations between the features in order to build the feature model. Relationships need to be detected between a parent and its child features. The main relationship types mainly considered in previous research works [DDH+13] and [IRB14a] are Mandatory, Optional, OR, and XOR.

The example feature model hierarchy shown below in figure 2.3 contains these four main relationships.

![Feature Model Hierarchy](image)

Figure 2.2: Flattening and simplifying the clustering outcome in the feature diagram creation stage

a). Mandatory: if the parent feature is selected in a configuration, it is implied that the child feature will also be present. The relationship between graph library and edge is the mandatory relationship.

b). Optional: The child feature may or may not be present even if its parent feature has been presented. The relationship between graph library and algorithm are optional.

c). OR: The children features have to be selected at least one under the condition that their parent has been selected. The child nodes number and cycle under the parent node algorithms have the OR relationship.
2.2. General Feature Model Construction Framework

d). XOR: If the parent node has been selected, there should be one and only one children feature being selected. The child nodes directed and undirected under the parent node edges have the XOR relationship.

In previous research work [DDH¹³], the rules created according to the basic definitions of the different relationships were used to deduce the relationships in a hierarchy structure of features. And in [DDH¹³], Association Rule Mining was introduced to mine the feature associations and build the feature model based on a set of features.

c). Feature Naming

Feature naming is a necessary step to name the features extracted from previous steps in order to make them more easily understandable for the users. In [DDH¹³], various of algorithms including the Apriori [AS⁹⁴] and FPGrowth [HPY⁰⁰] through mining frequent item sets were introduced for feature naming.

After post-processing, various outputs can be obtained, such as a feature list or feature model which contain only set of features or features with structural relations between them respectively.
2. Background
3. Design of Feature Model Generation Process

For our research work, we tried to generate our own automatic feature extraction process through using a set of tools and utilizing a set of algorithms in some fields such as data mining which will be implemented in Eclipse platform by java language after being decided to be used in this chapter. In this chapter, we will clearly explain the whole process of designing our own feature model generation process in greater details. Figure 3.1 shows the detailed steps of our own designed whole feature extraction process. We will explain the details of how we design and implement each component in the process in this chapter with choosing antivirus software as the analysis domain to show an example working process in which readers can easily understand how each components can work.
3. Design of Feature Model Generation Process

3.1 Raw Descriptions Collection

First of all, we have to mine raw natural language product descriptions texts in order to generate the feature model. In our work, we used the tool named network reptile to extract descriptions from the website which is a repository of software products which also includes relative general software products descriptions containing a set of bulled items (descriptors) describing specific features. The tool we used can automatically scrape the descriptions from the network.

3.2 Text Parsing

In order to do the feature extraction, first of all, we need to do the preprocessing work to deal with the raw descriptions retrieved from last step which are informal and not suitable for direct use of feature extraction. There are five points we need to achieve with the help of the natural processing tools automatically or semi-automatically.

3.2.1 Descriptor Extraction

After retrieving the descriptions of the software products, we have to split the descriptions into several sentences or phrases and each of them represent one descriptor containing only one potential feature and add then them into a fixed formal list. To achieve this aim, for some simple formation descriptions, we can easily use the natural language processing tool named Stanford Tokenizer, but as for some complicated

---

1softpedia.com
description paragraphs, we have to manually divided them. For the manual process, we referred to a collection of words which point to the potential presence of variability (e.g., different, like, such as) elements in text. For instance, some enumerations such as “...like mobile phones, websites...”, such enumerations can be detected by the (repeated) patterns of “like noun - comma - noun”. So this sample description can be easily split into two descriptors respectively considering mobile phones and websites. For our work, finally, there were in total of 4396 descriptors mined from the antivirus category which is chosen as the example working analysis domain and store them into a bulleted list of descriptors representing different features. Table 3.1 shows the examples of splitting one particular software product description to several descriptors.

<table>
<thead>
<tr>
<th>Description</th>
<th>Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td>a popular choice ensure PC protection against virus, Trojans, rootkits, and other types of malware</td>
<td>A popular choice ensure PC protection against virus, Trojans, rootkits, and other types of malware</td>
</tr>
</tbody>
</table>

Table 3.1: Descriptors got from an antivirus software description

3.2.2 Pronoun Replacement

We utilized the publicly available repository of products descriptions and mainly used nouns, verbs and adjectives for text similarity measure and finally extract the features. So some pronouns such as users or companies names are useless for our similarity measure process and our final feature model result creation. And we used the natural language processing tool named Stanford Log-linear Part-Of-Speech Tagger which provides the function of reading text in some language and assigning parts of speech to each word (and other token), such as noun, verb, adjective. With the help of the natural language process tools, we can easily replace the pronouns with the objects they refer to and only remain useful words such as nouns, verbs and adjectives.

3.2.3 Stop Word Removing

When we try to compute the semantic text similarities, some special words without real meanings such as "the", "an" and "can" are useless for our work even might influence our results. So we need to remove the stop word which are useless and not relative to our work to ensure that we can finally get the right result. But we still keep some words belonging to stop words which exists commonly but have real meaning and are necessary in one sentences such as some verbs like "provide" to keep the semantic complement of the descriptors. In our work , we create a stop word dictionary through gathering the stop words from relative website\(^2\), then we tried to traverse all the words in our descriptors to find and remove the stop words.

3.2.4 Stemming Each Word to Morphological Root

For our work, we try to use simplified descriptors containing verbs, adjectives, nouns to extract the features through some process. But as for the words we use, they need to be stemmed to their morphological root to make them available for semantic similarity measure and other process. To achieve this aim, we can easily use a natural language processing tool named Stanford CoreNLP to traverse all the words to stem them to their morphological root.

3.2.5 Semantic Role Labeling

In our feature extraction process, each descriptor generated from the input textual description need to be transferred into the behavior vector formation which means that each descriptor can be split into several components. We used natural language processing tool named Stanford parser which can be used for analyze the structure of sentences and do the work that recognize which words are the subject or object of a verb and the relationships between them. There is another approach mentioned for the more accurate and automatic labeling result [GJ02] which represents a system that is used to identify the semantic relationships, or semantic roles, filled by constituents of a sentence within a sentence frame. In our work, we only used the natural language processing tool to help us do the labeling job combining with our manual operations.

After we analyzed the descriptor sentences, their constituents are labeled with 4 semantic roles which have special importance and practicability for our situation: (1) subject- who perform the work? (2) verb- what is the action performed? (3) object- on what object the action is performed? (4) complement- How is it performed or what is the aim that it is performed? Table 3.2 shows examples of the behavior vectors we got from the descriptor sentences.

<table>
<thead>
<tr>
<th>NO</th>
<th>Subject</th>
<th>Verb</th>
<th>Object</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>virus removal application</td>
<td>integrate</td>
<td>real-time safeguard</td>
<td>against threats</td>
</tr>
<tr>
<td>2</td>
<td>antivirus software</td>
<td>detect clean</td>
<td>viruses</td>
<td>without slowing PC</td>
</tr>
<tr>
<td>3</td>
<td>software utility</td>
<td>provide</td>
<td>user</td>
<td>against antivirus antispyware</td>
</tr>
<tr>
<td>4</td>
<td>internet security solution</td>
<td>detect block</td>
<td>viruses</td>
<td>with FreeAntivirus firewall</td>
</tr>
</tbody>
</table>

Table 3.2: Examples of behaviors

We created the behavior vectors for each descriptor then stored them into a Excel document in which every row represents a behavior vector and the comma is used to separate the different components in a behavior vector. After storing all the behavior vectors into the document, in our application, we used the stream input method to read all the rows which represent the behavior vectors into our application. We build a class named Vector to store all the retrieved behavior vectors and use the instances of this class to store all the descriptors. Code listing 3.1 shows the process of inputting the behavioral vectors into our application.
3.3 Similarity Calculation

The approaches for measuring the similarity between words, sentences, paragraphs and documents is an important consideration for us to do various nowadays tasks such as information retrieval, document clustering, word-sense disambiguation, automatic essay scoring, short answer grading, machine translation and text summarization. In our work, we need to use the similarities between the different behavior vectors which represent the software product descriptors.

After we preprocess all the descriptions and get the available behavioral vectors representing every descriptors, we need to calculate the similarities between all the behavior vectors and build the similarity matrix used for later feature model generation. There are several steps we need to do from words similarity measure to final behavioral vectors similarity calculation. Figure 3.2 shows the whole similarity calculation process in our work.

According to the process shown in figure 3.2, firstly, we need to calculate the word similarity, and then we need to compute the similarity of each pair of components (subject, verb, object, complement) in the behavioral vectors through seeming each component as one text segments and doing the text segments similarity calculation. Finally we try to get the similarities between behavior vectors considering the similarities between the each pair of the same components respectively in two behavioral vectors.

For words similarity measure part, after comparing and analyzing among several candidates methods, we decided to use one of the semantic similarity measure approaches named knowledge-based similarity measure as primary method to compute the similarity between different words and the string-based similarity measure as the supplement. The process we compared and analyzed different approaches for words similarity measure and the reason why finally we chose the knowledge-based and string-based approaches for words similarity calculation will be explained in section 3.3.1. As for text segments similarity calculation part, we consulted a special method contained in a research paper [MCS+06] which introduces a method for text segments similarity measure also considering the weights of the words in the text segments for more accurate text segments similarity calculation. And considering our own special need of more semantic aspects’ considerations for similarity calculation, we improved the method for weight measure in the process of text segments similarity calculation. In section 3.3.2, we will talk about the details of our improved text segments similarity calculation process. And we referred to a method mentioned

---

//Input the behavior vectors into application

```java
Input: document containing descriptors
for i=1 to NumberOfRows do
    Create a vector instance v
    v.subject=rowsOfDocument[i].subject
    v.object=rowsOfDocument[i].object
    v.verb=rowsOfDocument[i].verb
    v.complement=rowsOfDocument[i].complement
end
return a list of instances of vectors containing all descriptors
```

Listing 3.1: Process of inputting the behavioral vectors
in another research paper [IRB14a] for behavioral vectors similarity calculation considering the similarity between each pair of the same components respectively in two behavioral vectors. But as for us, we used the different components partition method considering the semantic structure of the descriptors to split the descriptors and form the behavioral vectors for later behavioral vectors similarity calculation. And we also did not consider the complicated ontological information to simplify our whole similarity calculation process. In section 3.3.3, the details of our own behavioral vectors similarity calculation process will be introduced.

![Figure 3.2: Process of descriptors similarity calculation](image)

### 3.3.1 Word Similarity Measure

(1). **Different Approaches for Word Similarity Measure**

Firstly we need to decide the method we can use for words similarity calculation. We have read plenty of paper about words similarity calculation approaches. How to find the similarity between the different words is the most fundamental part of work for the text similarity which is used later as a primary stage for sentence or paragraph similarity measure.

We have read plenty of paper about words similarity calculation approaches, finally we summarized three main approaches: String-based, Corpus-based and knowledge-based for words similarity measure. They are all used to measure the similarity between two words according to one of the two rules one of which is lexically similar and another one is semantically similar. As the name we can know the meanings of them, the two words are lexically similar if they have very similar character sequences and the two words are semantically similar if they have very similar meanings or relatively closed meanings. String-based similarity measure belongs to the Lexically similar measure. Corpus-based similarity and knowledge-based similarity approaches belong to the semantically similar measure. Corpus-Based similarity is a semantic similarity measure that determines the similarity between words according to information gained from large corpora. Knowledge-Based similarity is a semantic similarity measure that determines the degree of similarity between words using information derived from semantic networks [GF13]. The figure 3.3 shows the basic architecture of the different types of the words similarity approaches and their representative algorithms.
3.3. Similarity Calculation

**Figure 3.3:** Basic architecture of the different types of the text similarity approaches

**a) String-based Similarity:**

For the string-based similarity measures, they focus on string sequences and character compositions. We use the string metrics that measure the similarity and dissimilarity (corresponding to the distance between two objects) between two text strings for estimating string matching results or comparing them. As shown in Figure 3.4, there are a set of representative and popular string similarity measures which have been implemented in SimMetrics package.

**Figure 3.4:** Basic architecture of the different types of the string-based similarity approaches

**b) Corpus-based Similarity:**

Corpus-based similarity is one of the semantic similarity measures which comes up with the similarity between the words according to the information contained in
3. Design of Feature Model Generation Process

A corpus is a very large set of written or spoken texts which is used for language analysis. There are a lot of corpus-based measures came up and mentioned in research paper [GF13]. The figure 3.5 shows a set of representative corpus-based measures and their basic architecture.

Figure 3.5: Basic architecture of the different types of the corpus-based similarity approaches

c) Knowledge-based Similarity

Knowledge-based similarity measure is one of the semantic similarity measures that is based on comparing the similar degree of the words’ meaning derived from semantic network. WordNet is one of the most popular large dictionaries for English in the area of measure the knowledge-based similarity between the words. Words which belong to nouns, verbs, adjective or adverbs are grouped together according to the cognitive synonyms(synsets). And each of them represents a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations.

As shown in figure 3.6, there are two main types in knowledge-based similarity measures. One is measure of semantic similarity and the other one is measure of semantic relatedness. Semantically similar concepts are deemed to be related on the basis of their likeness. As for the semantic relatedness, on the other hand, is a more general notion of relatedness [GF13], not specially tied to the shape or form of the concept. In other word, we can say that Semantic similarity is measured by how the two words are related, it covers a broader range of relationships between concepts of the words that contains extra similarity relations such as is-a-kind-of, is-a-specific-example-of, is-a-part-of, is-the-opposite-of.

We can use both of the two main types mentioned above to calculate the similarity between two words according to the different methods used behind them.
(2). Comparison of the Different Word Similarity Calculation Approaches

For the string-based similarity measure, it provides the function of comparing string sequences and character composition to measure the similarity degree between two words. This method will face the problem that without considering the semantic meaning of the words, the accuracy of the estimated similarity values can not be promised considering some exceptional cases. For example, when we only compare the string sequences and character composition for the words fridge and fringe. According to our knowledge, we all know that these two nouns have the totally different meaning. We used one of the string-based measures named Levenstein measure and one of the semantic measure named Lin measure to compare the similarity results based on these same sample words. Table 3.3 shows the similarity values calculated based on some samples of words.

<table>
<thead>
<tr>
<th>Words</th>
<th>Lin Measure</th>
<th>Levenstein measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>fridge VS fringe</td>
<td>0.12</td>
<td>0.83</td>
</tr>
<tr>
<td>antivirus VS virus</td>
<td>0.56</td>
<td>0</td>
</tr>
<tr>
<td>feed VS friend</td>
<td>0.12</td>
<td>0.5</td>
</tr>
<tr>
<td>antivirus VS virus</td>
<td>0.56</td>
<td>0</td>
</tr>
<tr>
<td>internet VS ethernet</td>
<td>0</td>
<td>0.625</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of different words similarity measure

We can easily see from the table 3.3 that string-based similarity measure sometimes can not deal with the situation that two words have very similar character sequence while they have totally different meanings. But this measure can also become very helpful for some very special words like antivirus, internet in some very professional
fields. Because of the limitation of word resources when we try to compute semantic similarity measure, the measures in knowledge-based or corpus-based can not recognize some specific words.

(3). Word Similarity Calculation Approach Choosing

In our work, after previous comparison and analysis of different word similarity calculation approaches, we decided to use both of the semantic measure and string-based semantic measure for measuring words similarity. In another word, we use semantic similarity measure as the main similarity calculation measure and use string-based measure as the supplement. For the string-based measure, we choose a measure named Levenstein measure which uses one two-dimension array to record if every string is same. As for the semantic measure, there are plenty of choices for us, we decided to use one of the knowledge-based measures named Lin measure [Lin98]. We chose knowledge-based measure instead of corpus-based measure because that for the corpus-based measure, the relative prediction of the similarity value is very hard to get and it is very hard to find a sufficient corpus to support the similarity measure. And as for knowledge-based measure, we can easily use a word network named WordNet which is a large lexical database in which nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concepts. And we can easily use java wordnet interface (JWI) as the interface for WordNet to use it on Java development platform to develop our own process. As for the implementation of Lin measure for words similarity calculation, we used jar file named JWS (Java WordNet Similarity) provided by University of Sussex. Based on these tools and techniques, we implemented our word similarity measure program based on Eclipse development platform. As shown in code listing 3.2, it is the process that we implement our similarity measure function.

```
1 //Words Similarity Measuring
2 Input: word1 and word2
3 int similarity=-1;
4 Similarity=LinMeasure(word1,word2)
5 if similarity<0
6 {
7    Similarity=LevensteinMeasure(word1,word2)
8 }
9 return similarity value
```

Listing 3.2: Process of implementing similarity measure function

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), and each of them expresses a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet’s structure makes it a useful tool for computational linguistics and natural language processing.

For our case, we use the jar file JWS (Java WordNet Similarity) to do the word similarity measure based on WordNet and using java to implement the approaches which we use to measure the words similarity based on Eclipse platform though
3.3. Similarity Calculation

Java WordNet Interface. After we finish the word similarity calculation process, the measures like Mihalcea, Coreley and Strapparava (MCS) would be used for phrases similarity computation.

3.3.2 Text Segments Similarity Measure

As mentioned above, the measures which are used to calculate the similarities between the words or concepts have been already determined. Then in this part we will talk about measuring the similarity between two text segments which consist several words in order to calculate the similarity values between the components in behavioral vectors, and there are much less suitable ways.

As for us, we came up with the process that is able to automatically compute a score which can reflect the similar degree of two given text segments referred to the previous research work [MCS+06] which are much better compared with the traditionally used method in which using simple lexical matching. After getting the similarity values of the words, we also need to consider the specificity of each word in order to measure the importance of the word in text segments for text segment similarity calculation (how important the word is in one text segment). So that we can give a higher value to the words that have meaningful information available to semantic text segments similarity calculation (e.g. In antivirus software product, words like scan, trojan are very important). In contrast, we can also set a much lower value for the weights of the words which cannot help much for our text segments similarity measure work (e.g. For some words which are very common and have no real meanings, we can set a very small value which near to 0 or directly remove or replace it from descriptors directly in pre-processing step). We combined metrics of word-to-word similarity and word specificity into one formula used for calculating the semantic similarity of two text segments.

Some methods using inverse document frequency (idf) or term frequency inverse document frequency (tf-idf) for term weighting were introduced by other previous research [MCS+06]. Inverse document frequency (idf) and term frequency inverse document frequency (tf-idf) is defined based on the theory that if one word only appears in very few documents, we can more easily use it to search our aim and its weight should be higher than other words which frequently appears in many documents. But in our situation of facing software products as objective objects and using antivirus software products as the example analysis domain in our research work, many special words having closed relations to the potential features always appear frequently in the documents to represent the functionalities owned by the software and we also need to use these words to form the frequent item sets to name the features after building the resultant feature model tree. So the basic idea behind using the inverse document frequency or frequency are not completely suitable for our situation.

According to the situation mentioned above, we chose the way to use the concepts or meanings of the words to decide how close and representative for them to their descriptions to decide the weights of them. In another word, we analyze the importance of the words in the descriptors to give them different weights. As shown in figure 3.7, we classify the different words into one of the different 4 levels and each level corresponds to different weights. The words in first level are the core words which can exactly represent the functionalities of our analysis domain. In our thesis,
we use antivirus software as the the example. So the words in first level are some words such as spyware, trojan. And in second level, there are some other words in software engineering field except the words in first level. As for the third level, there are some words which have less meanings for our work such as some verbs or nouns. The words in forth level are the stop words which nearly have no use for feature extraction. We give the weights with the values 1, 0.65, 0.35, 0 to different levels from first level to forth level. We created different levels’ dictionary to store the words belonging to the different level in order to assign the different weights values to the different words. We create the core words dictionary through retrieving words from wikipedia about antivirus software field for first level. We searched the key words "antivirus software" in wikipedia and manually select all the words in the search result page which are related to our sample work analysis domain of antivirus software field. And we collected the words from the relative website\(^3\) in which the words has been grouped and classified considering the different fields such as software engineering they belong to for second level about software engineering field. In previous work, we have set the stop word list, so we can use it to group the words in stop list which have not been removed in pre-processing step as the dictionary for forth level. The words do not belong to these three level will be seemed as third level. For example, for a given feature descriptor in one document: “prevent possible intrusion or attacks by spyware trying to enter your computer”, the words prevent and computer have lower weights than the word spyware which is more relevant to the descriptions to specifically represent which threat we try to avoid in antivirus software field. Table 3.4 shows an example list of the weights for the different words appearing in the documents.

\(^3\)https://techterms.com/category/software

Figure 3.7: General process for feature naming
After coming up with a metric for word-to-word similarity and a method to measure the specificity of each word appearing in the descriptors, we came up with a method used to compute the semantic similarity between two text segments $T_1$ and $T_2$ through considering the semantic words similarities in turn with respect to the other text segments. There are two steps in total for text segments similarity calculation between text segments $T_1$ and $T_2$.

1). For each word $w$ in the segment $T_1$, we try to find the word in the segment $T_2$ which has the highest word similarity value to word $w$ ($\text{maxSim}(w, T_2)$) through traversing all the the words in $T_2$ to compute the words similarity value between word $w$ and each word in $T_2$ according to the word similarity measure which we have decided to use.

2). Next step, the same process will be implemented to find the most similar word in $T_1$ for every word in $T_2$.

After these two steps, the maximal word similarities found in the two steps then will be weighted with corresponding word specificity, summed up, and normalized with the length of each text segment. Finally the immediate similarity value will be divided by two to combine it with simple average to get the final resultant similarity value for text segments $T_1$ and $T_2$.

Be careful that in our text segments similarity calculation process, basic words can be considered in the similarity calculation process using Lin measure based on WordNet, and all other special words in our analysis domain which are too professional or infrequent to be processed based on WordNet and not collected in WordNet such as words "antivirus" and "malware" should be considered using one string-based word similarity measure named Levenstein.

The semantic similarity between two text segments $T_1$ and $T_2$ can be determined by following scoring function:

$$sim(T_1, T_2) = \frac{1}{2} \times \left( \frac{\sum_{w \in T_1} \text{maxSim}(w, T_2) \times \text{weight}(w)}{\sum_{w \in T_1} \text{weight}(w)} + \frac{\sum_{w \in T_2} \text{maxSim}(w, T_1) \times \text{weight}(w)}{\sum_{w \in T_2} \text{weight}(w)} \right)$$

(3.1)

- Weight of word $\text{weight}(w)$ is the specificity of word $w$. 

<table>
<thead>
<tr>
<th>NO</th>
<th>Word</th>
<th>Weight</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>attachment</td>
<td>0.65</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>analysis</td>
<td>0.35</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>provide</td>
<td>0.0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>traverse</td>
<td>0.35</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>browse</td>
<td>0.65</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.4: Different weights for some sample words
- $maxSim(w, T_1)$ is the maximal similarity value between word $w$ and the words in text segment $T_1$. Similarity values between the words are calculated by Lin measure and if this knowledge-based measure cannot work for some special words, one of the string-based word similarity measure named Levenstein measure will be used.

- $maxSim(w, T_2)$ is the maximal similarity value between word $w$ and the words in text segment $T_2$. Similarity values between the words are calculated by Lin measure and if this knowledge-based measure cannot work for some special words, one of the string-based word similarity measure named Levenstein measure will be used.

This segments similarity measure function has a value ranging from 0 to 1. A maximal score value of 1 means these two text segments are identical and a score of 0 indicates that there is no overlapping between these two text segments.

Note that getting the maximal similarity value of 1 is considered only under the condition that it is within classes of words which belong to the same part-of-speech. The reason why this situation happens is that the majority of the knowledge-based similarity measures used between words cannot be applied across parts-of-speech, and therefore, for the goal of keeping the consistency, the restriction named "same word-class" is imposed to all the word-to-word similarity measures. For example, the word with the maximal similarity value of the noun word "friend" in the sentence "antivirus software provide a friendly interface for user interaction" will be found among the nouns "software" and "interface" and the words belonging to other class such as adjective and verb with different part-of-speech will be ignored(e.g.friendly). So in order to solve this case that a word-to-word semantic similarity measure cannot be used for some parts-of-speech (and also some knowledge-based measures are not defined among adjectives), we can use a string-based similarity measure as the supplement of knowledge-based similarity measure for these cases. In our work, we used a string-based similarity measure named Levenstein measure as a supplement.

### 3.3.3 Behavioral Vectors Similarity Measure

In our work for descriptors semantic similarity measure, we also considered the syntactical structure of the descriptors. So in preprocessing step, we tried to transfer all the descriptors into the behavioral vector formation. Then later we will consider about the similarity between each pair of descriptors through comparing different components in behavioral vectors respectively.

As mentioned before, for behavioral vectors similarity calculation, it is based on the calculation of the similarity of each pair of components (subject, verb, object and complement) and combination of the similarity values of different components to get the overall weighted average behavioral vectors similarity. So firstly we use the text segments similarity measure technique to calculate the components’ similarities, then we combine the components’ similarities and perspective profiles which represent the stakeholders’ preferences and needs to the different components and give the values to the weights of different components for measuring to get the similarity values of the behavioral vectors which is formally expressed as a weighted average of
3.3. Similarity Calculation

the corresponding behavioral vector’s components. The formula is expressed below:

\[
\sum_{\text{comp} \in \{\text{Subject, Verb, Object, Complement}\}} \sigma_{\text{comp}} \cdot W_{\text{comp}} \cdot \text{sim}(v_1.\text{comp}, v_2.\text{comp}) \\
\sum_{\text{comp} \in \{\text{Subject, Verb, Object, Complement}\}} \sigma \cdot W_{\text{comp}}
\] (3.2)

This is the formula used for weighted average behavioral vectors similarity calculation in which:

- Description \( W_{\text{comp}} \) is the weight given to a specific behavioral vectors’ component(subject, verb, object or complement) according to the preferences and needs of stakeholders and all the components’ should obey the constraint:

\[
\sum_{\text{comp} \in \{\text{Subject, Verb, Object, Complement}\}} W_{\text{comp}} = 1 \] (3.3)

- \( \sigma_{\text{comp}} \) means if the component \( \text{comp} \) has content in both of the behavioral vectors. If \( \sigma_{\text{comp}} = 1 \), it means that both of the behavioral vectors are not empty for component \( \text{comp} \), otherwise, there is at least one behavioral vector whose component \( \text{comp} \) is empty.

- \( \text{sim}(v_1.\text{comp}, v_2.\text{comp}) \) means the text segments similarity value between the same components of two behavioral vectors.

As mentioned before, \( W_{\text{comp}} \) is decided by stakeholders according to their preference and needs, so different \( W_{\text{comp}} \) distributions to the components will also influence the vectors similarity calculation result and the final feature model construction. The example below shows some details about the influence of the different weight distributions for components. There are two vectors presented which are captured from website in the example analysis domain antivirus software and have been preprocessed shown in table 3.5.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Verb</th>
<th>Object</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>antivirus software</td>
<td>delivery</td>
<td>file antispam solution</td>
<td>file scan</td>
</tr>
<tr>
<td>security tool</td>
<td>erase</td>
<td>file content</td>
<td>file traverse</td>
</tr>
</tbody>
</table>

Table 3.5: Samples of splitting descriptors into behavioral vectors

We can easily find from table 3.5 that for these two vectors, their object and complement components has more similar meanings which means that when considering their different components similarities, the component similarity values of object and complement parts are higher than those of subject and verb parts. So how to distribute different weight values to different components will leads to different similarity results. If we think that the type of tools who did this behave and their specific action are more important, we can set higher weights to subject and verb parts and on the contrary, if we think it is more important to know who has been
done the operations on and the goal and effect of our operations, we can set higher value to the object and complement parts. Given two examples in table 3.5, if we set higher weights to object and complement parts, we will get higher similarity values between these two vector, in contrast, if we set higher weights to subject and verb parts, we will get relatively lower similarity value between two given vectors. That is all depends on the real situations that we can choose the different components weights distributions.

In our work, we tried to analyze and simulate stakeholder’s logical thinking ways and try to find the most suitable ways to distribute the weights to different components. It is also an optimized problem considering different situations to set the most suitable values for different weights. In our work, finally we set higher weight values to objective and complement components considering that the most useful information is contained in these two parts.

After we decide the behavior vectors similarity measuring method, we need to calculate all the similarities between each pair of vectors and store them into a similarity matrix which is used for measuring the distance between two instances (behavior vectors) for later implementation of clustering algorithm to achieve the primitive feature model construction. In our work, we used the formula below to represent distance using similarity between two vectors $v_1$ and $v_2$.

$$distance(v_1, v_2) = 1 - sim(v_1, v_2)$$

As shown in formula above, distance can be seemed as the dissimilarity between two behavior vectors.

Listing 3.3 shows the implementation details in our work to do the previous mentioned process for vectors similarity matrix generation.

```java
1 Input: set of behavioral vectors DescriptorSet
2 double [][] SimilarityMatrix; // used to store the similarities
3 for(int i=0; i<DescriptorSet.size(); i++)
4 {
5     for(int j=i; i<DescriptorSet.size(); j++)
6         {
7             double s=similarity.SimilarityMeasureOfRow (RowSet[i],RowSet[j]);
8             DistanceMatrix[i][j]=s;
9             DistanceMatrix[j][i]=1-d;
10         }
11 }
12 return distance matrix
```

Listing 3.3: Vector similarity matrix generation

### 3.4 Feature Model Generation

In third stage, after getting the similarity matrix for all pairs of the behavioral vectors, a feature model represented by a hierarchical structure of features and extracted from set of textual software production descriptors is automatically generated. For
this part, we not only use the natural language processing technique to analyze
the content in texts but also data mining knowledge to extract useful information.
There are three steps in total to get the final resultant feature models as shown
in Figure 3.8. And we will discuss each steps clearly later.

![Figure 3.8: Process for feature model generation](image)

(1) **Primitive Feature Model Structure Generation**

To get the primitive feature structure, in our work, firstly we used the algorithm
named hierarchical agglomerative clustering algorithm which tries to build a hierar-
chical clustering tree from a set of descriptors. This method places every descriptors
into a independent and separated clusters, then iteratively merges pairs of clusters
whose distance are the lowest. The final output result of this algorithm is a binary
hierarchical clustering trees containing clusters of different levels. Secondly we will
implement the flattening and simplifying process to simplify our hierarchical tree by
merging subnodes to their parent nodes whose similarity is close and can be seemed
as one single node. To achieve this goal, we set a predefined similarity threshold to
define the similarity difference between two clusters that we can receive otherwise
we will see that two clusters as one.

(2) **Relationship Detection**

After constructing and flattening step to get the primitive feature model tree, we
need to analyze and indicate the relationships between the parent nodes and child
nodes which represents the potential different levels features in the primitive feature
model. There are four main relationships Mandatory, Optional, OR and XOR we
considered. In our work, these relationships are used to examine the appearances
of every descriptors in input documents (each product as the sample for our feature
extraction process corresponds to one document containing the descriptions of the
product used for our feature extraction process) and there are some concepts that
are used to define these relationship types that with the condition that there are at
least two descriptors that are gathered under the same parent node appearing in the
same input document, then the corresponding child nodes are OR-grouped. And
If all the descriptors stemming from totally different input documents are clustered
under the same parent node, we consider the corresponding child nodes as XOR-
grouped and the node can be Mandatory, if it includes descriptors from all input
documents, otherwise it should be optional.

And in our work, we came up with idea that it is possible to take use of the similarity
measure to deduce the OR and XOR relationship types that if two subnodes are
very similar, we can see them as XOR relationship, and in contrast, if the difference between two subnodes is very large, we will see them as OR relationship. As for the deduction of the mandatory and optional relationships, we used the definitions of them which have been mentioned above to do the deduction. We traversed all the descriptors in every node and check if there is at least one descriptor for every document we extracted from the website that exist in the node.

(3) Naming the Feature

In order to create a easily understandable feature model to the users, the nodes with clusters in the hierarchical structure which represent the different levels’ features should be given a meaningful name which can represent the meanings of descriptors it contains and the its corresponding functionality. Based on our basic demands and needs, we developed a process for nodes naming containing selection of the frequent itemsets that a set of descriptors contains in the cluster of the node. Then we try to identify the suitable name for the nodes with our extracted frequent itemsets. Our approach is similar to the method presented in [HL04] for summarizing customers reviews.

There are 3 steps in total to get final extracted name for the node from its set of relative descriptors contained in the corresponding cluster. Figure 3.9 shows the steps we should follow to achieve the goal.

Figure 3.9: General process for feature naming

We did these three steps: descriptors pruning, frequent itemset finding and name selection respectively and we will discuss these three steps in details later.

(1) Descriptors pruning: To easily access the descriptors and get the frequent item sets in next step, firstly we need to use natural language processing tool named Stanford Part-of-Speech Tagger to prune the descriptors and retain only useful nouns, adjectives and verbs which can usefully describe a feature.

(2) Frequent itemsets finding: Try to find all the frequent itemsets from the set of descriptors in the cluster with a given support value.

(3) Selecting name for a cluster: To select the name for a cluster, firstly all of its frequent itemsets of maximal size need to be found and based on them, we can find the suitable name for every node.

3.4.1 Primitive Feature Model Structure Construction

For this part, our aim is to form the primitive feature model structure. Firstly, we have to get the hierarchical tree structure formed by a set of descriptors as the
rudiment of the primitive feature model and then try to flatten and simplify it to get the suitable primitive feature model structure. In another word, we formed the hierarchical tree structure based on tightly-knit clusters of a set of relative descriptors then try to process it to get the primitive feature model structure. In order to firstly form the hierarchical tree structure, we need to choose a suitable algorithm for traversing and finding the relative descriptors to form the hierarchical tree structure in our work, after the comparison and analysis, we chose hierarchical agglomerative clustering algorithm (HAC) to do this job among a set of candidates algorithms found during we traversed previous related research works with the identified proposal that used other than natural language processing tools to extract features from the textual descriptions or requirements. The approaches mentioned in previous research works include a various set of algorithms especially the clustering algorithms, such as Fuzzy K-means, Association Rule Mining, Hierarchical Clustering and so on. Later we will introduce some details of some typical candidates algorithms, compare among all of them considering their advantages and disadvantages, state the reasons why we chose hierarchical agglomerative clustering algorithm, represent the hierarchical tree structure generation process we have implemented, how the tree structure flattening and simplifying process works and the relationships detection, feature naming process.

Next part we will introduce several typical candidate feature extraction algorithms mentioned in previous research work \([T^*06]\) and compare them with each other about their advantages and disadvantages.

1). Different Typical Candidate Primitive Hierarchy Structure Construction Algorithms

In this part, we will introduce several typical candidate feature extraction algorithms mentioned in \([T^*06]\) which are often used for primitive hierarchy structure construction and compare them with each other considering their advantages and disadvantages.

I. Hierarchical Clustering Algorithm:

In data mining and statistics field, hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of clusters. Figure 3.10 on the following page shows an example of a dendrogram which is returned as a result of hierarchical clustering algorithm. For data mining situation, You run a horizontal cut on this tree to get a set of clusters. But for our work, after we got the hierarchical tree, we tried to flatten and simplify it and get a suitable feature tree to represent the feature models in which nodes with clusters of descriptors represent different levels’ feature.
3. Design of Feature Model Generation Process

Figure 3.10: A hierarchical clustering result of four points shown as a dendrogram and as nested clusters

(1). Strategies for hierarchical clustering

Strategies for hierarchical clustering generally fall into two types. One is Agglomerative Clustering and another one is Divisive Clustering.

a). Agglomerative

This is a "bottom-to-up" approach in which each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. Listing 3.4 shows the bottom-to-up agglomerative clustering process.

Given a set of data points and starting with the bottom layer, where each data point is a cluster of its own:

```plaintext
if necessary
compute the proximity matrix
end
repeat:
merge the closest two clusters
update the proximity matrix to reflect the proximity between
the new cluster and the original cluster.
until only one cluster remains.
```

Listing 3.4: Basic agglomerative hierarchical clustering process

b). Divisive

This is a "top-to-down" approach in which all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy. Listing 3.5 shows the top-to-bottom divisive clustering process.

Starting with a single cluster that contains all data points:

```plaintext
repeat
split one cluster into two
until k clusters remain
```

Listing 3.5: Top-down: basic divisive clustering process

Comparing these two different types of the hierarchical algorithm, hierarchical agglomerative clustering is more frequently used in related research works than top-down clustering and also for our work, the hierarchical agglomerative clustering
3.4. Feature Model Generation

Algorithm is more suitable to deal with our problem that in our situation we try to process a set of descriptors to build a hierarchical clustering tree in order to extract features from them. So a Bottom-up algorithm which treats each single sample as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all single instances is a better choose for our work and more suitable for our situation to achieve our goal. So we choose hierarchical agglomerative clustering instead of hierarchical divisive clustering when we considered to use hierarchical clustering algorithm.

(2). Introduction to Typical Distance Measures Used in Hierarchical Agglomerative Clustering

When we try to use the hierarchical agglomerative clustering algorithm, we also need to consider the distance measure used for choosing pairs of clusters for each merging step. Agglomerative clustering algorithm’s results differ from their definitions of proximity which means the different distance measures between two clusters. There are several types of measurements for distance.

- MIN(single link)
- MAX(complete link)
- Group average
- Distance between centroids
- Ward’s method
- Squared error

In our thesis, we will intensively introduce several typical distances measure we can use, the advantages and disadvantages of them and the influence if we choose different distances measure given an example dataset shown in table 3.6.

<table>
<thead>
<tr>
<th>Points</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.00</td>
<td>0.24</td>
<td>0.22</td>
<td>0.37</td>
<td>0.34</td>
<td>0.23</td>
</tr>
<tr>
<td>P2</td>
<td>0.24</td>
<td>0.00</td>
<td>0.15</td>
<td>0.20</td>
<td>0.14</td>
<td>0.25</td>
</tr>
<tr>
<td>P3</td>
<td>0.22</td>
<td>0.15</td>
<td>0.00</td>
<td>0.15</td>
<td>0.28</td>
<td>0.11</td>
</tr>
<tr>
<td>P4</td>
<td>0.37</td>
<td>0.20</td>
<td>0.15</td>
<td>0.00</td>
<td>0.29</td>
<td>0.22</td>
</tr>
<tr>
<td>P5</td>
<td>0.34</td>
<td>0.14</td>
<td>0.28</td>
<td>0.29</td>
<td>0.00</td>
<td>0.39</td>
</tr>
<tr>
<td>P6</td>
<td>0.23</td>
<td>0.25</td>
<td>0.11</td>
<td>0.22</td>
<td>0.39</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 3.6: Euclidean distances matrix for 6 points
a). MIN Distance Measure

For MIN distance measure, it tries to find the minimal distance of two instances between two clusters $X$ and $Y$.

```plaintext
for each $x$ in $X$
    for each $y$ in $Y$
        compute $\text{dist}(x,y)$
    end
end
sort the distance values into a list $L$
return $\text{MinValue}(L)$
```

Listing 3.6: Min distance measure between cluster $X$ and $Y$

The distance value between two clusters $X$ and $Y$ is the minimum value in $L$:

$$distance(X,Y) = \min_{X,Y} L_{X,Y}$$  \hspace{1cm} (3.5)

Figure 3.11 shows the hierarchical agglomerative clustering result using MIN distance measure with a given dataset of 6 points.

![Figure 3.11: Single link clustering result of the six points given in table 3.6](image)

b). Max Distance Measure

For MAX distance measure, it tries to find the maximal distance between two instances respectively in two clusters as the distance between two clusters. Listing 3.7 shows the process for measuring minimal distance between two clusters $X$ and $Y$.

```plaintext
for each $x$ in $X$
    for each $y$ in $Y$
        compute $\text{dist}(x;y)$
    end
end
sort the distance values into a list $L$
return $\text{MaxValue}(L)$
```

Listing 3.7: Top-down: basic divisive clustering process

The distance between $X$ and $Y$ is the maximum value in $L_{X,Y}$:

$$distance(X,Y) = \max_{X,Y} L_{X,Y}$$  \hspace{1cm} (3.6)
Figure 3.12 shows the result of hierarchical agglomerative clustering algorithm using MAX distance measure with a given dataset of 6 points in table 3.6.

![Figure 3.12: Complete link clustering of the six points given in table 3.6](image)

c). **Group Average Distance Measure**

For Group average distance measure, it tries to find the average distance among all pairs of two instances respectively in two clusters. Listing 3.8 shows the process for measuring average distance between two clusters X and Y.

```plaintext
for each x in X
  for each y in Y
    compute dist(x;y)
  end
end
store all the distance values into a list L
return AverageValue(L)
```

Listing 3.8: Group average distance between cluster X and Y

The group average distance between clusters X and Y is the average value in $L_{X,Y}$:

$$
distance(X,Y) = \frac{\sum_{x \in X} \sum_{y \in Y} dist(X,Y)}{|X| \times |Y|} \quad (3.7)
$$

Figure 3.13 shows the result of hierarchical agglomerative clustering algorithm using group average distance measure with a given dataset of 6 points in table 3.6.

![Figure 3.13: Group average clustering of the six points given in table 3.6](image)
36  

3. Design of Feature Model Generation Process

d). Ward’s Distance Measure

For Ward’s distance measure, it calculate the distance between two clusters based on the increase in the sum of square errors incurred when merging the clusters. Listing 3.9 shows the process for measuring Ward’s distance between two clusters X and Y.

```
1 compute SSE(X)
2 compute SSE(Y)
3 compute SSE(X+Y)
4 distance=SSE(X+Y)−SSE(X)−SSE(Y)
5 return distance
```

Listing 3.9: Ward’s distance between cluster X and Y

For a clustering over a set D, computed with a distance function d() (the distance values are normalized), the formula of Sum of Square Errors (SSE) is shown below:

\[
SSE(X) = \sum_{x \in X} dist(x, center(X))^2
\]

Figure 3.14 shows the result of hierarchical agglomerative clustering algorithm using Ward’s distance measure with a given dataset of 6 points in table 3.6.

Figure 3.14: Ward’s distance measure of the six points given in table 3.6

(3). Comparison of Different Distance Measures

For different distance measures, because of their different criteria for distances calculation process, there are different effects which will influence our final result’s accuracy and the gap between our real and expected results [BS13]. So we need to consider their respective advantages and disadvantages to have a better understand to them and choose the most suitable one for us in order to get the result with highest quality and closest to our expected result.

a). MIN Distance Measure

The advantage of MIN is that it can discover non-spherical and non-equal-size clusters. Figure 3.15 on the next page shows the clustering result given two non-equal-size and non-spherical clusters.
3.4. Feature Model Generation

The disadvantage of MIN is that it is very sensitive to the outliers and noises, if there are some outliers existing between clusters, this measure cannot get out of the influence of them and clearly distinguish two clusters. Figure 3.16 shows the clustering result given two non-equal-size and non-spherical clusters with outliers between them.

Figure 3.15: Clustering result using MIN distance measure given two non-equal-size and non-spherical clusters

Figure 3.16: Clustering result using MIN distance measure given two clusters with outliers between them

b). MAX Distance Measure

The advantage of MAX distance measure is that it can get out of the influence of outliers and noises existing between clusters. Figure 3.17 on the next page shows the clustering result given two clusters with outliers between them.
3. Design of Feature Model Generation Process

Figure 3.17: Clustering result using MAX distance measure given two clusters with outliers between them

The disadvantage of MAX distance measure is that it only can find equal-size clusters and favour spherical clusters. Figure 3.18 shows the clustering result given two non-equal-size clusters.

Figure 3.18: Clustering result using MAX distance given two non-equal-size clusters

c). Ward’s Distance Measure

The advantage of Ward’s distance measure is that it can get out of the influence of outliers and noises existing between clusters as shown in Figure 3.19 on the next page.
3.4. Feature Model Generation

The disadvantages of Ward’s distance measure is that it only can find spherical clusters. Figure 3.20 shows the clustering result using Ward’s distance measure given two non-spherical clusters.

Figure 3.20: Clustering result using Ward’s distance measure given two spherical clusters

d). Group Average Distance Measure

The advantage of Group Average distance measure is that it can be used for non-spherical and non-equal clusters as shown in Figure 3.21 on the next page.
3. Design of Feature Model Generation Process

Figure 3.21: Clustering result using Group Average distance measure given two non-spherical and non-equal clusters

The disadvantages of Group Average distance measure is that it cannot deal with the noisy data around clusters. Figure 3.22 shows the clustering result using Group Average distance measure given two clusters with noisy data between them.

Figure 3.22: Clustering result using Ward’s distance measure given two spherical clusters

In our work, we tried to implement the most typical distance measures MAX, MIN and Group Average to compare the results of them given a set sample data to choose the most suitable one which can work well in our work. The comparison details will be shown in next chapter of implementing hierarchical clustering for primitive feature structure generation.
II. K-Means Algorithm:

K-means is one of the typical clustering algorithms with objective function based on prototype. It is the algorithm which uses the distance between data point and the prototype as the optimization objective and obey the rule that adapt the $k$ center points of $k$ clusters to achieve minimal sum of distances in the clusters. In k-means clustering algorithm, we use the distance as the dissimilarity metric and euclidean distance measure is commonly chosen in the algorithm to measure the distance. In another word, K-means is good for a quick-and-dirty attempt to find groups in the data. The initialization of $k$ center points will influence the final clustering result heavily. Listing 3.10 shows the process for implementing k-means clustering algorithm.

Listing 3.10: K-means clustering algorithm

| INPUT: a set of data points D in feature space F; |
| the number of clusters K |
| Select K initial centroids |
| Repeat |
| Assign each point to the centroid closest to it |
| Recompute the positions of the K centroids |
| Until the positions of the centroids not change anymore |
| OUTPUT: K clusters as clustering result |

K-means is the algorithm of which the aim is to optimize its objective function. The formula of k-means optimization criteria is shown as below:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} \text{dist}(x, \text{center}(X))^2$$  \hspace{1cm} (3.9)

where $c_i$ is the centroid of cluster.

III. Fuzzy K-Means Algorithm:

Fuzzy clustering (also referred to as soft clustering) is a form of clustering in which each data point can belong to more than one cluster. In fact, it is based on the k-means and the reinforcement of k-means clustering which allows abolishment of requirement to ‘crisply’ assign an instance to exactly one cluster. Instead in fuzzy k-means, instances can be assigned to a certain degree to more than one cluster and this method is more robust than traditional k-means algorithm. In fuzzy k-means clustering, euclidean distance is commonly used which is same to k-means clustering algorithm. Use euclidean distance formula as below:

$$d_{i\alpha}^2 = ||x_i - v_\alpha||^2 = \sum_{i=1}^{K} (x_{ij} - x_{\alpha j})^2$$  \hspace{1cm} (3.10)

In fuzzy k-means, because one instance does not belong to exact one cluster, we use a membership degree matrix to store the belonging status of each instances to every clusters and for which holds the formula that:
For $u_{\alpha i} \in [0, 1], \forall \alpha \in \{1..c\} \land \forall i \in \{1...N\}$

$$
\sum_{\alpha=1}^{c} u_{\alpha i} = 1, \forall i \in \{1...N\} \land 0 < \sum_{i=1}^{N} u_{\alpha i} < N, \alpha \in \{1..c\}
$$

(3.11)

Where $c$ represents the total number of clusters and $N$ represents the total number of the instances in our data set.

And similar to the k-means algorithm, in the fuzzy algorithm, we try to form the clusters in which we can optimize the object function. For fuzzy k-means algorithm, we use the formula below as the object function for which we try to optimize iteratively:

$$
J_m(U, V) = \sum_{i=1}^{N} \sum_{\alpha=1}^{c} u_{\alpha i}^m d_{i\alpha}^2 = \sum_{i=1}^{N} \sum_{\alpha=1}^{c} u_{\alpha i}^m |x_i - v_{\alpha}|^2
$$

(3.12)

The fuzzy k-means clustering process:
Firstly, we randomly initialize membership matrix $U$ and confirm $k$ initial centers. Then the values in matrix $U$ and centers are iteratively computed. The parameter $m$ represents fuzzifier.

$$
v_{\alpha} = \frac{\sum_{i=1}^{N} u_{\alpha i}^m x_i}{\sum_{i=1}^{N} u_{\alpha i}^m}
$$

(3.13)

$$
u_{\alpha i} = \frac{1}{\sum_{\beta=1}^{c} (\frac{d_{i\beta}^2}{d_{i\beta}^2})^{\frac{1}{m-1}}}
$$

(3.14)

Similar to k-means, the initialization of $k$ center points will also influence the final clustering result heavily. Before we implement our algorithm, we need to confirm the value of parameter $k$, initialize the k centers and define the fuzzier $m$.

2). Comparison of Different Typical Candidate Primitive Hierarchy Structure Construction Algorithms

After explaining several typical candidate algorithms in details, we can easily find the advantages and disadvantages of each algorithm and in which situation they can work well [XW05]. For this part, we also consider their time complexity and automatic degree they can provide for feature model construction to list all the advantages and disadvantages of each algorithm and then considering the listed advantages and disadvantages to explain the reasons why we chose hierarchical agglomerative clustering algorithm to realize the feature extraction process in our work after we compared and analyzed a set of candidates approaches in next algorithm choosing part. And because of the particularity of the fuzzy k-means algorithms that it is based on normal k-means, we can seem fuzzy k-means and normal k-means algorithms as one class of clustering algorithms when we consider the advantages and disadvantages for different algorithms.

I. Hierarchical Agglomerative Clustering Algorithm

Advantage:

(1) At the beginning before we implement the algorithm on data set, it is not needed
3.4. Feature Model Generation

to pre-define the number of clusters in advance.
(2) There are simple metrics and statistics which also allow user to have a better understand of the algorithm implementation process and be able to intuitively observe the final result through the dendrogramm.
(3) Facing the different situations and particularities of data set (e.g. outliers), there are appropriate variants (different distances measures) we can choose.
(4) When this algorithm is used in feature extraction field, it can promise a very high automatic degree to extract feature clusters from textual descriptors with fixed formation and without additional operations to get the hierarchy structure of different levels features.

Disadvantages:
(1) For hierarchical agglomerative clustering algorithm, the complexity of it is $O(N^3)$ in which $N = |D|$, although there are some variants with the complexity $O(N^2 log N)$. So when we use it for some situations that there is a large set of data as input, plenty of time will be needed to wait for final result.
(2) When we use hierarchical agglomerative clustering algorithm, we also need to be careful of each merging clusters step at each iteration which will affects our final result and the quality of all subsequent iterations of merging clusters.

II. K-means Algorithm

Advantages
(1) The time complexity of k-means clustering algorithm is better than the most of other clustering algorithms. The time complexity of k-means is $O(K \times |F| \times |D| \times I)$, for feature space F, dataset D, and number of iterations I.
(2) The process of the k-means clustering is very easy for users to understand.
(3) If the total number of clusters is known in advance, the k-means clustering algorithm can easily and precisely get clustering result.

Disadvantages:
(1) K-Means favours spherical clusters. So this algorithm is more suitable for the clusters which are the spherical shapes. As for other shapes' group it is not so easy to find using this algorithm.
(2) K-Means favours clusters of the same size. So if several clusters have the same or closed sizes, it is easy to detect them using k-means algorithm. But as for the clusters with different sizes, it is very hard to use this algorithm to detect all of them.
(3) K-Means favours clusters of the same density, so if the distributions of the instances in every cluster are similar or the same, k-means can easily detect all the well distributed clusters with the closed densities.
(4) K-Means is very sensitive to outlier objects: if there are some noisy points or outliers existing among the normal instances, the k-means clustering process and result will be bothered and mislead.

3). Candidate Algorithms Choosing Criteria and Algorithm Choosing

For different candidate feature extraction algorithms such as k-means and hierarchical agglomerative clustering algorithm as example explained in our paper, we need to find the algorithm which has to satisfy some special requirements to promise
that it can work well and efficiently in our situation to enable us can efficiently and as automatically as possible extract features contained in a set of descriptors gathered from the production descriptions repository in website. There are several requirements [BKS15] we need to consider carefully in order to achieve our aim:

1) Time complexity: in our work, we not only need to promise the accuracy and quality of our work result, we also need to ensure that we can efficiently get the result. When we do this transaction in practical situation in our daily work, we will face at least thousands of records and we have to deal with them as quick as possible. If the algorithm we use is not efficient and need to take up a large amount of hardware and time resources, we can not ensure that we can finish our work in time and satisfy the customers’ needs and it is also a waste of a variety of resources.

2) Automatic level: in our work, we are trying to come up with a idea that we can extract features from a set of descriptors using some special and suitable methods that can promise the whole process especially the specific feature extraction process we mentioned here can be done as automatically as possible and decrease the manual human involvement as much as possible. In practical work, if we want to promise our final approach can be utilized in some companies, we have to promise that our process does not need a lot of human involvements and the normal employers in the companies can use it individually without a lot of training.

3) Without so many preconditions: in our work, we tried to only extract information from textual contents and do not need to know so much knowledge and information in advance. For example, for k-means clustering algorithm, we need to consider the number of the clusters and the initialization process of the k cluster centers in order to get a not so bad clustering result. So for this situation, we need to know some information in advance before we start to implement the algorithm. If we have not known anything and randomly do the initialization work, the final clustering result will heavily depend on the luckiness and chance.

4) Deal with noisy data and outliers: In our work, we simulated a situation that all the descriptors we got from the website contains available information and all the descriptors we extracted can represent features. So when we extracted the feature model, we did not need to consider noisy data and outliers. While facing the real working situation, when we extract data from website or other resources, the qualities of the data extracted and the reliability of the data resources can not be promised. So it can not be avoided that there are some noisy data and outliers existing in our data set which will influence and mislead our working process and our final results.

Considering the problems discussed above, we need to compare different types of algorithms we can use for our work and find the most suitable one. In our work, we chose hierarchical clustering algorithms (HCA) because of several reasons:

1) Hierarchical agglomerative clustering algorithm can promise a very high level of automation of feature extraction in our work. In our work, firstly we input all the extracted descriptors with fixed formation, then we can directly use the application implementing the hierarchical agglomerative clustering algorithm to get a clustering tree which represent a primitive hierarchical structure. And only few later steps like flattening feature tree need additional measures to be done. To the contrary, if we use some other approaches like k-means, we can get some features, but later we also
need to analyze the hierarchical relationships among clusters and build the feature model by ourselves. That will lead to too much manual human involvement and can not promise enough automation in the whole process.

(2) For hierarchical clustering algorithm, we do not need to have some knowledge about the data set we want to process in advance. The whole process is just that input the descriptor set, implement the hierarchical agglomerative algorithm on input data and finally get the output hierarchical clustering tree. To the contrary, if we use some other approaches like k-means and fuzzy k-means, we need to know some additional information in advance in order to get the qualified result. For example, in k-means, only if the total number of clusters is known in advance and we can initialize the cluster centers properly, the k-means clustering algorithm can easily and precisely get clustering result.

(3) Considering about dealing with noisy data and outliers, hierarchical agglomerative clustering algorithm can also perform well. For different situations, we can choose different variants of distance measures for merging clusters process to avoid the influence of the noisy data and outliers.

Although the time complexity of hierarchical clustering algorithm can not be said in the perfect level, comparing with some other algorithms, it is still in the qualified level. Some other algorithms like k-means, have really good time complexity compared with hierarchical agglomerative clustering algorithm, but there are also some disadvantages of them which can not be ignored.

3.4.2 Primitive Tree Structure Simplification and Flattening

After implementing the hierarchical agglomerative clustering algorithm based on a set of descriptors, the output is a binary tree of clusters (each leaf node is a descriptor and also represent a cluster with only one descriptor). In order to decrease the complexity of our output to better and more precisely represent all of our extracted features, we have to simplify and flatten the output tree by setting a similarity threshold according to our experience and trying to merging all the subnodes into their parent node whose similarity is beyond the threshold through traversing all the nodes from bottom to up.

Figure 3.22 below shows an example process of flattening and simplifying a hierarchy structure to get the primitive feature model structure. The example tree in left side of figure is the original binary tree, the leaves of this tree represent descriptors (actually clusters with single descriptors), numbered 1 to 5. Each inner node exhibits its identity (e.g., $C_{1 \cdot 2}$) and the similarity of its two child nodes. Note that the sub nodes whose roots is $C_{1 \cdot 2}$ are very similar to their same level neighbor. Therefore, in the flattening and simplifying process, we can firstly merge R1 and R2 into their parent nodes as one node. Then because that the new generated node is very similar to its same level neighbor R4, we can merge them into their parent node $C_{1 \cdot 2}$. Finally we can get a simplified tree structure shown in right side of figure 3.22 [IRB14b].
3. Design of Feature Model Generation Process

Figure 3.23: Flattening and simplifying the clustering outcome in the feature diagram creation stage

After finishing this step, we store the simplified primitive feature model structure in a node list in which a class named node is used to store the nodes in primitive feature tree we have got which represent a potential feature and use pointers to represent the connections between different levels nodes. The nodes list then become the input for next step relationship detection. As shown in Listing 3.11 below, we can see the basic properties and methods provided in class node.

```java
1  int NodeID; // used as the identification of each node
2  int SubNode1; // the ID of one of the subnodes of the node
3  int SubNode2; // the ID of another one of the subnodes of the node
4  String RelationshipType; // relationship between parent and children
5  String NodeType; // the type of the node
6  List<Row> RowSet; // a list of descriptors contained in the node
7  String featureName;

8  public double nodeSimilarityCalculation(Node n1, Node n2)
9  public void featureNaming(Node node) // used to name the feature
10 public String RelationTypeDescision(Node node)
```

Listing 3.11: Class Node

3.4.3 Relationships Detection

After getting the primitive feature tree in which each node represents a potential feature stored in a node list, we used the node list as the input to deduce the relationships in feature tree. In our work, as mentioned before, the hierarchical structure we get also conceptually organize the feature into different levels in which we need to consider the different relationships between the parent and its children nodes. There are four main relationship types we need to consider in our work [DDH+13].

According to the definitions of different relationship types. With the condition
that there are at least two descriptors that are gathered under the same parent
node appearing in the same input document, then the corresponding relationships
are OR-grouped. And if all the descriptors clustered under the parent node are
stemmed from different input documents, we consider the corresponding descriptors
as XOR-grouped [IRB14b]. The node can be mandatory, if it includes descriptors
from all input documents, otherwise it should be optional [IRB14a].

In our work, for the Mandatory and Optional relationships, we can just use the
definitions of them to deduce them in our approach. As for deduction of the OR
and XOR relationships, sometimes it is too conservative to use the definition to
clearly detect the true relationship facts in real situation. So we came up with idea
that it is possible to take use of the similarity measure to deduce the OR and XOR
relationship types that if two subnodes are very similar, we can see them as XOR
relationship, and in contrast, if the difference between two subnodes is very large,
we will see them as OR relationship. The reason why we consider like this is that
in real situation, if two feature are very similar, it is very probable that the most of
their functionality are overlapped, so most of the time, we only need to choose only
one feature among set of features if the features are very similar. To the contrary,
if the features are not so similar, it is very probable that the functionality of two
features are very different and so for most of the time, for one product, we need
to choose many of them at the same time. Using the similarity measure can be
seemed as another way to deduce the relationships OR and XOR by computing the
similarities between the child nodes under one parent nodes. And first of all, we
need set a threshold for the similarity values, if the similarity value between two
child nodes are less than the threshold, we can set the XOR relationship for them,
else, set the OR relationship. And we will discuss about the problem that if the
method of deducing OR and XOR relationships can appropriate and cover all the
cases and if there are some special cases that don not conform to this criterion in
chapter Threats to Validity.

And in our work, we labeled the descriptors with the ID of document from which
we got this descriptor. Then we traverse all the nodes except the root in the tree
structure to check if the descriptors in one node is relative to all the documents we
extracted. If in one node, its descriptors are extracted from all the documents, we can
seem it as mandatory, otherwise, it is optional. As for OR and XOR relationships,
in our work, we traversed all internodes and we calculated the similarity of subnodes
of every internode and set a similarity threshold equals to 0.62 which is set because
of our experience in data mining research field, if the similarity is larger than the
similarity threshold, we can seem it as XOR, otherwise we seem it as OR. Listing
3.12 shows the process we implemented to do the relationship detection job.
3. Design of Feature Model Generation Process

3.4.4 Feature Naming

In order to create a feature model which is easily understandable for the users, the clusters in the hierarchical structure which represent the different features should be given a meaningful name which can represent the meanings of descriptors it contains and its relative functionality. Based on our basic demands and needs, we developed a process for nodes naming which contains the process of the selection of the most frequent item sets that a set of descriptors contained in the cluster of the node. Then we try to identify the suitable name for the nodes with our extracted and most frequent item sets with maximal size. There are 3 steps in total to get final extracted name for the node from its set of relative descriptors.

We did these three steps: descriptors prune, frequent itemset finding and name selection respectively.

1). Descriptors Pruning:

To easily access the descriptors and get the most frequent item sets in next step, firstly we need to use natural language processing tool to prune the descriptors and retain only useful nouns, adjectives and verbs which can usefully describe a feature. In fact, in our work, we have finished this job using Stanford Part-of-Speech (POS) in preprocessing step to prune the descriptors and only retain useful words in every descriptor because we only use nouns, adjectives and verbs in descriptors to extract features.

2). Frequent Item Sets Finding:

Try to find all the frequent item sets from the set of descriptors in the cluster of one node, with a given support value.

Support of an item set X is the sub set of transactions from D, that fulfill the rule:

$$support(X) = \frac{|t \in D | X \in t|}{|D|}$$  \hspace{1cm} (3.15)

In our work, we set a minimum support threshold and the support of one item set X is the total numbers of descriptors containing all the items in the item set X divided
3.4. Feature Model Generation

by the total number of descriptors in one node. If the support of the item set is larger than the minimum support we have set, we can seem this item set as the frequent item set for in this node.

3). Frequent Item Sets Mining

From previous relevant research paper [DDH+13], we have found various algorithms existing for doing the frequent item sets mining work including Apriori and FP-Growth algorithms and so on. And we chose Apriori algorithm in our work.

4). Name Selection

To Select the name for a cluster in one node, firstly all of its frequent item sets of maximal size need to be found and based on them, we can find the suitable names. To select the name of a cluster, we use the frequent item sets with maximal size from all the frequent item sets to form a term set $T_{S_{max}}$. Then all the descriptors in the cluster are examined. For every word $w$ in the $T_{S_{max}}$, we set a value equals to $Weight(w) \times frequency(w)$ in which $weight(w)$ means the weight of the word given in the similarity calculation step and $frequency(w)$ means the times of occurrence of the word $w$ in the cluster. Last step, after calculation the value for all the words in $T_{S_{max}}$, we choose the top 2 words ranked by their given values as the name of the node. And we can also manually change the name given by the above process.
3. Design of Feature Model Generation Process
4. Feature Model Construction
Process Implementation

After finishing the design work of our whole feature model generation process, we will try to utilize the suitable tools and implement the algorithms to build our feature model construction process based on a set of descriptions and show the sample work we have done through using our own feature extraction approach with antivirus software field as sample domain analysis.

4.1 Implementation of Clustering Algorithm for Primitive Hierarchy Construction

In our sample work, we implement the hierarchical agglomerative clustering algorithm with different distances measures respectively to build all the possible primitive hierarchy structures. Then we tried to compare these different clustering results and chose the most suitable one to our given sample data set of descriptors. In real work situation, we do not need to implement all the distance measures. Instead we can analyze our situation and choose the most suitable one considering their advantages and disadvantages.

In our simulated example situation, we used 21 descriptors of antivirus software products as sample data set which are extracted and processed carefully from software products website to show our designed feature extraction process. Figure 4.1, 4.2 and 4.3 respectively show our hierarchical agglomerative clustering results using MAX, MIN and Group average distance measures respectively.
Figure 4.1: Hierarchical agglomerative clustering result using MAX distance measure

Figure 4.2: Hierarchical agglomerative clustering result using MIN distance measure
As shown in figures above, in our simulated sample work, we used 21 descriptors which are extracted and preprocessed carefully from software products website containing software products descriptions as data resource. Each leaf node numbered from 0 to 20 in the figures represents a single descriptor. In our simulated work, although we only used a set of simple descriptors without considering outliers and some exceptions which will mislead our deduction and influence our final result accuracy, using these different distance measures respectively still leads to difference about our final results. When we consider about real working situation, there are plenty of outliers and exceptions which will easily make us fall into trap and we have to be very careful of choosing the most suitable distance measure from a set of candidates considering their advantages and disadvantages and real working situation that we are facing. From the figures shown above, we can also see that if we use different distance measure, we will get the different clusters merging sequences which might influence the final result heavily.

4.2 Primitive Tree Structure Simplification and Flattening

During this step, we try to get a simplified primitive feature model structure which is a tree structure and in which every node represents a potential unnamed feature. Figure 4.4 shows the process we implemented the hierarchical tree structure simplification and flattening based on the pre-defined distance threshold and the tree structure gotten from last step using hierarchical agglomerative clustering algorithm.
with MIN distance measure shown in figure 4.2. In our work we chose MIN distance measure to get the hierarchy because that as shown in figure 4.1, figure 4.2 and figure 4.3 for different distance measures, hierarchy generated by MIN distance measure is more reasonable considered from semantic and functional aspects of the products functionalities and also more accurate comparing with the other resultant hierarchies. In the situation of using MAX distance measure, the descriptors representing trojan and rootkit and keylogger cannot be clearly divided. As for the situation of using Group Average distance measure, the descriptors representing the file scanning and mail scanning are clustered together without clearly distinguishing.

![Hierarchical structure generated by hierarchical agglomerative clustering with threshold for simplification](image)

Figure 4.4: Hierarchical structure generated by hierarchical agglomerative clustering with threshold for simplification

As shown in figure 4.4, after getting a tree structure by implementing hierarchical agglomerative clustering algorithm, we set a distance threshold equals to 0.4 because of our experience for similarity measure. All the subtrees whose distance is under this threshold can be seemed as one node.

### 4.3 Relationships Detection

In our work, as mentioned before, we labeled the descriptors with the ID of document from which we got the descriptors. Then we traverse all the nodes except the root in the tree structure to check if the descriptors in one node is relative to all the documents we extracted. If in one node, its descriptors are extracted from all the documents, we can seem it as mandatory, otherwise, it is optional. As for OR
4.3. Relationships Detection

and XOR relationships, in our work, we traversed all internodes and we calculated
the similarity of subnodes of every internode and set a similarity threshold equals to
0.62 which is set this value because of our experience, if the similarity between the
subnodes is larger than the similarity threshold, we can seem it as XOR, otherwise
we seem it as OR.

After traversing all the nodes in the feature tree, in our work, we detected and
output all the relationships contained in the feature tree. Figure 4.5 and 4.6 shows
the output result of our work for detecting Mandatory/Optional and OR/XOR rela-
tionships respectively based on the input tree generated from last step on antivirus
software field.

Figure 4.5: Mandatory/optional relationship detection

Figure 4.6: OR/XOR relationship detection
4.4 Feature Naming

After the feature naming step mentioned in last chapter, we can generate a feature model represented by a feature tree structure. Figure 4.7 shows the final feature tree structure with feature names generated in our sample work based on the descriptors of antivirus software products.

Figure 4.7: The final feature model tree built in our work
5. Evaluation

After coming up with the entire process for feature extraction, we need to evaluate the process by measuring the qualities of the feature models generated through our own feature extraction process based on different sets of descriptions of different fields’ software products such as antivirus softwares and multimedia player softwares respectively retrieved from website through comparing them against the standards or using some special evaluation measures.

5.1 Evaluation Criteria

For evaluating the quality of our feature extraction process, there are some main criteria we have to consider to evaluate the quality of our own feature extraction process.

- Availability: we need to consider if our own process can work properly and generate suitable feature models.

- Universality: it means that our feature model extraction process should be available for most of the cases in software engineering fields considering the different types of software products.

- Accuracy: we have to promise that our feature extraction process can generate suitable feature models containing pertinent and correct features.

- Information Integrity: the feature models generated by our feature model construction process should include all the potential features which are useful and contained in the products descriptions used for feature extraction.

In next chapter 5.2, we will illustrate the details about how to evaluate our feature extraction process considering the criteria mentioned above through measuring the qualities of the feature models generated in our feature extraction process from qualified and quantitative aspects respectively.
5.2 Evaluation Setups

In this part, we will illustrate the different procedures of doing evaluation works from different aspects and the methodologies used in different procedures.

5.2.1 Different Aspects to Evaluate

In our paper, we will concentrate on two aspects to measure the qualities of our generated feature models. They are qualified and quantitative respectively.

(1). Qualified Evaluation

In this part, we will compare the generated feature model of antivirus software in our work with the other generated feature model provided in [DDH+13] built on the data which is in the same domain of antivirus software products that we used in our sample work to generate the sample feature model. In this part, we will manually compare the different features, the relationships between features and their names respectively in two feature models to check the differences between these two feature models to evaluate if our own feature model is better than the other feature model provided in the other research work.

(2). Quantitative Evaluation

In this part, we will use the evaluation approaches in data mining field to measure the qualities of the feature models generated in our work based on different software fields’ products descriptions. There are several measures chosen in our work used for our evaluation work. They are entropy, purity, cohesion, separation, recall, precision and f-measure respectively. After getting the data mining evaluation results, we try to horizontally compare the different evaluation values of different evaluation methods within every generated feature model and vertically compare the different generated feature models through comparing each evaluation method’s values between them.

5.2.2 Methodology for Evaluation Work

In our evaluation work, we have to choose suitable methodologies which can be used for measuring the qualities of our generated feature models.

(1). Methodology for Qualified Evaluation

In this part, we compare the generated feature model in antivirus software filed in our work with the other generated feature model provided in [DDH+13] build on data in the same domain of antivirus software products to see if we can get similar or even better result. We will manually compare the different features, the relationships between features and their names respectively in two feature models to check the differences between them and find out if our generated feature model contains more correct and complete features, relationships and feature names according to our experience about antivirus software field. Then We can come up with the conclusion that if our own feature extraction process can meet the availability, accuracy and information integrity requirements. Figure 5.1 shows a feature model tree generated in our work and figure 5.2 is the feature model generated in previous research paper
Both of these two feature model trees are based on data in the same domain of antivirus software products.

![Feature model tree built in our work](image1)

Figure 5.1: Feature model tree built in our work

![Feature model tree built in [DDH+13](image2)](image2)

Figure 5.2: Feature model tree built in [DDH+13](image2)

### 2. Methodology for Quantitative Evaluation

In last part, we plan to compare the feature model of antivirus software generated in our work with the other generated feature model provided in [DDH+13](image2) built on data in the same analysis domain of antivirus software products to see if we can get similar or even better result. In this part, we will use some data mining evaluation measures commonly used in data mining field [BKS15](image2) to evaluate the qualities of our resultant feature models generated in our work on several fields. And
in our work, we generated feature models focusing on three representative software products fields antivirus, multimedia player and Web browser respectively. The advantages of these three considered feature models is that they correspond to different and representative domains of software engineering field and therefore are able to prove the broad availability of our own feature extraction process. Figures below show the feature models we have generated on different fields and their truth sets generated manually by ourselves and corresponding to each of the feature models generated by our own feature extraction process.

In this part, we will need to consider how well we synthesize feature models conforming to the input descriptors or compare the generated feature model and its truth set in the same domain. And we use three different software products types as analysis domains to generate feature models and evaluate them to check the universality of our own feature extraction process and the resultant values of different evaluation methods will also reflect the availability, accuracy and information integrity of our feature extraction process.

Figure 5.3: Generated feature model on antivirus software field

Figure 5.4: Generated feature model on multimedia player field
5.2. Evaluation Setups

Figure 5.5: Generated feature model on web browser field

Figure 5.6: Truth set on antivirus software field

Figure 5.7: Truth set on multimedia player field
There are two types of evaluation measures we can use for evaluation of our clustering results. They are internal and external indices respectively.

I. Internal Indices

For the internal indices, we evaluate the results without knowing their labeling knowledge. In our situation, we did not consider the real functions that the descriptors represent in clusters of the nodes and only considered the distance (dissimilarity) between instances (descriptors) to check if we have got the compact clusters. The typical measures are sum of the squared errors (SSE), cohesion, separation and so on. In this paper, we will focus on cohesion and separation evaluation measures which are more suitable to our situation to check if in each of our generated clusters, the instances are impact or for different clusters, if they have been separated well.

For a clustering $\xi$ over a set $D$, computed with a distance function $d()$ (the distances are normalized):

$$\forall X \in \xi \hspace{1cm} cohesion(X) = 1 - \frac{1}{|X|(|X| - 1)} \sum_{x,y \in X; x \neq y} d(x, y)$$ (5.1)

$$separation(X) = \frac{1}{|\xi| - 1} \sum_{y \in \xi / x} \frac{1}{|X||Y|} \left( \sum_{x \in X, y \in Y} d(x, y) \right)$$ (5.2)

After getting separation and cohesion values for each cluster in clustering result, we can calculate the overall weighted cohesion and separation values for the clustering result.
5.2. Evaluation Setups

Listing 5.1 and 5.2 below show the processes for cohesion and separation calculations respectively.

```
1 Input: ClusterSet
2 for i=1 to ClusterSet.size
3   sum+=clusterCohesion(ClusterSet[i])
4 end
5 cohesion=sum/ClusterSet.size
6 return cohesion
```

Listing 5.1: Cohesion Calculation

```
1 Input: ClusterSet
2 for i=1 to ClusterSet.size
3   for j=i to ClusterSet.size
4     distance=clusterSeparation(ClusterSet[i],ClusterSet[j])
5     sum+=distance
6     num++
7 end
8 end
9 separation=sum/num
10 return separation
```

Listing 5.2: Separation Calculation

The formulas and algorithms above shows the calculation process to get cohesion and separation values. In our work, after getting the hierarchical clustering feature tree structure, we tried to seem all the leaf nodes in the tree as the separated clusters and calculated the weighted average cohesion and separation values among them.

II. External Indices

For the external indices, to the contrary of internal indicies, we evaluate the results under the condition of knowing instances’ labeling knowledge [HHM+14]. It means that in our situation, we considered the real functions the descriptors represent in clusters of each node to measure how accurate our process is to put descriptors with same functionality into the same cluster or know the truth sets of the generated feature models which will be used to measure how pertinent our generated feature models are considering their corresponding truth sets. The typical measures are entropy, recall, precision, f-measure and so on. In paper, we will focus on entropy, purity, recall, precision and f-measure evaluation measures which are more suitable to our situation.

1). Entropy and Purity

For entropy and purity measure, they are used to measure how accurately we cluster instances with the same labels into the same clusters and how pure each cluster is considering the instances’ labels distribution in the cluster.

As for us, in our situation, we will seem all the leaf nodes in generated feature model which contain sets of descriptors as different classes which represent different functionalities and then manually check the real meanings of each descriptor
and functions they represent to see if these descriptors are classified into suitable classes(functionalities) and get entropy and purity values.

Given is the dataset $D$, the true assignment of the data points in $D$ to classes $\varphi = C_1...C_L$ and a set of clusters $\xi = X_1...X_k$ over $D$.

**a). Entropy**

$$entropy(\xi, \varphi) = \sum_{i=1}^{k} \frac{|X_i|}{|D|} \text{clusEntropy}(X_i, \varphi)$$ (5.3)

where $\text{clusEntropy}(X_i, \varphi) = -\sum_{j=1}^{L} p_{ij} \log_2(p_{ij})$ with $p_{ij} = \frac{|X_i \cap C_j|}{|X_i|}$

**b). Purity**

$$purity(\xi, \varphi) = \sum_{i=1}^{K} \frac{|X_i|}{|D|} \text{clusPurity}(X_i, \varphi)$$ (5.4)

Where $\text{clusPurity}(X_i, \varphi) = \arg\max_{j=1...L} p_{ij}$, e.g. each cluster $X_i$ is assigned to the class, to which most of its members belong.

The formulas above show the calculation process we use to get entropy and purity values. In our work, after getting the hierarchical clustering feature tree structure, we tried to seem all the leaf nodes in the tree as the separated clusters which represent one feature and every feature we have found can be seemed as one class containing the descriptors who represent this feature, and then check the descriptors in every cluster to find which feature they belong to to get the entropy and purity value for each cluster. Finally we can get the weighted average entropy and purity values for the overall clustering result.

**2). Recall, Precision and F-measure**

We can use recall, precision and f-measure(the weighted harmonic mean of the precision and recall) which are calculated in terms of features to evaluate the ability of our method in identifying the features. Recall is used to present the proportion of pertinent features contained in our generated feature model considering the total number of the features contained in our generated feature model and precision presents the proportion of pertinent features contained in our generated feature model considering the total number of the features contained in relative truth set. As for f-measure, it is the harmonic way for evaluating considering both of recall and precision. We use these measures to measure how correctly our feature extraction work has been done to extract correct and suitable features and how well our extracted features can cover the features contained in truth set.

To compute the recall and precision values, firstly, we have to list all the features extracted in one of our own generated feature models and those contained in its corresponding truth set. Then we need to manually compare the features’ names meanings within the generated feature model in our work and those from the truth set. In another word, we make two lists for each domain: the first contains the feature names retrieved from the feature model generated in our work and the second contains the feature names integrated in the feature model of the truth set. Then
we need to compare the names manually in two lists and find out the features names in two lists which have the same meanings for each domain.

Formulas below show the the ways to calculate recall, precision and f-measure values.

\[
\text{Precision} = \frac{TP}{TP + FP} \tag{5.5}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \tag{5.6}
\]

\[
F - \text{measure} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Precision} + \text{Recall}} \tag{5.7}
\]

Where \( TP \) denotes the true positive, i.e. the number of true features generated by our method; \( FP \) is the false positive, i.e. the number of features generated by our method but absent in the truth set. \( FN \) denotes the false negatives, i.e. the number of pertinent features, which are absent in the feature models generated by our own process.

5.3 Evaluation Results Analysis

In this part, we will try to analyze the evaluation results got from qualified and quantitative evaluation works and make the conclusion that if our own feature extraction process can work well.

5.3.1 Qualified Evaluation Result Analysis

After comparing two feature model trees represented in Figure 5.1 and 5.2, We can find that there are some difference between two feature trees. Both in figure 5.1 and figure 5.2, there are two branches under the root nodes named antivirus but with the not totally same meanings. In figure 5.1, we have two branches named Virus Protection and Scan Analysis, as for figure 5.2, the two branches are Scan Detection Files and Mail Spam. The main difference is that in our work, we put file scan function with mail scan function together as one branch, and anti-spyware function as another individual function below the root node. As for the previous research work result, they combine the scan file function together with anti-spyware function as one subnode under the root node and put mail scan function as another individual subnode below the root node. The main reason why this situation happens is that there are two aspects that technical aspect and semantic aspect we should consider when we construct the feature models and the bias to different aspects will lead to the different results. From basic software development aspect, we can seem file scan together with spyware protection. In our work, we mainly considered the semantic aspect. So in our result feature tree, we grouped the function file scan and mail scan together considering their closed semantic similarity. And the different methods used for the process of generating feature trees will also probably influence the final feature tree we get. As mentioned above, compared with feature model generated in previous research work, our own generated feature model is satisfied and with good quality.
5.3.2 Quantitative Evaluation Result Analysis

As shown in the table below, the evaluation values for each domain are listed. And according to these evaluation values, we can measure the qualities of our generated models from more mathematic and direct aspect.

Firstly, we horizontally compare the different evaluation values of different evaluation methods within each analysis domain. For antivirus software field, the entropy and purity values are very high which means that the most of the descriptors used for feature model construction have been grouped properly into right classes. Although cohesion and separation values are not so good compared with other evaluation values in the same domain, they are still satisfied for us. Recall and precision values are not in the perfect level because in our generated feature models, there are some features which are contained in our generated feature model but not in truth set and we did not extract some features which exist in truth set. And the relatively low recall and precision values also lead to the low f-measure value. For multimedia player, the high recall, precision and f-measure values mean that for our generated feature model of multimedia player, it has came up with the most the features existing in truth set without so many mistakes. But the cohesion and separation values are not so good which mean that in this generated feature model, some descriptors of the relative products have been grouped into the wrong clusters. As for the web browser field, all of the evaluation values of different evaluation methods are very good compared with the other domains which means that we have extracted the most of the suitable features and clustered the descriptors into the suitable clusters.

Secondly we will vertically compare the different domains’ evaluation results through comparing the values of every evaluation method between each pair of domains. In web browser field, all of the values of the evaluation results are very good compared with the other domains which means that the feature model generated in our work in web browser products domain has very good quality. And for multimedia player field, the cohesion and separation values are not so good compared with the values of cohesion and separation in the other domains which means that in the feature model generated in our work for multimedia player, the descriptors did not been perfectly clustered. And the recall, precision, f-measure values in antivirus field are not so good compared with the values of recall, precision, f-measure in the other domains which means that in the feature model of antivirus generated using our own process of feature extraction, there are some non-pertinent features which are not in the truth set and we did not extract all the features which are in the truth set when we constructed the feature model of antivirus using our own feature extraction process. For all of these three domains, the entropy and purity values are not bad which means that when we constructed the feature models for each domain using our own feature extraction process, we have rightly grouped the most of the descriptors considering their semantic meanings. Considering all the values in the table below and the three representative and common domains we have used for implementing and evaluating our feature extraction process, although some of the values are not so good, but all of them are at least at the satisfied level for us. So we can say that according to the evaluation results in this part, our own process of feature extraction based on software products descriptions can work well for most of the cases.
5.3. Evaluation Results Analysis

<table>
<thead>
<tr>
<th>Domain</th>
<th>Entropy</th>
<th>Purity</th>
<th>Cohesion</th>
<th>Separation</th>
<th>Recall</th>
<th>Precision</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antivirus</td>
<td>0.143</td>
<td>0.845</td>
<td>0.239</td>
<td>0.617</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>Multimedia Player</td>
<td>0.174</td>
<td>0.793</td>
<td>0.312</td>
<td>0.574</td>
<td>0.73</td>
<td>1</td>
<td>0.844</td>
</tr>
<tr>
<td>Web Browser</td>
<td>0.122</td>
<td>0.883</td>
<td>0.203</td>
<td>0.798</td>
<td>0.813</td>
<td>1</td>
<td>0.897</td>
</tr>
</tbody>
</table>

Table 5.1: Evaluation values for our generated feature models

5.3.3 Evaluation Conclusion

Until now, we have concentrated on two aspects to evaluate the qualities of our generated feature models. In qualified evaluation part, after comparing two feature models, we can find out that our generated feature model contains set of correct features which also correspond to suitable relationships among them and well-formed names which means that our generated feature model can meet the requirements of availability, accuracy and universality mentioned in chapter 5.1. And also from quantitative evaluation part, through analyzing and comparing different evaluation values, we can find that within each analysis domain, the evaluation results are pretty good which means that each of our generated feature models corresponding to different analysis domains can promise the availability, accuracy and information integrity requirements and when comparing the evaluation results among these three generated feature models, we can find that the feature models of all of these three different and representative analysis domains are pretty good which means that our own feature extraction process can work well for different cases in software engineering filed considering the universality requirement. So We can come up with the conclusion that according to the evaluation process shown above, our feature model generation process can get relatively accurate and satisfied results.
6. Threats to Validity

(1). Construct validity: Because of the limited time span in our research work, in the thesis, we only concentrate on the development process of our own feature model generation approach without paying enough time on studying other previous relevant research papers to systematically sort out and summarize them from which we can also draw on more helpful experience to our own research work. And for the relationship detection part, the method of using similarity to deduce the XOR and OR relationships cannot deal with all the cases considering different features we need to represent in our feature model. It can cover the most of the cases. But as for some special cases in which the sub-features under the same parent feature have low level semantic similarity value but they are mutually exclusive or have high level semantic similarity value but they are compatible, our method is not suitable.

(2). Internal validity: Because of the time and space complexity reduction consideration for us, in our sample work of implementing our own approach of feature model generation, we cut down the number of the sample descriptors used as our data set to compute the similarity matrix between descriptors and construct the feature model structure.

(3). External validity: In our work, we did not have the truth set used for the evaluation of the qualities of the feature models generated through our own work process. When we planned our whole research process, at the beginning, we wanted to use the evaluation measures recall, precision and f1-measure which can be used based on the truth data set to evaluate the qualities of the feature models generated by our own feature extraction process. But because of the reason that we did not have the truth data set, we had to try to create simulated truth data sets which might be as similar as possible to the real truth data sets which correspond to the same analysis domains and related data sets as the analysis domains and related data sets we have focused on in our feature extraction process.

(4). Conclusion validity: In our evaluation part, we lack the experts evaluation process in which experts will evaluate our own feature extraction process through their own rich experience to deduce if our feature extraction process and generated
feature models are reasonable and helpful for software engineering work to make readers more easily understand the true significance and meaning of our research work and get more authoritative and impartial evaluation for our own feature extraction process. In our work, in order to get the evaluation result as accurate as possible, we tried to evaluate our own feature model generation approach from two aspects that one is comparing the generated feature model in our sample work in antivirus field with the sample feature model created in previous research work [DDH+13] for same analysis domain and the other one is using data mining evaluation measures to measure the qualities of several feature model structures in several different analysis fields generated by our own feature model construction process.
7. Related Work

In the thesis, we present an implementation process of feature extraction from software products descriptions in which automation has been promised as much as possible. There are several related research works which have been done and presented in previous research paper.

(1). In previous research paper [CW07], K. Czarnecki et al. provided the technique for synthesizing an feature model from a collection of configuration and constraints. Comparing with our approaches, there are several disadvantages existing in this method in some special situations. Firstly, this method cannot be applied in the context that we cannot assume the variability of formal and complete descriptions of configurations or constraints. As for us, we developed an approach to extract features by dealing with the informal and textual natural products descriptions. Secondly, another limitation of this work is the identification of the feature hierarchy. In the work of K. Czarnecki and A. Wasowski, a diagrammatic result will be generated which represents all the possible feature models. But there is still the problem that how to select a unique feature model from all the possible candidates feature models. In our work, we try to adapt the variants in our feature generation process to finally generate a unique feature model which is as perfect as possible.

(2). M. Acher et al. presented a semi-automated process to implement transition from product descriptions to feature models [ACP+12]. In their process, there are too much human manual involvements needed. As for our own approach, we keep the very high automatic level from text input to final feature model generation.

(3). K. Czarnecki et al. introduced the process of feature model construction through association rule mining process [CSW08]. In their work, they used association rules with the confidence of 100% to build the hierarchy and association rules with the confidence less than 100% but above a predefined threshold to extent the feature models. And in [DDH+13], Jean-Marc Davril et al. talked about the fully automated, two-step process of mining features and building feature model through using Spherical k-Means clustering algorithm for clustering features and association rule mining for feature association mining to build the feature model. As for us,
in our approach, we utilize the Hierarchical Agglomerative Clustering technique to build the feature models which can simply and directly generate a feature model without the complicated process from getting set of features to building the feature hierarchy.

(4). Alives et al. conducted a related research [ASB+08] based on the use of Vector Space Model (VSM) and Latent Semantic Analysis (LSA) which are used to determine the similarities between requirements. As for our work, we use knowledge-based similarity measure to calculate the similarity between words and the method named Mihalcea, Corley, and Strapparava’s (MCS) to calculate the similarities between text segments. And as for descriptors similarity calculation, we also consider the syntactical structures of the descriptors which can get more accurate result compared with that of using Vector Space Model (VSM). And there are also more difficulties to collect the requirements information used to extract features compared with the way of using descriptions which can be easily extracted from software products website to extract features.

(5). In [KMB06], L. K. Kit et al. talked about the utilization of latent semantic analysis (LSA) for identification of concerns and aspects from natural language contents adapted in Theme approach [BC04]. They used the similarity analysis to detect the relative concerns which is also used in our process. But to the contrary, our main attention is on generation of the feature model trees and their variants instead of crosscutting.

(6). Chen et al. presented the method [CZZM05] to do the feature clustering and variability detection for building feature models based on the retrieved requirements. Although the clustering process can be done automatically, it is still based on the previous defined functional requirements and their relationships stored in a graph. As for our work, descriptors clustering is done based on similarity matrix generated by descriptors similarity calculation and we use the hierarchical agglomerative clustering algorithms for feature model tree construction which is also discussed by Chen et al..
8. Summary and Feature Work

In this paper, we presented a process for automatically generating feature models from a set of descriptions of software products retrieved from the relative website of software products through using a set of techniques for mining and summarizing products descriptions based on data mining and natural language processing methods. The results of our experimental work show that in the cases of antivirus software, multimedia player and web browser products which are some representative software products in software products field and used as analysis domains to generate feature models in our work as tests and example works, we can apply the clustering technique and other handling processes to get the suitable feature models with obvious improvements of the quality of the feature models generated by us compared with other previous research works.

Furthermore, one of the most important improvements of our approach of feature models construction compared with other approaches presented by previous researches is that during the process of generating feature model, for each step, we tried to choose the most suitable ways to get the most accurate intermediate results used for next steps to promise the high accuracy and quality of our final result. For example, for the descriptors similarities calculation step, we came up with the idea that use both of knowledge-based and string-based similarity measures to calculate the similarity between two descriptors. And another very important improvement of our approach is that it is suitable for different kinds of products and is not limited to only some special software products which have been used as domain analysis to generate feature models in our work as examples. The main reason why it is possible is that we used the WordNet tools containing the most of frequently used words and we can also use string-based similarity measure as the supplement for some exceptions. Under this condition, we can easily deal with the descriptions of different kinds of products and construct the final feature model trees through the hierarchical agglomerative clustering, relationships detection and feature naming techniques.

For future work, in one hand, we will try to improve the quality and accuracy of our approach to generate the feature models through adjusting the techniques and methods used in different steps of the whole feature model generation process especially the method used for XOR and OR relationship deduction which we will try
to make suitable for the most of the cases given the different features needed to be represented in the feature model. And in another hand, in future work, we will evaluate our approach on a far broader set of products to make sure that our approach can be available for other different types of products descriptions resources.
Bibliography


Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000. (cited on Page 11)


Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den July 2017