Thesis Draft

Rubei Riccardo

April 17, 2018

Contents

1 Introduction 3
1.1 Overview . . . 3

2 The Similarity Problem 3
2.1 Overview . . . 3
2.2 Mathematical Background . . . 3
2.2.1 Term-Document Matrix . . . 3
2.2.2 Cosine Similarity . . . 4
2.2.3 Singular Value Decomposition . . . 5
2.2.4 Latent Semantic Analysis . . . 5

3 MUDABlue 6
3.1 Overview . . . 6
3.2 The Approach . . . 6
3.2.1 Exctract Identifiers . . . 6
3.2.2 Create identifier-by-software matrix . . . 6
3.2.3 Remove useless identifiers . . . 7
3.2.4 Apply the LSA . . . 7
3.2.5 Apply the Cosine Similarity . . . 7
3.2.6 Categorization . . . 7

1 Introduction

1.1 Overview

A general introduction of Crossminer, my work and how it contributes.

2 The Similarity Problem

2.1 Overview

How to solve the problem of similarity between documents, subsection about SVD, LSI and other techniques.

2.2 Mathematical Background

2.2.1 Term-Document Matrix

In Natural Language Processing [**?**], a term-document matrix (TDM) is used to represent the relationships between words and documents [**?**]. In a TDM, each row corresponds to a document and each column corresponds to a term. A cell in the TDM represents the weight of a term in a document. The most common weighting scheme used in document retrieval is the *term frequency-inverse document frequency (tf-idf)* function [**?**]. If we consider a set of n documents $D = (d_1, d_2, .., d_n)$ and a set of terms $t = (t_1, t_2, .., t_r)$ then the representation of a document $d \in D$ is vector $\vec{\delta} = (w_1^d, w_2^d, .., w_r^d)$, where the weight w_k^d of term k in document d is computed using the *tf-idf* function [**?**]:

$$w_k^d = tf \cdot idf(k, d, D) = f_k^d \cdot log \frac{n}{|\{d \in D : t_k \in d\}|} \quad (1)$$

where f_k^d is the frequency of term t_k in document d.

Another common weighting scheme uses only the frequency of terms in documents for cells in TDM, i.e. the number of occurrence of a term in a document, instead of *tf-idf*. As an example, we consider a set of three simple documents $D = (d_1, d_2, d_3)$ as follows:

+ d_1: *She is nice.*
+ d_2: *Today is nice.*
+ d_3: *Nice is a nice city.*

The set of terms t consists of 6 elements, i.e. $t = (she, is, today, a, nice, city)$ and the corresponding term-document matrix for D is depicted in Figure 1.

TDM has been exploited to characterize software systems and finally to compute similarities between them [**?**],[**?**],[**?**]. In a TDM for software systems, each row represents a package, an API call or a function and each column represents a software system. A cell in the matrix is the number of occurrence of a package/an API/function in each

$$\begin{array}{c} \\ d_1 \\ d_2 \\ d_3 \end{array} \begin{array}{c} \begin{array}{cccccc} she & is & today & a & nice & city \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 2 & 1 \end{pmatrix} \end{array} \tag{2}$$

Figure 1: An example of a term-document matrix

corresponding software system. A TDM for software systems has a similar form to the matrix shown in Figure 1 where documents are replaced by software systems and terms are replaced by API calls.

2.2.2 Cosine Similarity

Cosine similarity is a metric used to compute similarity between two objects using their feature vectors [**?**]. An object is characterized as a vector, and for a pair of vectors $\vec{\alpha} = (\alpha_1, \alpha_2, .., \alpha_n)$ and $\vec{\beta} = (\beta_1, \beta_2, .., \beta_n)$ there is an angle between them. Intuitively, the cosine similarity metric measures the similarity as the cosine of the corresponding angle between the two vectors and it is computed using the inner product as follows.

$$CosineSim(\vec{\alpha}, \vec{\beta}) = \frac{\sum_{i=1}^{n} \alpha_i \cdot \beta_i}{\sqrt{\sum_{i=1}^{n} (\alpha_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (\beta_i)^2}} \tag{3}$$

Figure 2 illustrates the cosine similarity between two vectors $\vec{\alpha}$ and $\vec{\beta}$ in a three-dimension space. This can be thought as the similarity between two documents with three terms $t = (t_1, t_2, t_3)$.

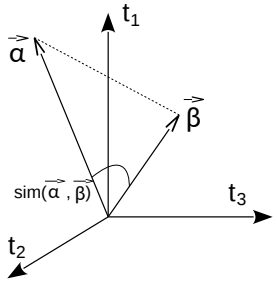

Figure 2: Cosine similarity between two feature vectors $\vec{\alpha}$ and $\vec{\beta}$

Cosine similarity has been popularly adopted in many applications that are related to similarity measurement in various domains [**?**],[**?**],[**?**],[**?**],[**?**]. Among the similarity metrics being recalled in this deliverable, the prevalence of Cosine Similarity is obvious as it is utilized in almost all of them as follows: *MUDABlue* [**?**], *CLAN* [**?**], *CLANdroid* [**?**], *LibRec* [**?**], *SimApp* [**?**], *WuKong* [**?**], *TagSim* [**?**], and *RepoPal* [**?**]

2.2.3 Singular Value Decomposition

2.2.4 Latent Semantic Analysis

3 MUDABlue

3.1 Overview

The first procedure analysed was MUDABlue, unfortunately none implentation was available on the web, so i reimplemented it from scratch. The MUDABlue method is an automatc categorizaton method or a large collecton of software systems. MUDABlue method does not only categorize sooware systemsd but also determines categories rom the sooware systems collecton automatcally. MUDABlue has three major aspects: 1) it relies on no other information than the source code, 2) it determines category sets automatically, and 3) it allows a software system to be a member of multiple categories. Since we were interested only in the evaluation of the similarity we discarded the phases related to clusterization and categorization.

3.2 The Approach

The MUDABlue approach can be briefly summarized in 7 steps, as the following image depicts:

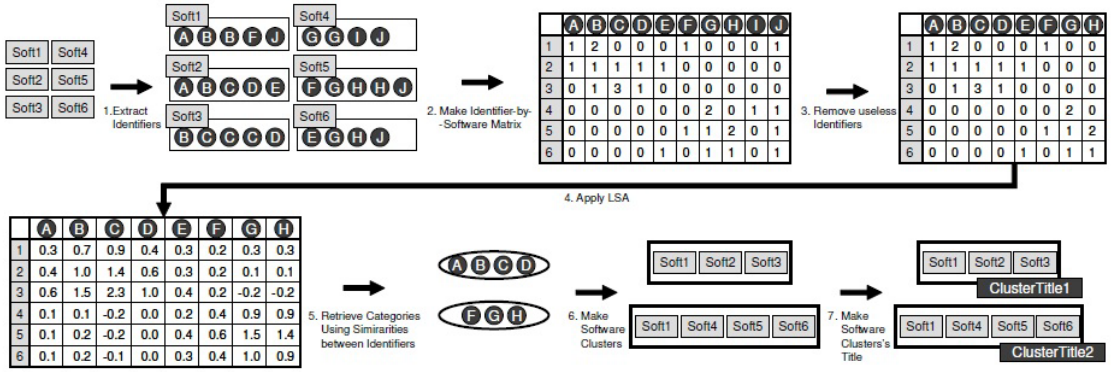

Figure 3: MUDABlue phases.

3.2.1 Exctract Identifiers

With identifier we are talking about relevant strings that can allow to characterize a document. In this phase each repository is scanned in order to find the target files, and for each of them the identifiers are exctracted, avoiding adding useless items such as comments. The dataset was a 41C projects gathered from SourceForge.

3.2.2 Create identifier-by-software matrix

As stated before, the main item to work with is the term-document matrix, in this case we count how many times each term appears in each file for all the projects. The result is matrix **m x n** with m terms and n projects.

3.2.3 Remove useless identifiers

From the matrix we remove all the useless terms, that is all the terms that apperas in just one repository, considered a specific terms, and all the terms that appears in more than 50% of the repositories, considered as general terms.

3.2.4 Apply the LSA

Once the matrix is ready con be worked, the SVD procedure is applied and then the LSI. As explained before [NOTE] the SVD procedure decompose the original matrix in 3 other matrices. When we multiply back these matrices we use a rank reducted version of the S matrix in order to generete the final one. The authors didn't provide us any details about their final rank value, so we tested many values and eventually selected one.

3.2.5 Apply the Cosine Similarity

By using the cosine similarity method, we compare each repository vector with all the others and eventually getting an **n x n** matrix, in which is expressed the similarity of all the repository couple, with a value [0.0-1.0].

3.2.6 Categorization

The point 6 and 7 are not covered because not related to our work.