Journal of Computer Science 3 (5): 266-273, 2007
ISSN 1549-3636
© 2006 Science Publications

Approaches for Categorization of Reusable Software Components

[1]Parvinder Singh Sandhu, [2]Janpreet Singh and [3]Hardeep Singh
[1, 2] Department of Computer Science and Engineering, Guru Nanak Dev Engineering College
Ludhiana, Punjab, India
[3]Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar,Punjab, India

Abstract: Reuse repositories manager manages the reusable software components in different categories and needs to find the category of reusable software components. In this paper, we have used different pure and hybrid approaches to find the domain relevancy of the component to a particular domain. Probabilistic Latent Semantic Analysis (PLSA) approach, LSA, Singular Value Decomposition (SVD) technique, LSA Semi-Discrete Matrix Decomposition (SDD) technique and Naive Bayes Approach purely as well as hybrid, are evaluated to determine the Domain Relevancy of software components. It exploits the fact that Feature Vector codes can be seen as documents containing terms -the identifiers present in the components- and so text modeling methods that capture co-occurrence information in low-dimensional spaces can be used. The FV code representation of clusters or domains is used to find the domain-relevancy of the software components. PLSA has provided better results than LSA retrieval techniques in terms of *Precision* and *Recall* but its time complexity is too high. SVD Transformation with Naïve Bayes scheme has outperformed all other approaches and shows better results than the existing approach (LSA) being used by some open source code repositories e.g. Sourceforge. The DR-value determined is close to the manual analysis, used to be performed by the programmers/repository managers. Hence, the tool can also be utilized for the automatic categorization of software components and this kind of automation may improve the productivity and quality of software development.

Key words: LSA, Naïve Bayes, PLSA, Reusable Components, SVD, SDD

INTRODUCTION

The demand for new software applications is currently increasing at the exponential rate, as is the cost to develop them. The number of qualified and experienced professionals required for this extra work is not, however, increasing commensurably[1]. Software professionals have recognized reuse as a powerful means of potentially overcoming the above said software crisis [2, 3, 4, 5] and it promises significant improvements in software productivity and quality [6].

There are two approaches for reuse of code: develop the reusable code from scratch or identify and extract the reusable code from already developed code. The organization that has experience in developing software, but not yet used the software reuse concept, there exists extra cost to develop the reusable components from scratch to build and strengthen their reusable software reservoir [7, 8]. The cost of developing the software from scratch can be saved by identifying and extracting the reusable components from already developed and existing software systems or legacy systems [9, 10].

Tracz in [11] observed that for programmers to reuse software they must first find it useful. Poulin [12] further concluded that 65% of typical software is made-up of Domain-specific class of software. So we can expect the most savings, if we reuse the domain-specific software [13]. It means one should concentrate on evaluating the software in terms of its relevancy to a particular domain.

Kawaguchi in [14] used code clones-based similarity metric, decision trees, and latent semantic analysis (LSA) approaches to help finding similar software systems in software archive. Further, Kawaguchi in [15] explained the use of LSA approach to automatic categorization of software systems and developed web interface to visualize determined categories.

Corresponding Author: Parvinder Singh Sandhu, Assistant Professor, Department of Computer Science and Engineering, Guru Nanak Dev Engineering College, Ludhiana(Punjab)- 141 0006 India. Tel: 9855532004

In this paper, a probabilistic approach called PLSA, is shown to extract different aspects in a software component, that provides the domain-relevancy of the software component. The LSA approach is also proposed to automatically cluster feature-vector (FV) codes into meaningful categories. The FV code derived descriptions are computed by Latent Semantic Analysis (LSA) using Singular Value Decomposition (SVD) and Semi-Discrete matrix Decomposition (SDD) techniques. The FV code representation of clusters or domains is further used to find the domain-relevancy (DR-value) of the software components automatically. The purely and hybrid Naïve Bayes schemes are also tried for the categorization of the software components.

METHODOLOGY

An approach is proposed that allows automatic clustering of feature-vector (FV) codes, extracted from different software domains, into meaningful categories. It exploits the fact that FV codes can be seen as documents containing terms – the identifiers present in the components- and so text modeling methods that capture co- occurrence information in low-dimensional spaces can be used. The comments in the source code describing the function of the chunks of code and Good naming conventions used in the high-level language source code. Interfaces of the code components (such as function, class, module etc) make high use of verbs, nouns, adjectives and adverbs. These verb phrases convey the functional characteristics of the component. Therefore, function name, constant names and variable-names used make a good source of representation information. The common reserved words are excluded because they have no relation with software features. If, the keyword consist of more than one word joined with "underscore", "hyphen" or "capital letter and small letter combination" then the keyword is broken down to find the primitive or root words. The organizations of common keywords extracted from the samples of a domain can act as descriptor for that particular domain.

The words can be divided into two categories – open-class words and closed-class words. The words, which are nouns, verbs, adjectives or adverbs are called open-class words and are supposed to convey desired functional information about the component. The closed-class words include articles, pronouns, prepositions, conjunctions, interjections, helping verbs and do not convey any functional information. The closed-class words are eliminated from the list of keywords.

The FV code derived descriptions are computed by Probabilistic Latent Semantic Analysis (PLSA), LSA's Singular Value Decomposition (SVD), LSA's Semi-Discrete Matrix Decomposition (SDD) and Naïve Bayes Approaches and performance of different approaches is evaluated. The FV code representation of clusters is used to find the domain-relevancy (DR-value) of the software components.

Naïve Bayes Based Approach: The Naive Bayesian classification is the optimal method of supervised learning, if the values of the attributes of an example are independent given the class of the example [16]. Researchers have applied Naïve Bayes algorithm for the text-document classification. But values of the attributes of an example are not independent in case of text-document classification but the in case of the Software component classification the attributes of an example can be taken independent to each other. We propose two-step approach, as mentioned below.

Learning Phase: In the learning phase, let V is the set of all possible target values. This function learns the probabilities terms $P(w_k|v_j)$, describing the probability that an extracted features from a software component in class v_j will be the feature named w_k. It also learns the class prior probabilities $P(v_j)$. Vocabulary is the set of all distinct features and other tokens occurring in the example software components. The features and frequency of occurrence of the features in different example software components is collected using following steps:

1. Extract features from Training software belonging to different domains
2. Create Identifier-by-Software Matrix
3. Calculate the required $P(v_j)$and $P(w_k|v_j)$ probability terms. For each target value v_j in V, calculate $P(v_j)$ and $P(w_k|v_j)$ using (1) and (2).

$$P(v_j) = \frac{|\,docs_j\,|}{|\,Examples\,|} \tag{1}$$

$$P(w_k \mid v_j) = \frac{n_k + 1}{n + |\,vocabulary\,|} \tag{2}$$

Where $docs_j$ is the subset of software components from examples for which the target value is v_j, $Text_j$ is a single software component created by concatenating all members of $docs_j$, n is total number of distinct positions in $Text_j$ and For each feature w_k in Vocabulary, n_k is number of times features w_k occurs in $Text_j$.

Classification Phase: This phase returns the estimated target value for query the software component that need to be categorized. The following steps are involved in the classification phase:

1. Extracting features and frequency of features from the query software components as performed in learning phase.
2. Fold query software's frequency vector according to existing features of the Vocabulary.
3. Calculate V_{NB} according to (3).

$$V_{NB} = \underset{v_j \in V}{\arg\max} P(v_j) \prod_{i \in positions} P(a_i \mid v_j) \qquad (3)$$

Where a_i denotes the ith feature of query software and Positions is all features in query document that found in Vocabulary. This step also makes use of "Thesaurus" for automatically increasing the search space, by replacing a_i with a group of matched features.

PLSA Based Approach: The core of PLSA is a statistical, which is known as aspect Model [17]. Aspect Model is latent variable model for general co-occurrence data which associates an unobserved class variable z Є Z={z_1, z_2,…z_k} with each observation, i.e., with each occurrence of a word w Є W={w_1,w_2,…w_m} in a document d Є D = {d_1, d_2,…, d_N}. The following steps are proposed to find the DR-value of potential reusable components using training software components:

Learning Phase*:* The following steps are followed in the training phase:

1. Extract keywords/identifiers from Training software belonging to different domains. In identifiers include function names, constant names and variable-names used in the software. From identifiers, exclude reserved because they have no relation with software features. The comments are also included in the analysis to extract more meta information of the software component.
2. Create identifier-by-software matrix. Considering a software system as a document and an identifier as a word; create an identifier-by-software matrix, similar to the word-by-document matrix.
3. Remove useless identifiers and perform Normalization to obtain *f(d, w)* matrix.
4. Initialize the *P(w|z)* and *P(d |z)* randomly with numbers between [0,1] and normalize them to sum to 1 along rows. *P(z)* is also initialize randomly.
5. Apply EM algorithm [18] as shown in eq. (4)-(7) and iterate it until convergence or iterations are less than maximum number of iterations. The convergence means the maximization of log-likelihood function [17] as shown in (8).

$$P(w \mid z) = \frac{\sum_d f(d,w) P(z \mid d,w)}{\sum_{d,w'} f(d,w') P(z \mid d,w')'} \qquad (4)$$

$$P(d \mid z) = \frac{\sum_w f(d,w) P(z \mid d,w)}{\sum_{d',w} f(d',w') P(z \mid d',w)'} \qquad (5)$$

$$P(z) = \frac{1}{R} \sum_{d,w} f(d,w) P(z \mid d,w) \qquad (6)$$

Where

$$R \equiv \sum_{d,w} f(d,w) \qquad (7)$$

$$Log - likelihood = \sum_{d \varepsilon D} \sum_{w \varepsilon W} f(d,w) \log(P(d,w) \qquad (8)$$

Where, *f (d, w)* is the frequency of occurrence of word *w* in document *d*.

The output of the Training phase is the probability of finding words in different latent classes, i.e. *P(w|z)* and probability of finding documents in different latent classes, i.e. *P(d |z)*.

Learning Phase*:* In the estimation phase the following steps are followed:

1. Extract the features from q, the potential reusable components and FV is mapped according to occurrence matrix's keyword list.
2. Find different aspects' values in Query Software Components
3. After training, the estimated *P(w|z)* parameters are used to estimate *P(q|z)* for query software components, *q*, through a "folding-in" process[17]. In the "folding-in" process, EM is used in a similar manner to the training process: the E-step is identical, the M-step keeps all the *P(w|z)* constant and only re-calculates *P(q|z)*, which shows the level of different aspects in Query Software Components i.e. DR-value.

Hybrid LSA and Naïve Bayes Classification: The Latent Semantic Analysis (LSA) can be applied to induce and represent aspects of the meaning of English language words [19, 20]. LSA is a variant of the vector space model that converts a representative sample of documents to a term-by-document matrix in which each cell indicates the frequency with which each term (rows) occurs in each document (columns). Thus a document becomes a column vector and can be compared with a user's query represented as a vector of the same dimension. There are four different schemes are evaluated to find DR-value of a software component:

- SVD transformation of data with Similarity Measure
- SDD transformation of data with Similarity Measure
- SVD transformation of data with Naïve Bayes Classification
- SDD transformation of data with Naïve Bayes Classification

The following phases are followed for evaluation:

Construction of Feature Vector (FV) of Domains: The following steps are proposed to find the FV of the different domains using training software components:

1. Extraction of Meta Information: Meta information is collected from the sample software components in form of identifiers/keywords and identifier-by-software matrix is created. The useless identifiers are removed and Normalization is performed.
2. SVD/SDD Transformation: LSA (SVD and SDD [21]) is used for decomposition and Dimensionality Reduction of the features extracted from previous step.

SVD is a form of factor analysis, or more properly, the mathematical generalization of which factor analysis is a special case [19]. It constructs an n-dimensional abstract semantic space in which each original term and each original (and any new) document are represented as vectors. In SVD a rectangular term-by-document matrix X is decomposed into the product of three other matrices W, S, and P^T as shown below:

$$X_K = W\ S\ P^T \tag{9}$$

Where W is a orthonormal matrix and its rows correspond to the rows of X, but it has m columns corresponding to new, specially derived variables such that there is no correlation between any two columns; i.e., each is linearly independent of the others. P is an orthonormal matrix and has columns corresponding to the original columns but m rows composed of derived singular vectors. The third matrix S is an m by m diagonal matrix with non-zero entries (called singular values) only along one central diagonal. A large singular value indicates a large effect of this dimension on the sum squared error of the approximation. The role of these singular values is to relate the scale of the factors in the other two matrices to each other such that when the three components are matrix multiplied, the original matrix is reconstructed.

After the decomposition by SVD, the k most important dimensions (those with the highest singular values in S) are selected as shown in (10). All other factors are omitted, i.e., the other singular values in the diagonal matrix along with the corresponding singular vectors of the other two matrices are deleted. The reduced dimensionality solution then generates a vector of n real values to represent each document. The reduced matrix ideally represents the important and reliable patterns underlying the data in X. It corresponds to a least-squares best approximation to the original matrix X [22].

$$X_K = W_K S_K P_K^T \tag{10}$$

The X_k matrix should now contain the major associational structure in the matrix and has left out the noise. In this reduced model, the overall pattern of term usage determines how close the documents will be located, regardless of the precise words in the documents [23].

The Semi-Discrete Matrix Decomposition (SDD) is similar to the SVD, in that the original matrix is decomposed into three matrices [24] as shown in the following equation:

$$X_K = W_K S_K P_K^T \tag{11}$$

Where matrices W_k and P_k^T contain entries from the set -1, 0 and 1 [21].

3. Naïve Bayes Learning: The SVD and SDD transformed frequency or occurrence tables of the keywords are used for calculating $P(v_j)$ & $P(w_k|v_j)$ values separately according to the equations mentioned in the Naïve Bayes section.

Estimating Domain Relevancy value (DR-value): The following steps are taken to calculate DR-value of a potential reusable Component:

1. Extraction of features from query Component: Features are extracted from the potential reusable component; FV is formed and FV is mapped according to occurrence matrix's keyword list.
2. Perform Similarity Analysis: Similarity analysis between FV of the potential Reusable Component and the FV of different domains is performed and the similarity vector tells the relevancy level with existing domains. Here assumption is taken that the input software might belong to a number of domains with different extent.

In SVD based technique, the query component's similarity with the other components in the repository is measured by calculating the cosine between the vectors, x_k and a query vector, q_k as shown below:

$$S = \tilde{q}^T \tilde{A} \tag{12}$$

Where $\tilde{A} = S_K^{1-\alpha} P_K^T$ (13)

And the query vector is projected into the same k-dimensional space [20] by:

$$\tilde{q} = q^T W_K S_K^{\alpha} \tag{14}$$

The performance of queries generally improves as k increases, but will decrease past a threshold. It is possible for an SVD based system to locate terms which do not even appear in a document. Documents which are located in a similar part of the concept space (i.e. which have a similar meaning) are retrieved, rather than only matching keywords. By using a concept space, following problems can be solved.

1. Polysemy, or the problem that most words have more than one meaning, and that meaning is obtained from the word's context.

2. Synonymy, or the problem that there are many ways of describing the same object. The presence of synonyms tends to decrease the *Recall* performance of Information Retrieval systems [22].

In the SDD based technique, the similarity '*S*' [21] between a document and query vector can be calculated as:

$$S = \tilde{q}^T \tilde{A} \tag{15}$$

Where $\tilde{A} = S_K^{1-\alpha} P_K^T$ (16)

And the query vector is projected into the same k-dimensional space by:

$$\tilde{q} = S_K^{\alpha} W_K^T q \tag{17}$$

In this study, the value of the splitting parameter α in equation has left at the default 0

3. SVD/SDD with Naïve Bayes Evaluation: The performance of the Naïve Bayes Approach is also monitored for classifying the query software components while considering the $P(v_j)$ & $P(w_k|v_j)$ values of the previous section. The results are recorded in terms of *Precision, Recall* and *F-Measure* values as discussed in the nest section.

EVALUATION OF DEVELOPED SYSTEM

It is tried to evaluate the system in terms of *Precision* and *Recall* criteria. Let *S* be a set of all software systems contained in a repository. *Precision* and *Recall* are defined in (18)-(21).

$$Precision = \frac{\sum_{s \varepsilon S} precision_{soft}(s)}{|S|} \tag{18}$$

Where

$$precision_{soft}(s) = \frac{|C_{Actual}(s) \cap C_{Ideal}(s)|}{|C_{Actual}(s)|} \tag{19}$$

And

$$Recall = \frac{\sum_{s \varepsilon S} recall_{soft}(s)}{|S|} \tag{20}$$

Where

$$recall_{soft}(s) = \frac{|C_{Actual}(s) \cap C_{Ideal}(s)|}{|C_{Ideal}(s)|} \tag{21}$$

Where $C_{actual}(s)$ is a set of clusters containing software "s", generated by our software and $C_{Ideal}(s)$ is a set of clusters containing input software "s", determined manually by the Domain Experts. Using *Precision* and *Recall* values F-value is calculated as a measure of performance evaluation i.e.

$$\text{F-Value} = \frac{2pr}{p+r} \tag{22}$$

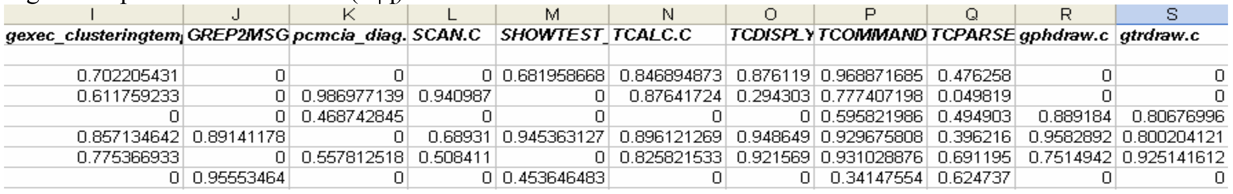
Where, *p* is the *Precision* and *r* is the *Recall* of the system.

IMPLEMENTATION AND RESULTS

As a software implementation of the discussed concept, a deployable Component Object Model (COM) based Component, which is Microsoft's binary standard for object interoperability, is developed. The developed component's objects can be accessible through Visual Basic, C++, or any other language that supports COM. A sample data from various Reusable Repositories of 'C' components is collected and the program is run for the 63 components belonging to six categories or domains (that can be grouped in three main domains/categories) and frequency table is formed with 2942 extracted keywords.

As evidenced from table 1, the Naïve Bayes evaluation phase results show 74.4186% *Accuracy*, 0.1705 Mean Absolute Error (*MAE*) and Root Mean Square Error (*RMSE*) in classifying the software components.

Fig. 1: Snapshot of Calculated P(z |q) values



I	J	K	L	M	N	O	P	Q	R	S
gexec_clusteringtem	*GREP2MSG*	*pcmcia_diag.*	*SCAN.C*	*SHOWTEST*	*TCALC.C*	*TCDISPLY*	*TCOMMAND*	*TCPARSE*	*gphdraw.c*	*gtrdraw.c*
0.702205431	0	0	0	0.681958668	0.846894873	0.876119	0.968871685	0.476258	0	0
0.611759233	0	0.986977139	0.940987	0	0.87641724	0.294303	0.777407198	0.049819	0	0
0	0	0.468742845	0	0	0	0	0.595821986	0.494903	0.889184	0.80676996
0.857134642	0.89141178	0	0.68931	0.945363127	0.896121269	0.948649	0.929675808	0.396216	0.9582892	0.800204121
0.775366933	0	0.557812518	0.508411	0	0.825821533	0.921569	0.931028876	0.691195	0.7514942	0.925141612
0	0.95553464	0	0	0.453646483	0	0	0.34147554	0.624737	0	0

Table 1: Evaluation Phase Statistics of Naïve Bayes

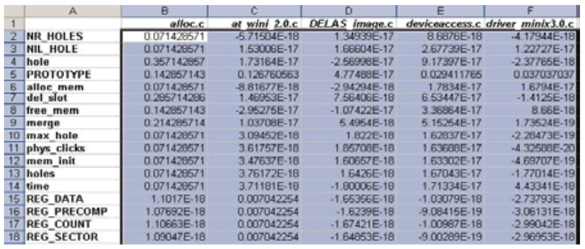
Algorithm	Evaluation Phase Statistics		
	Accuracy (%)	MAE	RMSE
Naïve Bayes	74.4186	0.1705	0.413

The Training phase of the PLSA is run and P(w|z) is calculated in from of frequency matrix. Thereafter, query components are used and P(z|q) is calculated, as shown in Fig. 1. The figure shows different aspect or concept or unobserved latent variable levels found in the software components and these values gives indication of the DR-values of the software component.

Table 2: Detailed results of PLSA based Similarity Measure

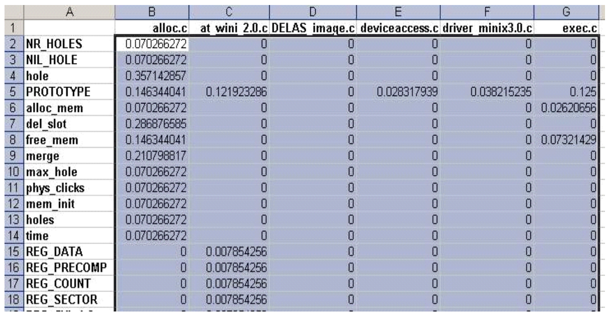
Algorithm	Class Type	Precisi-on	Reca ll	F-Measur e	Accu-racy (%)
PLSA Based Similarity Measure	Class 1	0.6364	0.70 00	0.6667	72.09
	Class 2	0.8000	0.66 67	0.7273	
	Class 3	0.7059	0.80 00	0.7500	

Similarity analysis is performed on the P(z |q) and the query software components are clustered in different clusters according to their latent variable values. The results show 72.09% *Accuracy* for the correct classification of query components. The detailed class-wise results are shown in Table 2.



	A	B	C	D	E	F
1		*alloc.c*	*at_wini_2.0.c*	*DELAS_image.c*	*deviceaccess.c*	*driver_minix3.0.c*
2	NR_HOLES	0.071428571	-5.71504E-18	1.34939E-17	8.6876E-18	-4.17944E-18
3	NIL_HOLE	0.071428571	1.53006E-17	1.66604E-17	2.67739E-17	1.22727E-17
4	hole	0.357142857	1.73164E-17	-2.56998E-17	9.17397E-17	-2.37765E-18
5	PROTOTYPE	0.142857143	0.126760563	4.77488E-17	0.029411765	0.037037037
6	alloc_mem	0.071428571	-8.81677E-18	-2.94294E-18	1.7834E-17	1.6794E-17
7	del_slot	0.285714286	1.46953E-17	7.56406E-18	6.53447E-17	-1.4125E-18
8	free_mem	0.142857143	-2.95275E-17	-1.07422E-17	3.36864E-17	8.66E-18
9	merge	0.214285714	1.03708E-17	5.4954E-18	5.15254E-17	1.73524E-19
10	max_hole	0.071428571	3.09452E-18	1.822E-18	1.62837E-17	-2.28473E-19
11	phys_clicks	0.071428571	3.61757E-18	1.85708E-18	1.63688E-17	-4.32588E-20
12	mem_init	0.071428571	3.47637E-18	1.60657E-18	1.63302E-17	-4.69707E-19
13	holes	0.071428571	3.76172E-18	1.6426E-18	1.67043E-17	-1.77014E-19
14	time	0.071428571	3.71181E-18	-1.80006E-18	1.71334E-17	4.43341E-18
15	REG_DATA	1.1017E-18	0.007042254	-1.65366E-18	-1.03079E-18	-2.73793E-18
16	REG_PRECOMP	1.07692E-18	0.007042254	-1.6239E-18	-9.08415E-19	-3.06131E-18
17	REG_COUNT	1.10663E-18	0.007042254	-1.67421E-18	-1.00987E-18	-2.99042E-18
18	REG_SECTOR	1.09047E-18	0.007042254	-1.64853E-18	-9.00289E-19	-2.96953E-18

Fig. 2: Snapshot of Occurrence Matrix formed after the SVD decomposition



	A	B	C	D	E	F	G
1		alloc.c	at_wini_2.0.c	DELAS_image.c	deviceaccess.c	driver_minix3.0.c	exec.c
2	NR_HOLES	0.070266272	0	0	0	0	0
3	NIL_HOLE	0.070266272	0	0	0	0	0
4	hole	0.357142857	0	0	0	0	0
5	PROTOTYPE	0.146344041	0.121923286	0	0.028317939	0.038215235	0.125
6	alloc_mem	0.070266272	0	0	0	0	0.02620656
7	del_slot	0.286876585	0	0	0	0	0
8	free_mem	0.146344041	0	0	0	0	0.07321429
9	merge	0.210798817	0	0	0	0	0
10	max_hole	0.070266272	0	0	0	0	0
11	phys_clicks	0.070266272	0	0	0	0	0
12	mem_init	0.070266272	0	0	0	0	0
13	holes	0.070266272	0	0	0	0	0
14	time	0.070266272	0	0	0	0	0
15	REG_DATA	0	0.007854256	0	0	0	0
16	REG_PRECOMP	0	0.007854256	0	0	0	0
17	REG_COUNT	0	0.007854256	0	0	0	0
18	REG_SECTOR	0	0.007854256	0	0	0	0

Fig. 3: Snapshot of Occurrence Matrix formed after the SDD decomposition

When the SVD and SDD Similarity Measure based Domain-Relevancy module is run to determine DR-value of the query software component then the results of shows 62.4 % *Accuracy* in both cases as shown in table 3 with best *F-Measure* value of 0.7097, but the space complexity of SDD is less as compared to SVD technique.

Table 3: Detailed Results of SVD and SDD Based Similarity Measures

Algorithm	Class Type	Precisi-on	Recall	F-Measure	% Acc-uracy
SVD Transform and Similarity Measure	Class 1	0.7500	0.3000	0.4286	62.4
	Class 2	0.8462	0.6111	0.7097	
	Class 3	0.5385	0.9333	0.6829	
SDD Transform and Similarity Measure	Class 1	0.7500	0.3000	0.4286	62.4
	Class 2	0.846 2	0.611 1	0.7097	
	Class 3	0.538 5	0.933 3	0.6829	

In the hybrid scheme of SVD/SDD with Naïve Bayes Classification is applied, the SVD with Naive based scheme shows better results as compared to its counterpart SDD with Naïve Bayes scheme as shown in Table 4.

Table 4: Detailed Results of SVD and SDD Based Similarity

Scheme	Evaluation Phase Statistics		
	Accuracy %	MAE	RMSE
SVD Transformation with Naïve Bayes Classification	76.7442	0.155	0.3937
SDD Transformation with Naïve Bayes Classification	74.4186	0.1705	0.413

The detailed results of the SVD Transformation with Naïve Bayes Classification results are shown in table 5 with 76.7442 % *Accuracy,* 0.833 best *Precision* and 0.889, 0.889 best *Recall* and 0.842 best *F-Measure* values.

Table 5: Detailed results of SVD Transformation with Naïve Bayes scheme

Class Type	Precision	Recall	F-Measure	Accuracy (%)
Class 1	0.636	0.7	0.667	76.7442
Class 2	0.8	0.889	0.842	
Class 3	0.833	0.667	0.741	

CONCLUSION

The PLSA based software categorization approach provides better results than purely LSA based retrieval techniques in terms of *Precision* and *Recall* but its time complexity is too high. At the same level of dimension the categorization results of the SDD are similar to that of results of SVD technique, but SDD produced *Precision* rates similar to SVD with less storage. However the SDD decomposition requires more time to decompose the original matrix., and requires a higher dimension than SVD. It is found that the SDD provided a significantly higher average *Precision* than the SVD if the same query time was required. In order to match the SDD query speed, a much lower dimension must be used for the SVD. The pure and hybrid Naïve Bayes approach is found better performer than the PLSA and LSA based approaches. In the hybrid approaches the SVD Transformation with Naïve Bayes scheme has outperformed all other approaches and shows better results than the existing approach (LSA) being used by some open source code repositories e.g. Sourceforge. The categorization results are close to the manual analysis, used to be performed by the programmers/repository managers. Hence, the developed tool can be also be utilised for the automatic categorization of software components and domain-relevancy of software components. Ultimately, this kind of automation may improve the productivity and quality of software development.

REFERENCES

1. Smith, E., A. Al-Yasiri and M. Merabti, 1998. A Multi-Tiered Classification Scheme For Component Retrieval. Euromicro Conference, 24(2): 882 – 889.
2. Basili, V. R., 1989. Software Development: A Paradigm for the Future. Proc. COMPAC ‘89, Los Alamitos, Calif.: IEEE CS Press, pp: 471-485.
3. Boehm, B. W., 1988. A Spiral Model of Software Development and Enhancement. IEEE Computer, 21(5): 61- 72.
4. Griss, M. L. and M. Wosser, 1995. Making reuse work at Hewlett-Packard. IEEE Software, 12(1): 105 - 107.
5. Succi, G., C. Uhrik and M. Ronchetti, 1996. Reusability and Portability of Logic Programming. Journal of Programming Languages Design, Chapman & Hall, 4(2): 101-114.
6. Boehm, B., 1999. Managing Software Productivity and Reuse. IEEE Computer, 32(9): 111 - 113.
7. Joos, R., 1994. Software Reuse at Motorola. IEEE Software, 11(5): 42-47.
8. Lim, W., 1994. Effects of Reuse on Quality, Productivity, and Economics. IEEE Software, 11(5): 23-30.
9. Ahrens, J. D. and N. S. Prywes, 1995. Transition to a legacy- and reuse-based software life cycle. IEEE Computer, 8(10): 27 - 36.
10. Caldiera, G. and V. R. Basili, 1991. Identifying and Qualifying Reusable Software Components. IEEE Computer, pp .61-70.
11. Tracz, W., 1991. A Conceptual Model for Megaprogramming. SIGSOFT Software Engineering Notes, 16(3): 36-45.
12. Poulin, J. S., 1997. Measuring Software Reuse– Principles, Practices and Economic Models, Addison-Wesley Publishers.
13. Price, Margaretha W., S. A. Demurjian and D. M. Needham, 1997. A Reusability Measurement Framework and Tool For Ada 95. Conference on TRI-Ada '97.

14. Kawaguchi, S., P. K. Garg, M. Matsushita and K. Inoue, 2003. Automatic categorization algorithm for evolvable software archive Software Evolution. Sixth International Workshop on Principles of Software Evolution (IWPSE'03), pp: 195 – 200.
15. Kawaguchi, S., P. K. Garg, M. Matsushita and K. Inoue, 2004. MUDABlue: an automatic categorization system for open source repositories. 11th Asia-Pacific Software Engineering Conference (2004), pp: 184 – 193.
16. Mitchell, T., 1997. Machine Learning. McGraw Hill Publishers, 2nd Ed.
17. Hofmann T., 1999. Probabilistic latent semantic indexing. Proc. of SIGIR'99.
18. Gildea, D. and T. Hofmann, 1999. Topic Based Language Models Using EM. 6th European Conference On Speech Communication and Technology (Eurospeech'99, 1999), pp: 2167-2170.
19. Berry, M., S.T. Dumais and G. W. O'Brien, 1995. Using Linear Algebra For Intelligent Information Retrieval. SIAM: Review, 37(4): 573-595.
20. Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, 1990. Indexing By Latent Semantic Analysis. Journal of the American Society For Information Science, 41: 391-407.
21. Kolda, T., 1997. Limited-Memory Matrix Methods with Applications. Ph.D. thesis, University of Maryland at College Park, Applied Mathematics Program (1997).
22. Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, 1990. Indexing By Latent Semantic Analysis. Journal of the American Society For Information Science, 41: 391-407.
23. Dumais, S. T., 1992. LSI meets TREC: A status report. Text Retrieval Conference, pp: 137-152.
24. Kise, K., M. Junker, A. Dengel and K. Matsumoto, 2001. Experimental evaluation of passage-based document retrieval. 6th International Conference on Document Analysis and Recognition, pp: 592-596.