Component Classification and Retrieval Using Data Mining Techniques

Achala Sharma, Daman Deep Kaur
Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology, Patiala
{achala.sharma, damandeep}@tiet.ac.in

Abstract

A significant hurdle confronts the software reuser attempting to select candidate components from a software repository - discriminating between those components without resorting to inspection of the implementation(s). In this paper we discuss data mining techniques to this problem which avoids requiring the repository administrators to define a conceptual closeness graph for the classification vocabulary because all the existing techniques provide the static classification of components but the data mining techniques provides the dynamic classification of components, these techniques also provides an optimal balance between precision and recall rate.

Keywords: Data Mining, Component Retrieval, Software Reuse, Artificial Intelligence

1. Introduction

Reuse has long been an accepted principle in many scientific disciplines. Biologist’s use established laboratory instruments to record experimental results; chemist’s use standardized measuring devices. Engineer’s design based upon the availability of components that facilitate product development. It is unreasonable to expect an electrical engineer to design and develop the transistor from first principles every time one is required. Software engineers, however, are frequently guilty of a comparable practice in their discipline. The reasons for this are as varied as the environments in which software is developed, but they usually include the following:

- A lack of development standards;
- The not invented here syndrome;
- Poor programming language support for the mechanical act of reuse; and
- Poor support in identifying, cataloging, and retrieving reuse candidates.

The first three items involve organization mentality, and will not be addressed here. We instead focus upon the final item in this list, the nature of the repository itself, and more specifically upon the mechanisms provided for classification and retrieval of components from the repository. The complexity of non-trivial software components and their supporting documentation easily qualifies reuse as a "wicked" problem - frequently intractable in both description and solution.

2. The Problem

A mature software repository can contain thousands of components, each with its own specification, interface, and typically, its own vocabulary. Consider the signatures presented in Figures 1 and 2 for a stack of integers and a queue of integers, respectively.

Create: => Stack
Push: Stack x Integer => Stack
Pop: Stack => Stack
Top: Stack => Integer
Empty: Stack => Boolean

Figure 1: Signature of a Stack

Create: => Queue
Enqueue: Queue x Integer => Queue
Dequeue: Queue => Queue
Front: Queue => Integer
Empty: Queue => Boolean

Figure 2: Signature of a Queue

These signatures are isomorphic up to renaming, and thus exemplify what we have come to refer to as the vocabulary problem. Software reusers implicitly associate distinct semantics with particular names, for example, pop and enqueue. Thus, by the choice of names, a component developer can mislead reusers as to the semantics of components, or provide no means of discriminating between components. Figure 3, for example, appears to be equally applicable as a signature for both stack and queue, primarily due to the neutral nature of the names used.

Create: Sequence
Insert: Sequence x Integer => Sequence
Remove: Sequence => Sequence
Current: Sequence => Integer
Empty: Sequence => Boolean
3. Software Classification

Retrieval mechanisms for software repositories have traditionally provided some sort of classification structure in support of user queries. Keyword-based retrieval is perhaps the most common of these classification structures, but keywords are ill-suited to domains with rich structure and complex semantics. This section lays out the principle representational problems in software classification and selected solutions to them.

a. Literary Warrant

Library scientists use literary warrant for the classification of texts. Representative samples drawn from the set of works generate a set of descriptive terms, which in turn generate a classification of the works as a whole. The adequacy of the classification system hinges on the initial choice of samples. With appropriate tools, literary warrant in software need not restrict itself to a sample of the body of works. Rather, it can examine each of the individual works in turn, providing vocabularies for each of them. This may indeed be required in repositories where the component coverage in a particular area is sparse.

b. Conceptual Closeness

The vocabulary of terms built up through literary warrant typically contains a great deal of semantic overlap words whose meanings are the same, or at least similar. For instance, two components, one implementing a stack and the other a queue might both be characterized with the word insert, corresponding to push and enqueue, respectively, as discussed in section 2. Synonym ambiguity is commonly resolved through the construction of a restricted vocabulary, tightly controlled by the repository administrators. Repository users must learn this restricted vocabulary, or rely upon the assistance of consultants already familiar with it. It is rarely the case, however, that the choice is between two synonyms. More typically it is between words which have similar, but distinct, meanings (e.g., insert, push, and enqueue, as above).

c. Algebraic Specification

While not really a classification technique, algebraic specification techniques (e.g., [1]) partially (and unintentionally) overcome the vocabulary problem through inclusion of behavioral axioms into the specification. The main objection to the use of algebraic specifications in reuse is the need to actually write and comprehend the specifications. The traditional examples in the literature rarely exceed the complexity of the above Figures. Also, algebraic techniques poorly address issues such as performance and concurrency. A repository containing algebraic specifications depends upon the expertise of the reusers browsing the repository; small repositories are easily understood whereas it is unreasonable to require a reuser to examine all components in a large repository for suitability.

d. Basic Faceted Classification

Basic faceted classification begins by using domain analysis (aka literary warrant) “to derive faceted classification schemes of domain specific objects.” The classifier not only derives terms for grouping, but also identifies a vocabulary that serves as the values that populate those groups. From the software perspective, the groupings, or facets become taxonomy for the software.

Prieto-Diaz and Freeman identified six facets: function, object, medium, system type, functional area, and setting [2]. Each software component in the repository has a value assigned for each of these facets. The software reuser locates software components by specifying facet values that are descriptive of the software desired. In the event that a given user query has no matches in the repository, the query may be relaxed by wild-carding particular facets in the query, thereby generalizing it. The primary drawback in this approach is the flatness and homogeneity of the classification structure. A general-purpose reuse system might contain not only reusable components, but also design documents, formal specifications, and perhaps vendor product information. Basic faceted classification creates a single tuple space for all entries, resulting in numerous facets, tuples with many “not applicable” entries for those facets, and frequent wildcarding in user queries. A number of reuse repository projects have incorporated faceted classification as a retrieval mechanism (e.g., [3] [4]), but they primarily address the vocabulary problem through a keyword control board, charged with creating a controlled vocabulary for classification. Gagliano, et. al. computed conceptual closeness measures to define a semantic distance between two facet values [5]. The two principle limitations to this approach are the static nature of the distance metrics and the lack of inter-facet dependencies; each of the facets had its own closeness matrix.

e. Lattice-Based Faceted Classification

Eichmann and Atkins extended basic faceted classification by incorporating a lattice as the principle structuring mechanism in the classification scheme [6]. As shown in Figure 4, there are two major sub lattices making up the overall lattice.
On the left is the sub lattice comprised of sets of facet values (for clarity, shown here with only three facets), partially ordered by the subset relation. The Facet\(_1\) vertex in the lattice represents the empty facet set, while the Facet vertex represents the set of all facet values in the classification scheme. Each member of the power set of all facet values falls somewhere within this sub lattice. On the right is the tuple sub lattice, containing facet set components, and partially ordered by the subtype relation [7]. The vertex denotes the empty tuple. The tuple vertex denotes the tuple containing all possible facet components, with each component containing all the values for that facet. Adding facet values to a component or adding a new component to a tuple instance moves the tuple instance down through the lattice. Queries to a repository supporting lattice-based faceted classification are similar to those to one supporting basic faceted classification, with two important distinctions - query tuples can mention as many or as few facets as the reuser wishes, thereby avoiding the need for wildcarding, and classifiers can similarly classify a given component with as many or as few facets as are needed for precise characterization of the component. Lattice-based faceted classification avoids conceptual closeness issues through the specification of sets of facet values in the classification of components. If there are a number of semantically close facet values that all characterize the component, all are included in the facet instance for that component. This avoids the need to generate closeness metrics for facet values, but it also may result in reuser confusion about just what the component does.

f. Towards Adaptive Classification and Retrieval

The principle failing in the methods described so far is the static nature of the classification. Once a component has been classified, it remains unchanged until the repository administrators see fit to change it. This is unlikely to occur unless those same administrators closely track reuser retrieval success, and more importantly, retrieval failure - particularly in those cases where there are components in the repository matching reuser requirements, but those components were not identified during the query session. Manual adjustment of closeness metrics becomes increasingly unreasonable as the scale of the repository increases. The number of connections in the conceptual graph is combinatorially explosive.

The principle goal of this paper is to discuss about the techniques, which can create an adaptive query mechanism - one capable of altering its behavior based upon implicit user feedback. This feedback appears in two guises; failed queries, addressed by widening the scope of the query; and reuser refusals, cases where candidate components were presented to the reuser, but not selected for retrieval. The lattice provides a nice structure for the former, but a different approach is required for the latter. Besides of classification system the performance of a retrieval system is usually measured by recall and precision. Recall is the proportion of relevant material retrieved; measuring how well a system retrieves all the relevant material. Precision is the proportion of retrieved material that is relevant, measuring how well the system retrieves only the relevant material. Recall and precision tend to be related inversely. When a search is broadened to achieve better recall, precision tends to go down and vice versa. In the context of software retrieval, for a given query we want to find the most relevant software component for reusing it with minimum effort to adapt it to a new application. In this case, precision is more important than recall. However, if the recall is not maintained in a reasonable level some most relevant software components may be missed. Therefore, how to improve precision without excessive compromise of recall is crucial for software retrieval systems. To achieve such a retrieval performance an appropriate classification scheme must be developed.

4. Data Mining Techniques

The most commonly used techniques in data mining are:

a. Artificial neural networks: Neural networks are an approach to computing that involves developing mathematical structures with the ability to learn. Neural networks have the remarkable ability to derive meaning from complicated or imprecise data and can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Neural networks use a set of processing elements (or nodes) analogous to neurons in the brain. These processing elements are interconnected in a network that can then identify patterns in data once it is exposed to the data, i.e. the network learns from experience just as people do. This distinguishes neural networks from traditional computing programs that simply follow instructions in a fixed sequential order.

The structure of a neural network looks something in Figure 5. The bottom layer represents the input layer, in this case with 5 inputs labels X1 through X5. In the middle is something
called the hidden layer, with a variable number of nodes. It is
the hidden layer that performs much of the work of the
network. The output layer in this case has two nodes, Z1 and
Z2 representing output values we are trying to determine from
the inputs. For example, predict sales (output) based on past
sales, price and season (input).

Each node in the hidden layer is fully connected to the inputs
which mean that what is learned in a hidden node is based on
all the inputs taken together. Statisticians maintain that the
network can pick up the interdependencies in the model. The
following diagram provides some detail into what goes on
inside a hidden node.

Simply speaking a weighted sum is performed: X1 times W1
plus X2 times W2 on through X5 and W5. This weighted sum
is performed for each hidden node and each output node and
is how interactions are represented in the network. The
issue of where the network gets the weights from is important but
suffices to say that the network learns to reduce error in its
prediction of events already known (i.e., past history).

Hopfield networks are the type of neural networks; these
networks can be used as content-addressable or associative
memories. Initially the weights in the network are set using
representative samples from all the exemplar classes. After
this initialization, the input pattern \( I \) is presented to the
network. The network then iterates and converges to an
output. This output represents the exemplar class, which
matches the input pattern best.

Genetic algorithms: Genetic algorithms are non-
deterministic search algorithms based on the mechanics of
natural selection and natural genetics in a biological system.
Genetic algorithms have been used in modeling evolving
systems or combinatorial optimization problems. Genetic
algorithms are robust in many application areas and search a
huge problem space while exploiting historical information to
speculate on new search points with expected improvement of
performance [8, 9]. The genetic algorithms attempts to find a
very good or appropriate solution to the problem by
generically breeding the population of individuals. The
genetic algorithm transforms a population of individual
objects, each with an associated fitness value, into a new
generation of the population using the Darwinian principle of
reproduction and survival of the fittest and naturally occurring
genetic operations such as crossover and mutation. Each
individual in the population represents a possible solution to a
given problem [9, 10]. Before run genetic algorithms, we
define a relevant encoding of chromosome to solve a
problem, design an objective function for fitness, and
construct genetic operators. In order to run Genetic
Algorithms, we generate an initial population consisting of
chromosomes and evaluate these chromosomes using the
objective function designed. And we select two chromosome
randomly and crossover and mutate them and replace a low
quality chromosome with a new one of high quality. As these
processes have been repeated, the population consists of high
quality chromosomes. Classification can also be performed on
the basis of genetic algorithm; we can take an encoding
scheme [11], which maps the characteristics of software
component from their initial form (whatever that may be, e.g.
linguistic terms, numerical types, etc.) to a form that will be
used by a genetic algorithm to discover component classifiers.
The genetic algorithm attempts to discover several different
classifiers, each of which classifies a number of software
components into a homogenous set in terms of characteristics.
The classifying sets may have common elements as the
classification process is based on component characteristics,
with which it attempts to find large groups of components
with common values.

From this description we can understand that genetic
genome performs dynamic classification of the components
with the help of crossover and mutation process and on the
basis of fitness value it generates the next generation of
components in case of retrieval or classifiers in case of
classification.

c. Nearest neighbor method: A technique that classifies each
record in a dataset based on a combination of the classes of
the k record(s) most similar to it in a historical dataset (where
k \( \geq 1 \)). Sometimes called the k-nearest neighbor technique.

d. Rule induction: The extraction of useful if-then rules from
data based on statistical significance, data mine system has to
infer a model from the database that is it may define classes
such that the database contains one or more attributes that
denote the class of a tuple i.e. the predicted attributes while
the remaining attributes are the predicting attributes. Class
can be then be defined by condition on the attributes. When the
classes are defined the system should be able to infer the rules
that govern classification, in other words the system should find the description of each class.

5. Benefits of Data Mining in Component Classification and Retrieval

From all the data mining techniques which are described above, it is clear that all the data mining techniques provide

a. Dynamic classification of components:-In all data mining techniques component classification is not static but in it the classification is based on the some algorithms or process applied on the population at the time of retrieval such as in genetic algorithm, In neural networks unsupervised Neural networks are also called as Self-Organising Map SOM [12], A SOM can learn from its input data. Each input stimulus elicits a localised response. This corresponds to a non-linear projection of the input data onto the network that makes the most important semantic relationships among the input data items geometrically explicit [12]. It is this property of SOM or Neural networks that makes it useful for dynamic classification., so from all this we can say that data mining techniques provides dynamic classification of components, that’s why the performance of these techniques are better than other techniques.

b. Optimal balance between precision and recall:- We can achieve this if we perform retrieval in two steps, or in other words we can say that if we integrate data mining technique based retrieval with some other retrieval technique e.g. if we integrate genetic algorithm or neural network with keyword based retrieval, then we can maintain the balance between precision and retrieval, because keyword based retrieval maintains the recall rate in first step and data mining based retrieval maintains the precision rate when applied in second step. The NSOM [13] is a system, which integrates neural networks with fuzzy related thesaurus, the NSOM based classification will be done in two levels and the accuracy of the classification will be enhanced from the first course-grained level to the second fine-grained level. The coarse-grained classification at the first level is used to maintain a high level of recall and the precision will be improved by the fine-grained classifications at the second level.

6. Conclusion

In this paper we discussed various techniques such as Literary Warrant, Conceptual Closeness, Algebraic Specification, Lattice-Based Faceted Classification, Basic Faceted Classification and all the limitations during the classification and retrieval of components by using these methods such as the main objection to the use of algebraic specifications in reuse is the need to actually write and comprehend the specifications, the two principle limitations to Basic Faceted Classification approach are the static nature of the distance metrics and the lack of inter-facet dependencies; each of the facets had its own closeness matrix, Lattice-based faceted classification avoids conceptual closeness issues through the specification of sets of facet values in the classification of components. If there are a number of semantically close facet values that all characterize the component, all are included in the facet instance for that component. This avoids the need to generate closeness metrics for facet values, but it also may result in reuser confusion about just what the component does. So what’s the reason behind all these limitations, reason is the static nature of the classification defined by all the approaches. Once a component has been classified, it remains unchanged until the repository administrators see fit to change it. That’s the reason due to which data mining techniques are more successful in case of component classification and retrieval, because all the techniques based on data mining provides dynamic classification of components, instead of it we can also produce an optimal balance between precision and recall with the help of these techniques.

References